A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network

Edmund H. Durfee

COINS Technical Report 87-84
September 1987

This research was sponsored, in part, by the National Science Foundation under Grant MCS-8306327, by the National Science Foundation under Support and Maintenance Grant DCR-8318776, by the National Science Foundation under CER Grant DCR-8500332, and by the Defense Advanced Research Projects Agency (DOD), monitored by the Office of Naval Research under Contract N00014-79-C-0439. The author also received support from an IBM Graduate Fellowship and by a University of Massachusetts Graduate Fellowship.

A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network

A Dissertation Presented

By

Edmund Howell Durfee

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

Doctor Of Philosophy

September 1987

Computer and Information Science

This research was sponsored, in part, by the National Science Foundation under Grant MCS-8306327, by the National Science Foundation under Support and Maintenance Grant DCR-8318776, by the National Science Foundation under CER Grant DCR-8500332, and by the Defense Advanced Research Projects Agency (DOD), monitored by the Office of Naval Research under Contract NR049-041. Edmund Durfee was also partially supported in the 1985-6 academic year by a University Fellowship sponsored by the University of Massachusetts, and in the 1986-7 academic year by an IBM Graduate Fellowship.

A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network

September 1987

Edmund Howell Durfee

A.B., Harvard University

M.S.E.C.E., Ph.D., University of Massachusetts

Directed by: Professor Victor R. Lesser

Committee: Prof. Victor R. Lesser (Chair)
Dr. Daniel D. Corkill
Prof. Krithivasan Ramamritham
Dr. Reid G. Smith

Acknowledgments

I remember telling Victor, a couple years ago, that I did *not* want to do planning. I wanted to work on distributed stuff, on control and cooperation. Victor didn't argue, and I think it was because he knew, as I know now, that the research has a momentum of its own. If we knew where the research would lead, there would be no point in following it. As it turns out, mine led back to issues in planning—*and* to control and cooperation.

I'd like to thank Victor Lesser, my advisor, for all of his help as I pursued that research. He has led me sometimes, followed me other times, encouraged me to keep going, applauded my successes, asked tough questions when I thought I had all the answers, supplied insights when I got stuck, greeted my insights with delight, and throughout has radiated a contagious enthusiasm for this research. It has been a pleasure to have collaborated with him.

I also owe a great debt to Dan Corkill, another of my committee members. It was he who introduced me to the implementation and simulation of distributed problem solving networks. Because he shared so much of his practical knowledge and expertise with me, I was much better prepared to turn my own ideas into realities. Dan's conceptual insights, moreover, have continually helped me recognize interesting and promising research issues in distributed problem solving.

My other committee members have made important contributions to this research. Krithi Ramamritham helped open my eyes to the ties between distributed AI and distributed operating systems. His interest in my work as it applies to other aspects of computer systems has encouraged me to consider some of the more general aspects of coordinating cooperating computers. Reid Smith has provided an alternative view, since he has approached distributed problem solving from a different angle. His questions and comments have helped me see relationships between different approaches to control and coordination, and have led me to better understand the limitations and costs of my approach, where I might have otherwise not looked beyond its benefits.

The past and present members of the distributed problem solving lab here at UMass have helped this research and researcher in many different ways, and I would like to thank them all. In particular, Joe Hernandez and I have had

many brainstorming (and other types of) discussions which have helped me better understand issues and channel my energies. Others that have contributed ideas, criticisms, or technical help to this research are Joe Walters, Dave Hildum, Anil Rewari, and Dave Westbrook (and Teri Westbrook often provided dinners that helped break up the monotony of my typical bachelor fare).

Many others inside and outside of the UMass COINS community deserve thanks as well. Among those who have had an influence on my work here at UMass are Jim Kurose, Jack Stankovic, Peter Bates, Paul Cohen, and Michael Arbib. Discussions with other DAI researchers, especially Les Gasser of USC, have also helped me mature my ideas. I'd also like to thank Michele Roberts for her administrative help so that I could more fully concentrate on my research. The many people that maintain the computer facilities here deserve credit as well for providing a relatively reliable and stable environment in which to compute. Finally, I'd also like to acknowledge the financial assistance of the sponsors of this research, and thank both IBM and the University of Massachusetts for supporting me personally through fellowships.

Many others have helped and encouraged me over the years, even though not directly with this work. I'd like to thank Ed Heideman, Mr. V, and Fred Snyder for many lessons in science, in self-discipline, and in teamwork. I'd also like to thank the friends and colleagues I had while I was a chemist at GE. Although I wasn't cut out to be a chemist, I learned invaluable lessons in self-motivation and self-organization, especially from George Loucks.

Thanks to my other friends who, though working in unrelated areas, have helped me in their own ways. These include Jane and Nigel, the Esch, and Blaise. I'd also like to thank Mary, who has reminded me of the other fine things in life (besides computers), who has inspired me in many ways to complete my degree, and who has made this difficult process much more bearable.

My family has helped me throughout. I'd like to thank my sister Sue and her family: Sue for wonderful meals and her cheerful outlook on life; her husband David for bike rides and for helping me keep my Volvo running through most of my graduate career; and the nieces for hours of play and entertainment that distracted me, for a while, from the other demands I faced. My brother Ken, as the other computer professional in the family, has helped by being someone that does not look completely baffled when I explain what I do at family get-togethers. My sister Rachel, by being an undergraduate while I've been in school, has helped me avoid being the only student in the family, so I wasn't alone in being accused of not having a "real" job.

Finally, I'd like to thank my parents for more things than I can mention. They have always been there, supporting and encouraging me; and though they have not pushed me, somehow their influence makes me want to do my best and make them proud. Thanks, folks.

Abstract

As distributed computing is used in applications where the distributed tasks are highly interrelated and change dynamically, coordination becomes increasingly important and difficult. Distributed artificial intelligence (DAI) applications often have these characteristics. In distributed problem solving networks, for example, individual *nodes* solve interacting subproblems of larger network problems. Network problems may change over time, so effective network problem solving depends on nodes coordinating their local actions and planning their interactions to cooperate as a coherent team.

We introduce *partial global planning* as a new approach to coordination. Whereas previous DAI approaches, such as contracting or multi-agent planning, are specialized for particular situations, our new partial global planning approach provides a unified and versatile framework for dynamically coordinating independent nodes. The approach views control as a planning task, where nodes develop local plans and then exchange summaries of their plans to develop network awareness. When they see that they have pieces of larger goals, nodes form and follow *partial global plans* that specify cooperative actions and interactions. Since a node's plans can change and communication takes time, nodes may base decisions on incomplete, out-of-date, and inconsistent views of the network. Our partial global planning approach lets nodes plan satisfactory, though not necessarily optimal, cooperation even in dynamic environments.

This dissertation describes partial global planning and details its implementation in a simulated distributed problem solving network for vehicle monitoring. One major phase of the research is building individual nodes that can plan for control. We present a new local planner and evaluate its effects on control decisions and overhead. The other major phase is developing mechanisms for communicating about plans and forming partial global plans. We describe these mechanisms

in detail and experimentally examine their benefits in improving coordination, their costs in additional overhead, and their versatility in allowing cooperation in different styles to achieve a variety of goals. Our research shows that partial global planning is a flexible and practical approach for coordinating problem solving networks (and distributed computing systems in general) and we outline future directions for improving and extending our approach.

Table of Contents

List of Tables

List of Figures

The controlling intelligence understands its own nature, and what it does, and whereon it works. -Marcus Aurelius

Chapter 1

Overview

Distributed computing systems have several advantages over centralized, monolithic computing systems. They are often more cost effective, because n simple processors may be considerably cheaper than a single processor that is n times as powerful. A group of processors may also be more reliable, since the failure of some of these processors only reduces the computing power, while if the single powerful processor fails then all computing power is lost. In addition, many computing tasks are inherently distributed. The tasks may be spatially distributed, such as tasks to monitor different spatial areas or tasks to process transactions at different branches of a bank. The tasks may also be functionally distributed, such as tasks to perform low level processing of visual and tactile data, where processors with different functional capability might be used to process different types of data. These advantages have led to distributed computing systems being applied in increasingly complex domains.

As distributed computing systems are used in more diverse applications, however, the difficulties in controlling these systems become more pronounced. Not only must an individual computing element make control decisions about how it should most effectively apply its local resources, but the computing elements as a group must also coordinate their control decisions to effectively use net-

work resources. Computing elements are more capable of helping or hindering each other when the interdependencies between their tasks are more complex, so distributed computing applications involving highly interdependent tasks require sophisticated control mechanisms to promote effective cooperation. This research develops control mechanisms for distributed computing systems with the following characteristics:

- The tasks performed in the network are highly interdependent.
- The tasks performed in the network can be grouped together based on their relationships, where a group of tasks leads to a single larger result.
- The computing elements can *roughly* predict the time (and other resource) needs of tasks and groups of tasks.
- The network situation is sufficiently predictable for computing elements to *tentatively* plan sequences of related tasks.
- The communication delays are of the same order of magnitude as the task times, so that computing elements can communicate about planned actions ahead of time, but synchronizing decisions about current activities is not feasible.

Distributed problem solving networks often display these characteristics. In a distributed problem solving network, the computing elements are problem solving *nodes* that are cooperatively trying to solve problems by individually solving subproblems and integrating subproblem solutions into overall solutions. These networks are typically used in applications such as distributed sensor networks [Lesser and Erman, 1980; Smith, 1980], distributed air traffic control [Cammarata *et al.*, 1983], and distributed robot systems [Fehling and Erman, 1983], where there is a natural spatial distribution of information but where each node has insufficient local information to completely and accurately solve its subproblems. The performance of the distributed problem solving network depends on the control decisions that nodes make about their local actions (to solve relevant subproblems in a timely manner) and about their interactions (to share useful information and converge on overall solutions). The nodes pool their resources and expertise, work in parallel on different parts of a problem to solve it faster, avoid harmful

interactions such as resource conflicts or working at cross-purposes, and promote helpful interactions such as moving information to where it is most needed or tasks to where they can best be performed.

To make coordination decisions, the nodes need knowledge about the *goals of cooperation*: whether they should avoid redundant work (to save on computation) or promote this work (to verify each other's results); whether they should assign important tasks to multiple nodes (in case those nodes fail) or should avoid this (so that nodes do not unnecessarily duplicate each other's activities); whether they should exchange predictive partial results (to focus each other on forming compatible results) or not (to avoid overly influencing each other). In complex domains, the goals of cooperation may differ as circumstances change, so mechanisms for coordination must be flexible for nodes to achieve appropriate goals of cooperation in a given situation.

Similarly, the nodes' *style of cooperation* depends on the problem domain and on the environmental characteristics. Sometimes the nodes should channel information about their local activities to "controller" nodes that make coordination decisions for the network. At other times, the nodes might broadcast this information to each other and individually form more global views about how their local activities fit into network problem solving. When communication is very expensive or integrating information into more global views is costly, however, nodes should work relatively independently and selectively exchange their local solutions to converge on global solutions. Alternatively, they may negotiate in small groups to contract out tasks in the network. To work in a wide variety of situations, the coordination mechanisms must be flexible enough for the nodes to coordinate in different styles.

The focus of this research is on developing, implementing, and evaluating a new approach for coordinating cooperating problem solvers (and, more generally, computing elements with interdependent tasks). This approach, called partial global planning, is significant because it provides a unified framework where nodes can achieve various goals of cooperation using different styles of cooperation. In the next section, we introduce the partial global planning approach to coordination, citing its contributions and our methodology in developing it. Subsequently, we discuss major research issues in more detail: the role of local control sophistication, the need to meet different goals of cooperation, the importance

of cooperating in different styles, and the difficulties in balancing predictability and responsiveness in dynamic domains. We then contrast our work to previous research in distributed control and artificial intelligence. In the final section, we provide an overview of the remainder of this dissertation, including suggestions about how the reader can find the information in which he or she is interested.

1.1 Partial Global Planning: A Unified Approach to Dynamic Coordination

Partial global planning is a new approach to coordination in which cooperating nodes explicitly plan their actions and interactions in complex and dynamically changing situations. Through partial global planning, each node can represent and reason about the activities of groups of nodes and about how local activities fit into this more global view. These representations are called *partial global plans* because they specify how different *parts* of the network (subsets of nodes) *plan* to work together to achieve more *global* goals. A partial global plan thus indicates the concurrent actions of individual nodes (planned sequences of actions each node should pursue) and the important interactions between nodes (planned communication of partial solutions to help other nodes perform their tasks or build more complete solutions). Each node maintains a set of partial global plans that represents its own local view of network activity, and the nodes can exchange information about their local and partial global plans to improve each other's view.

Different problem situations may call for different styles of cooperation, and partial global planning allows versatility in how nodes coordinate because nodes may exchange and reason about plan information in a variety of ways. Because the network may be capable of achieving only some of its objectives, the approach also incorporates flexibility in how the network balances different goals of cooperation. Finally, coordination in dynamic, uncertain, distributed domains must be based on possibly inaccurate, incomplete, and out-of-date information. Partial global planning allows cooperating nodes to use whatever information they have and to spend whatever computing time they can afford in order to dynamically plan satisfactory (not necessarily optimal) actions and interactions to meet network goals.

The remainder of this section describes the contributions of our partial global planning research from conceptual, technical, and empirical perspectives, and then outlines our research methodology.

1.1.1 Conceptual Contributions

Conceptually, partial global planning is at the same time both simple to understand and versatile. It is simple to understand because it is so familiar—it resembles coordination among humans: (1) each node determines its own goals and forms plans to achieve them; (2) nodes exchange information to identify cases where their goals and plans interact; and (3) nodes change their planned local actions and interactions to coordinate better based on their view of group activity. It is versatile because it has the flexibility to coordinate nodes in a variety of situations: individual nodes are free to form local goals and plans based on their local knowledge and priorities, and the nodes' coordination responsibilities and their criteria for choosing what plans to communicate can be changed to modify how (and whether) nodes decide to coordinate. Whereas previous approaches to coordination, such as contracting [Smith, 1980] and multi-agent planning [Cammarata *et al.*, 1983], are specialized for particular situations (as is discussed in Section 1.3), partial global planning consolidates these different approaches into a unified and versatile framework for dynamically coordinating independent nodes.

The flexibility of partial global planning means that nodes can coordinate in many different ways, but this flexibility can lead to incoherent network behavior when nodes have different views of how they should go about developing and using partial global plans. An important new concept introduced in this research is the use of an explicit *meta-level organization* to structure how nodes form, exchange, manipulate, and react to partial global plans. Unlike a *domain-level organization* that nodes use to guide problem solving [Corkill and Lesser, 1983], the meta-level organization guides coordination. Partial global planning reduces the need for relatively static, domain-level organizations because information to coordinate problem solving is dynamically formed and represented in partial global plans. Instead, the network needs a relatively static, meta-level organization to delimit coordination roles and responsibilities. By representing the meta-level organization explicitly and declaratively, we allow the partial global planning mechanisms to be used appropriately for different situations to achieve effective cooperation.

Partial global planning coordinates nodes working in dynamic situations, and this also leads to important conceptual contributions. Because nodes cannot fully predict their future situations, their local plans are tentative and uncertain. Individual nodes must be able to resolve uncertainty about long-term goals to pursue, must monitor and repair plans, must modify plans in response to changing circumstances, and often must meet deadlines. Our local planning framework that underlies partial global planning is a conceptually new approach for controlling problem solving in dynamic and uncertain domains.

Because local plans are tentative, so are partial global plans. The conceptual development and experimental evaluation of partial global planning provide initial answers to important questions about how nodes should coordinate in a tentative, uncertain domain. We explore conceptual issues such as how much detail about their plans nodes should exchange and when a change to a local plan is worth communicating about. In addition, we examine the ramifications of frequently modifying and communicating about plans in dynamic situations. We study issues in how and when nodes should build more resilient partial global plans, and how they can build partial global plans that assign tasks to better exploit network resources. Our work also explores the complexities involved in coordinating larger networks, and how the complexity be reduced. Most importantly, we recognize that local and partial global planning are tools that are more or less cost effective depending on the situation: although useful in a wide range of situations, they do introduce overhead that makes them impractical in either very simple situations (where cheaper and less sophisticated control mechanisms would suffice) or in very ambiguous, complicated situations (where nodes either cannot distinguish promising plans or else the cost of forming useful plans is prohibitively large).

Partial global planning also makes reasoning about coordination an integral part of a node's activity. Unlike approaches where coordination mechanisms are added as an afterthought to stand-alone systems, partial global planning recognizes that, to cooperate effectively, nodes need to reason in depth about network activity when making local control decisions. Partial global planning thus represents an important conceptual contribution in that it integrates local and network control into a single control structure: nodes are essentially independent and determine their own tentative goals and plans, and yet they can exchange information about their plans to develop and use a more global view of the network.

The result is a new understanding about how local awareness at nodes can lead to better network awareness among nodes, how local and network awareness call for representing and reasoning about actions and interactions at multiple levels of detail, and how using this awareness can improve coordination.

More generally, partial global planning has implications for control in other types of distributed systems as well. Our research indicates that effective cooperation among computing nodes requires that each node be capable of representing and reasoning about network activity and that this reasoning be an integral part of its local decisionmaking. Even if the computing nodes are not knowledge-based systems, partial global planning techniques could still be incorporated into their control mechanisms to improve how they coordinate when working on highly interdependent tasks.

1.1.2 Technical Contributions

To evaluate the partial global planning approach from a practical as well as a conceptual viewpoint, we have implemented the approach as part of a distributed problem solving network working in a non-trivial and dynamic task domain—the vehicle monitoring domain. In the course of this implementation, we have encountered many technical issues and obstacles that could be overlooked in a more speculative, conceptual treatment. As a result, a large proportion of this dissertation is devoted to the more technical aspects of how the partial global planning approach can be implemented into specific control mechanisms. The technical details contribute to our understanding of partial global planning and provide a basis for implementing the approach in other domains. In the rest of this section, we provide an overview of the actual mechanisms to indicate the important technical issues that our research addresses.

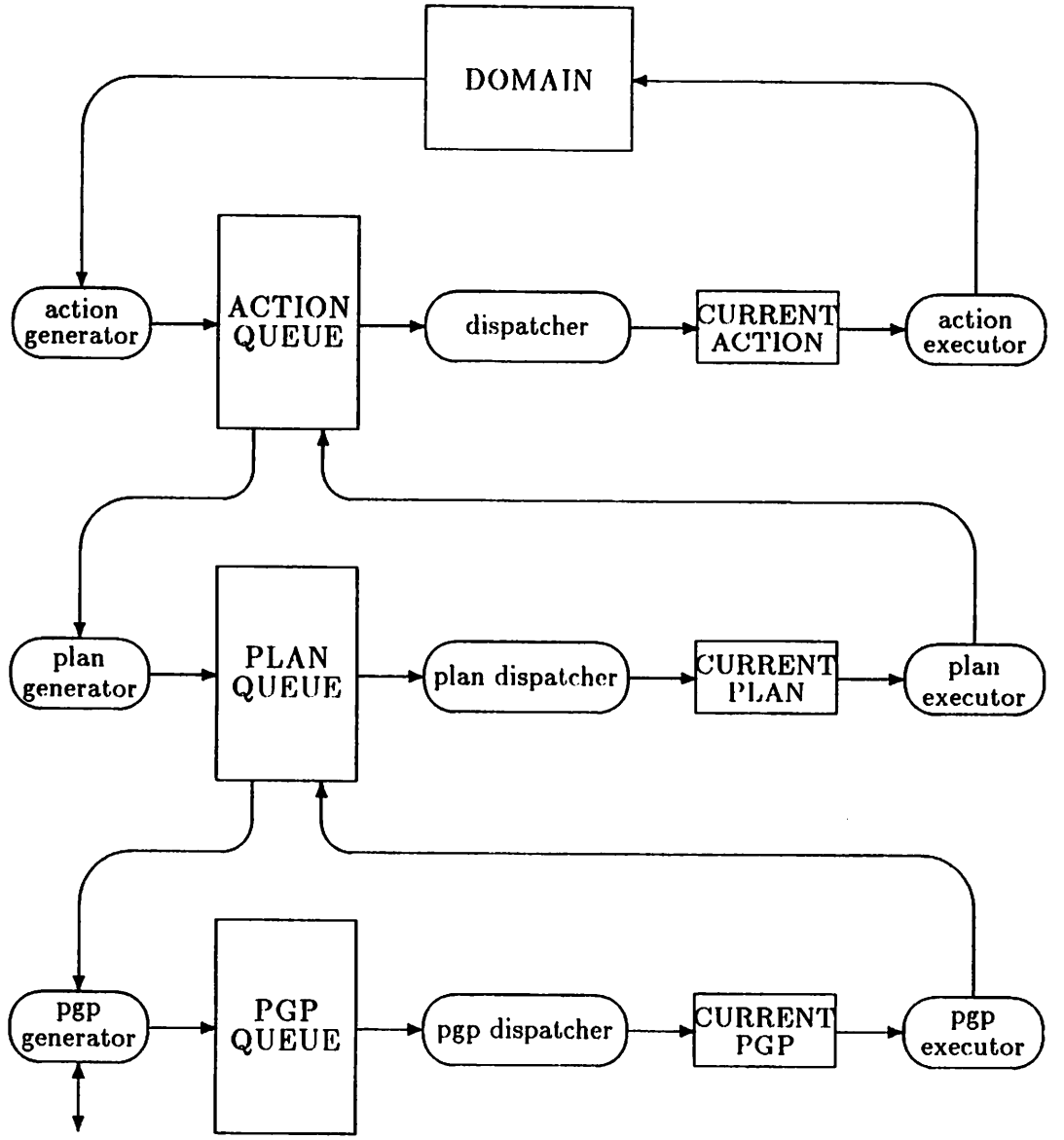
The technical development has two main thrusts. One thrust is developing local planning mechanisms, because before it can cooperate with others a node must first identify its local goals and possible actions to achieve those goals. That is, to determine how it fits into more global activities, a node must have a certain amount of *self-awareness*. Subsequent chapters are devoted to describing how nodes recognize local goals (Chapter 3) and build plans for achieving them (Chapter 4). The other thrust is developing partial global planning mechanisms. Given self-awareness of its own plans and, through communication about plans,

network awareness about the plans of others, a node must reason about the plans and the goals of cooperation to develop partial global plans to coordinate group activity. In later chapters, we describe how nodes develop greater network awareness (Chapter 6) and use it to form partial global plans (Chapter 7).

An important assumption behind our mechanisms is that nodes can plan their local future actions and their future interactions, and that although situations may change over time, the changes are not so fast and unpredictable that planning is not worthwhile. In some domains, nodes cannot plan their activities, plans are obsolete before significant progress is made on them, or the overhead expended on local and partial global planning is excessive. Partial global planning should not be applied when nodes would be better off using a relatively static, domain-level organization to coordinate in a more general manner rather than attempting to dynamically coordinate for specific situations. Our mechanisms allow nodes to resort to domain-level organizations in such situations.

The basic components of our new mechanisms are shown in Figure 1. Initially, we begin with a problem solver with a queue of possible pending actions (tasks), a dispatcher that chooses the next action to take, and an executor that takes that action. The action might cause changes to the problem state that trigger new actions to be placed on the queue of actions. To the problem solver we now add a local planner that plans sequences of actions and associates any pending actions with their corresponding plans. These plans are stored on a plan queue, and the plan dispatcher chooses the pending plan to follow. The plan executor pursues that plan by making suitable changes to the queue of pending actions so that the next action for the chosen plan is taken. Finally, to the problem solver and local planner we add a partial global planner that uses the local plans and plan information from other nodes to form partial global plans. It maintains a queue of partial global plans and has a dispatcher that chooses the partial global plan that the node should follow. The partial global planner modifies the local plan associated with the chosen partial global plan to improve coordination, and the local plan queue is updated to ensure that the appropriate local plan is dispatched next.

The local planning mechanisms use the relationships between actions to form and incrementally update local plans. Even in domains where actions are interdependent, however, it is often a complicated process to identify larger groups



The control components of the basic problem solver form and execute problem solving actions. The local planner combines related pending actions into plans, and executes the best plan on the queue by inserting the appropriate action at the top of the action queue. The partial global planner builds partial global plans (PGPs) from local plans and received plan information, and executes the best PGP by inserting the relevant local plan at the top of the plan queue. The different control activities can be pursued concurrently and asynchronously.

Figure 1: An Overview of the Levels of Control.

of related actions, develop promising sequences of actions, and form predictions about when those actions will be done and what kind of results they are likely to generate. Nodes therefore make only tentative local plans that may change over time.

The partial global planning mechanisms must adequately coordinate local plans despite the fact that these plans can change over time. To form partial global plans, a node uses its current view of other nodes' plans, but because nodes may be in the process of changing their plans, this view may be incomplete or obsolete. As a result, partial global planning is a complex process, balancing nodes' needs for predictability with their needs for responsiveness. The mechanisms work by grouping together nodes' plans that participate in larger, more global long-term goals and finding ways that nodes could revise their local plans to improve coordination. In addition, the partial global planning mechanisms build predictions about when and where nodes should communicate partial results to construct more global results. Hence, the partial global planning mechanisms reason about nodes' concurrent actions (to improve how they complement each other's local activities) and their interactions (to determine how they should combine the results of their activities).

Because several nodes may be forming partial global plans simultaneously, the partial global planning mechanisms must tolerate some inconsistency among nodes' views of cooperation and must allow nodes to exchange coordination information to resolve inconsistencies. How nodes go about doing this depends on the meta-level organization. The meta-level organization specifies what nodes a node should inform about its local and partial global plans, what nodes are responsible for forming partial global plans and for what other nodes, and how much credibility a node has in information it gets from another node. Although the partial global planning mechanisms have the flexibility to let nodes cooperate in whatever style is indicated by the meta-level organization, an important (and open) research issue is how the nodes can reorganize their coordination roles when circumstances change.

Problem solving, local planning, and partial global planning can be done asynchronously and in parallel: the local planner follows the top ranked local plan whether or not the partial global planner has altered the queue, and similarly the problem solver executes the top ranked action whether or not the local plan-

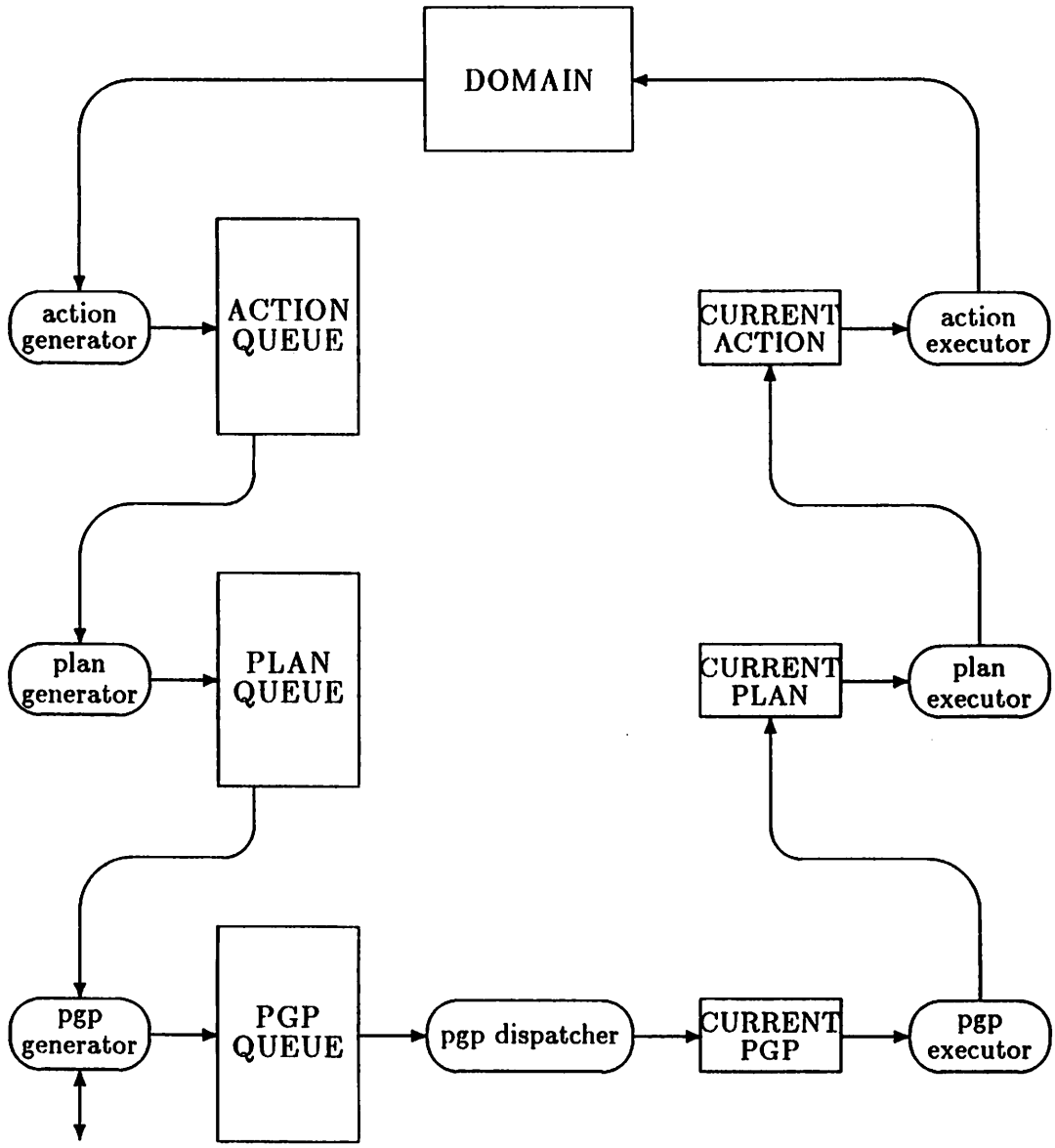
ner has influenced this choice. In the current implementation, however, problem solving, local planning, and partial global planning are done on a single processor, so a node's control activities step through these mechanisms, as shown in Figure 2. The dispatcher for choosing the next action is replaced with the local planner which finds the next action for the best local plan. In turn, the dispatcher for choosing the best local plan is replaced with the partial global planner which finds the local plan of the best partial global plan. The detailed description of the planning and coordination mechanisms in the next chapters use this view when describing the sequence of control activities performed by a node. However, in the final chapter, we briefly return to issues in performing problem solving, local planning, and partial global planning in parallel.

1.1.3 Empirical Contributions

To more fully understand the benefits, costs, and limitations of the partial global planning approach, the approach should be evaluated from both a conceptual and a practical perspective. Because we have implemented the approach in a complex domain, we are able to experimentally evaluate its performance in a variety of situations. The experiments measure not only the improvements to control due to the new mechanisms, but also the overhead costs that the mechanisms incur.

In experiments with local planning, we show that planning is advantageous even when a node is working alone. We present experimental results showing how our new local planning mechanisms resolve uncertainty about goals, monitor and repair plans, modify plans in response to changing circumstances, and allow a node to meet deadlines. The overhead of planning is generally offset by the saved effort in problem solving, since a node makes better control decisions and takes fewer incorrect actions.

While local planning makes each node a better problem solver, partial global planning makes nodes work as a better team. In the problem situations we experimentally explore, we show that partial global planning improves control decisions in the network, usually with acceptable overhead. Moreover, our experiments verify that nodes should coordinate differently in different situations, so that the versatility of partial global planning is extremely useful. For a given situation, we use different meta-level organizations to promote coordination in different ways, and use the experimental results to recognize the characteristics of a situation



In a serial view of control, the control components of the basic problem solver first form problem solving actions. The local planner then combines related pending actions into plans, which the partial global planner uses, along with received plan information, to build partial global plans (PGPs). The partial global planner executes the best PGP by making the relevant local plan the current plan, and the local planner executes this local plan by making the appropriate action the current action. This action is taken, causing changes to the domain information that may trigger the generation of new potential actions, and the cycle repeats.

Figure 2: A Serial Overview of the Levels of Control.

that influence decisions about how nodes should coordinate.

The experiments that examine local planning (Chapter 5) and partial global planning (Chapter 8) thus contribute to our understanding of control and coordination in general, and represent initial explorations that further research can build upon.

1.1.4 Research Methodology

A fable by Aesop:

> A crab said to her son, "Why do you walk so one-sided, my child? It is far more becoming to go straightforward." The young crab replied: "Quite true, dear mother; and if you will show me the straight way, I will promise to walk in it." The mother tried in vain, and submitted without remonstrance to the reproof of her child.
>
> Moral: *Example is more powerful than precept.*

Experimentation plays a major role in our research because an approach to coordination should be evaluated from a practical as well as a conceptual standpoint. In fact, experimentation and conceptualization are concurrent activities. Concepts guide the experiments, but experimental results (especially unexpected results) cause revision and extensions to concepts.

An advantage of coupling experimental and conceptual development is that it forces concepts to be well defined, otherwise they cannot be implemented. Moreover, experimental verification of a conceptual approach can point out holes in that approach or its limitations. Because our goal is to understand not only how nodes can cooperate, but also the costs and limitations of our approach, implementing and experimenting with the approach allows us to evaluate its practical aspects. Working with concrete, experimental examples also allows us to anchor our concepts in reality, so that we can better characterize and understand phenomena. There are disadvantages in developing a conceptual and experimental approach concurrently, however, and the most important of these is the danger of forming concepts that are only applicable in the experimental domain. To some extent, this is unavoidable: examples lead to seeing issues in certain ways, and the concurrent evolution of concepts and implementations make disassociating the two difficult.

The design, implementation, experimentation, and evaluation of our approach has been (and still is) an iterative process. The high-level concepts behind partial global planning—building plans for groups of nodes from plans of individual nodes—are by themselves not particularly enlightening, but making these ideas actually work in a non-trivial domain is an interesting and exciting problem. For this reason, the bulk of this dissertation is concerned with *how* things work. We mostly transmit our ideas through implementation details and examples, although at intervals we also highlight how our mechanisms are more generally applicable in other domains. The result, hopefully, is a balance between details and generalizations, so that the reader can develop an understanding of how we have made things work and how we believe similar things could be done elsewhere.

In conclusion, our research methodology emphasizes the simultaneous evolution of concepts and mechanisms: we are continually examining new cooperative situations and using what we learn to improve our conceptual and experimental approach to cooperation. Although we here describe the research at an important stage of completion, it would be wrong to assume that further evolution will not occur. In fact, the final chapter outlines several directions for future research that, although they will leave many aspects of the current system unchanged, could extend our approach in new and unexpected directions.

1.2 Research Issues

In this section, we discuss several important research issues in more detail: how nodes can be sufficiently aware of their local state to relate their own activities with the activities of others; what the different goals of cooperation are; what the different styles of cooperation are; and how nodes should balance predictability and responsiveness.

1.2.1 Local Node Sophistication

When a computing node is working alone, it can afford to be less sophisticated because it need not *explicitly* represent its current and expected future states and actions. However, for nodes to plan their interactions, they must have an explicit representation to communicate about. By explicitly representing their current and planned activities, nodes can recognize possible future interactions,

both desirable and undesirable. An interesting (but on reflection not surprising) common thread in much distributed computing research is that individual computing nodes need more self-awareness than when they work alone so that better techniques for coordination can be employed.

This research has similar needs for local node sophistication, and an important thrust of this work is to improve control in individual nodes. When working on its own, a node can explore possible solution paths in an erratic manner, possibly switching frequently among a number of paths depending on the promise of the next step of each. When it must coordinate with others, however, a node should attempt to be more predictable so that others can anticipate its actions and can coordinate interactions. A node should consider the long-term promise of the various paths and should plan purposeful sequences of actions. A major aspect of this research is to give a node the ability to form the long-term view it needs to recognize promising goals to work toward, but to form this view without incurring excessive overhead.

Once it has long-term goals, a node must plan actions to achieve them, and mechanisms for forming these plans are needed. In dynamic domains, moreover, the situation changes over time, and so a node must be able to change its goals and plans. The goals (solutions to work toward) are therefore uncertain and changing, and a node cannot simply plan out a sequence of actions based on the initial situation. Instead, it should interleave building, modifying, monitoring, and repairing plans with executing plans. A node should plan *incrementally* by sketching out a general sequence of activities but adding detailed actions over time since the results of earlier actions affect how (and whether) later actions should be taken.

Local node sophistication is thus crucial to coordination because, without the ability to plan local actions, the nodes are incapable of communicating about their future activities to plan coordinated interactions. The increased self-awareness achieved through locally planning also helps a node behave more purposefully, and can improve local problem solving. Because it is so important to coordination, and because it is difficult to achieve in independent problem solvers working in dynamic domains, much of the research described in the later chapters (especially Chapters 3–5) is concerned with increasing local node sophistication.

1.2.2 Goals of Cooperation

The form that cooperation takes depends on the situation, and cooperative actions in one situation may be undesirable in other situations. For example, students may cooperate to improve each other's exam grades by discussing the material *before* an exam, but discussion *during* the exam is likely to achieve the opposite effect. Before the exam, the students cooperate by sharing resources (ideas and information), while during the exam the students should cooperate by avoiding harmful interactions (any interactions at all might lead to accusations of cheating).

Choosing appropriate actions therefore requires an understanding of the goals of cooperation. The generic goals of cooperation include:

1. To increase the task completion rate through parallelism.

2. To increase the set or scope of achievable tasks by sharing resources (physical devices, information, expertise, etc.).

3. To increase the likelihood of completing tasks (reliability) by undertaking duplicate tasks, possibly using different methods to perform those tasks.

4. To decrease the interference between tasks by avoiding harmful interactions.

In particular applications, specific variations and combinations of these generic goals of cooperation determine how nodes interact. In a distributed problem solving application, for example, these generic goals are refined into a set of more specific goals of cooperation. Associated with their related generic goals of cooperation (numbers in parentheses refer to generic goals above), these goals include:

- To increase the solution creation rate by forming subsolutions in parallel (1).

- To minimize the time nodes must wait for results from each other by coordinating activity (1,4).

- To improve the overall problem solving by permitting nodes to exchange predictive information (2).

- To increase the probability that a solution will be found despite node failures by assigning important tasks to multiple nodes (3).
- To improve the use of computing resources by allowing nodes to exchange tasks (2).
- To improve the use of individual node expertise by allowing nodes to exchange tasks (2).
- To reduce the amount of unnecessary duplication of effort by letting nodes recognize and avoid useless redundant activities (4).
- To increase the confidence of a (sub)solution by having nodes verify each other's results through rederivation using their potentially different expertise and information (2,3).
- To increase the variety of solutions by allowing nodes to form local solutions without being overly influenced by other nodes (1,4).
- To reduce the communication resource usage by being more selective about what messages are exchanged (4).

Because all of these goals cannot be achieved simultaneously, nodes must cooperate differently depending on the particular situation. For example, if a solution must be found quickly, the nodes should not spend time verifying each other's results or developing a wide variety of solutions. Because of the diversity in the forms that cooperation can take in a distributed problem solving system, distributed problem solving is an appropriate context in which to study how different goals of cooperation can be met.

1.2.3 Styles of Cooperation

The purpose of the coordination mechanisms is to control problem solving so that cooperating nodes work together as a coherent team. However, the question arises about how to control coordination activity so that nodes develop more coordinated views appropriately. That is, before they can coordinate their problem solving, nodes must first decide how they will coordinate their activities to form this coordinated view. There are several different ways of controlling how nodes

coordinate [Cammarata *et al.*, 1983; Corkill, 1983; Davis and Smith, 1983; Steeb *et al.*, 1986]. For example, the nodes could send coordination information to a single node and expect that node to coordinate the network. Or the nodes could broadcast coordination information to each other and individually develop views about how they should all cooperate. Or pairs of nodes could exchange information to decide on task reassignments, forming shared plans (contracts) with each other. Or the nodes could avoid exchanging any coordination information and instead fall back on any domain-level organizational knowledge they have about each other's general problem solving capabilities and interests.

Each of these different styles of cooperation has advantages and disadvantages. Having a single coordinator node has advantages such as having only one node spend its resources on reasoning about coordination and having that node enforce consistent views of coordination, but has disadvantages such as this node being a potential bottleneck and the entire network being prone to collapse if this coordinating node fails. On the other hand, having each node coordinate itself, though more reliable, may incur large amounts of overhead since more information must be exchanged and since nodes may duplicate each other's reasoning (form the same views of coordination). Forming contracts between pairs of nodes can be very cost effective when some nodes must pass tasks to others, but when larger groups of nodes already have tasks distributed among them they may be unable to gain a suitably global view of how to interact if they only consider how pairs can coordinate. The advantage of not exchanging any information at all is that the coordination overhead (computation and communication) is eliminated, but at the potential cost of a lack of network coherence (duplication of effort, working at cross-purposes, poorly timed interactions) leading to poor network performance. Therefore, no single style of cooperation is appropriate for all distributed problem solving situations [Smith and Davis, 1981].

In fact, some distributed problem solving situations may call for several styles of cooperation simultaneously. For example, consider a vehicle monitoring problem where nodes track vehicles moving through an area monitored by acoustic sensors. A sample problem situation (Figure 3) has four nodes, each connected to a different sensor (node i to sensor i) that supplies it with signal information at discrete times. A node tries to combine signals into tracks, which we represent as d_i–d_j where d_i is data for the track's first sensed time and d_j is data for its

last. When all the data is present,[1] then from a local perspective: node 1 plans to develop two tracks (d'_1–d'_5 and d_4–d_{12}); 2 plans for one track (d_{10}–d_{15}), but because its sensor is faulty it will need much more time to filter out noisy data; 3 plans for one track (d_1–d_6); and 4 has no local plans. From a global perspective: node 1 should first work on track d_4–d_{12} since it has more global significance (joining the tracks of 2 and 3), and also should quickly generate and send a predictive result (like the short track d_8–d_9 which borders on node 2's view) to help 2 disambiguate its noisy data; 2 should expect this predictive information; 3 should take responsibility for the data that both it and 1 sense (d_4–d_6) since 1 has more data to process; and 4 should take on some tasks, either by getting data from other nodes, or by acting as network coordinator, or both. Pairs of nodes therefore need to transfer tasks (form contracts), and larger groups of nodes need to exchange partial results to converge on global solutions and need to plan interactions that will help each other perform their tasks better.

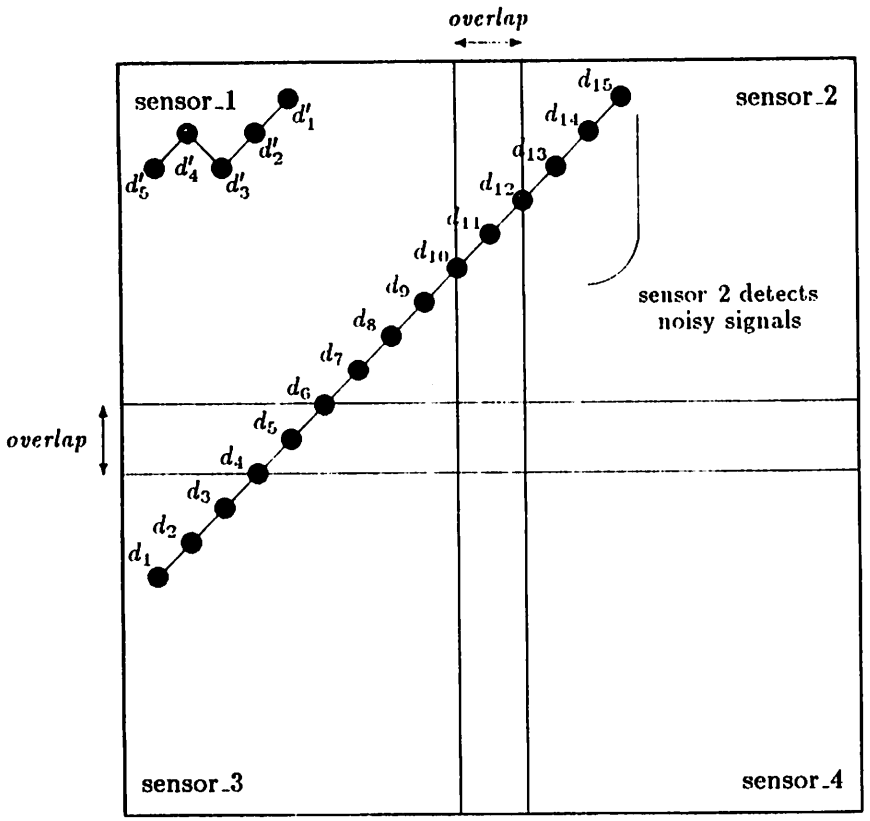
In complex distributed problem solving domains like vehicle monitoring, therefore, the problem solvers need the flexibility to cooperate in several different styles, and partial global planning allows nodes to cooperate in different styles within a single framework.

1.2.4 Predictability and Responsiveness

To behave in a dynamic domain, nodes should be responsive to changes. Conversely, nodes should also be predictable so that they coordinate with others as expected. The conflict between predictability and responsiveness is therefore a recurring problem in coordination.

Before it coordinates with others, a node can respond to changes in its situation in any way that its local control component considers best. So long as it need not coordinate with others, in fact, a node can and should be responsive. However, to cooperatively solve problems, nodes should exchange information about their plans and goals, and what they send determines how other nodes will perceive them. When a node changes its local plans and goals, it deviates from the view that other nodes have, which is the view that they use when mak-

[1] To simplify many examples, we assume that all of the data arrives before problem solving begins. In experiments in Chapters 5 and 8, we also explore situations where data arrives over time.



The four overlapping sensors detect signal data at particular locations for discrete sensed times (the dots with associated times). Sensor_2 is faulty and not only generates signal data at the correct frequencies for each location but also detects noisy signals at spurious frequencies for each location. Node i is connected to sensor i. Node 1 should send predictive information to node 2 to help resolve ambiguity, nodes 1–3 should avoid redundant work in their overlapping areas, and node 4 should take on tasks to avoid sitting idle and letting its resources go to waste.

Figure 3: A Situation Needing Several Styles of Cooperation.

ing coordination decisions. Delays in communication mean that other nodes will continue to use obsolete views of the node for some interval of time until they receive an update, and the coordination decisions are thus not as effective as they might be over this interval. Ineffective coordination can cause nodes to work at cross-purposes or on unimportant tasks, and the best overall solutions may be found later than if the node had not changed its local plans. On the other hand, when the changes to its plans can substantially impact network behavior (trigger working in completely different areas or in different ways), then the node should change its plans. The network can tolerate an interval of uncoordinated activity if the network performs significantly better after the changes have been made known to others.

Deciding when a change to its local plans is substantial enough to warrant a response is a difficult problem, but one which our approach takes initial steps toward solving. Our approach to coordination has nodes plan their interactions—not only *how* they will interact but also approximately *when* they will interact. When a node changes its local plans, therefore, it can compare how these changes will affect planned interactions: can the node still interact in the expected way and will the time that the node will be ready for the interaction be about the same? When nodes can no longer interact in the expected way, then it is important to respond to the situation. However, if the change indicates that they will interact at about the same time (where the tolerance to changes in time can be adjusted), then the node need not change its plans. Tolerance for minor variations in interactions increases predictability at the cost of possibly degrading performance (since nodes might have cooperated more effectively had they responded to these variations). However, being responsive can itself degrade performance, not only because of intervals where nodes have inconsistent views of each other but also because reformulating how they should coordinate incurs overhead and the costs of this overhead might outweigh its benefits. Our approach allows us to adjust the tolerance and observe the effects to better understand how predictability and responsiveness should be balanced.

1.3 Relationship to Previous Research

The relevant previous research in cooperation and coordination among independent nodes can be divided into three basic categories. The first of these is distributed artificial intelligence. Because we are interested in cooperation between artificially intelligent problem solvers, and because we view coordination decisions as knowledge based (knowledge about local plans, plans of other nodes, and goals of cooperation), previous research in distributed artificial intelligence is very relevant. The second category of related research is in distributed operating systems and distributed control in general. Although principally interested in domains with much less interdependence between tasks (where avoiding resource conflicts is the primary objective), distributed operating systems research is evolving toward distributed artificial intelligence research as it faces similar issues in coordinating nodes with tasks that are related in more complicated ways. Finally, the third category of related research is studies of cooperation in natural systems. This includes social, biological, game theoretic, and economic views of cooperation, each of which can enrich our understanding of cooperation among complex agents.

1.3.1 Distributed Artificial Intelligence

Distributed Artificial Intelligence (DAI) research, in our view, studies how intelligent agents can combine their resources so that the intelligence of the group is more than the sum of the individual agents' intelligence. This view assumes that the individuals do have some intelligence (as opposed to connectionist views where simple, unintelligent computing elements combine to form an intelligent whole [Feldman and Ballard, 1982]). For our purposes, we define intelligence as the ability to flexibly respond to situations: an intelligent agent (or network) does not simply react to its environment but instead uses knowledge (possibly heuristic) to make informed decisions about how to act. When they cooperate with others, the combined abilities of the agents transcend their individual capabilities so that the size and scope of the problems they can solve as a group increases dramatically.

Many crucial issues in DAI revolve around providing agents with the ability to control how they cooperate:

- Agents need to decompose high-level tasks, assign subtasks among themselves, and combine the results of these subtasks.
- Agents need to represent and reason about their own actions, the actions of other agents, and the interactions between agents.
- Agents need to recognize when they have conflicting or inconsistent goals and respond appropriately by, for example, negotiating, competing, or appealing to a higher authority.
- Agents need to coordinate their interactions so that they work as a coherent team when cooperating to achieve shared goals.

The emphasis of this section is to review past approaches to control in DAI systems so that we can contrast our approach with these. For more complete information about DAI in general, the reader is referred to the surveys by Rosenschein [Rosenschein, 1985] and Corkill and Lesser [Corkill and Lesser, 1987], and the DAI annotated bibliography by Jagannathan and Dodhiawala [Jagannathan and Dodhiawala, 1986].

Control of Closely Cooperating Experts

Early DAI research dealt with closely cooperating agents. Because these agents cannot be considered intelligent in themselves, these systems do not fit in the definition of DAI given above. Nevertheless, they do represent some initial explorations into issues of cooperation between independent computing agents.

In Hearsay-II, the computing elements are KSs that interact through a blackboard data structure [Erman *et al.*, 1980; Fennell and Lesser, 1977]. A KS monitors a portion of the blackboard, waiting for particular patterns of data. When such a pattern occurs, the KS takes the data and manipulates it, typically forming new combinations of data which it places on other portions of the blackboard. A KS cannot be construed as an intelligent agent by itself because it responds in a set way to expected patterns of data. However, by acting asynchronously and by communicating through the blackboard, a group of cooperating KSs can respond flexibly to data so that their overall behavior can be viewed as being intelligent. Control of cooperation is effected in several ways: the specificity of KSs (in terms of the data they respond to and produce) limit the ways they can interact; the

structure of the blackboard allows KSs to work concurrently in different areas of the blackboard; and a blackboard monitor resolves contentions between KSs attempting to modify the same part of the blackboard.

In the ACTOR system, simple, unintelligent computing agents very similar to KSs (but often of much smaller granularity) cooperate through a communication channel [Hewitt, 1977]. An ACTOR monitors the communication channel, waiting for messages fitting its desired pattern. The ACTOR responds to such a message by performing some computation on that message and possibly generating new messages during and after this computation. These new messages in turn trigger other ACTORs to act, and so the network of communicating ACTORs, which are individually simple, can flexibly interact to different situations (patterns of messages) to form complex, intelligent computations. Moreover, the ACTORs may maintain a local state that associate messages received in the past with the results of the computation for that message, so that it need not recompute results that it already computed. Control of cooperation in this system is based on: ACTORs responding to highly specialized patterns and generating specialized results; patterns of messages triggering concurrent activities; and a communication channel that has the speed and capacity for ACTORs to freely broadcast messages in a timely manner.

The two systems are extremely similar. Both work toward deriving intelligent overall behavior from groups of unintelligent agents, and depend on concurrency and close coupling to achieve this. Concurrent activity of a number of agents allows these systems to perform adequately despite a lack of explicit coordination. The incoherent behavior of many of the agents can be tolerated because the odds are that some of the agents are doing useful work at any given time. The close-coupling allows large amounts of relatively simple information to be exchanged, and this means that agents communicate often to maintain a more global view of their activity. In blackboard-based cooperation, this view is maintained on the blackboard, while in the ACTOR system, this view is maintained by the constant flow of messages in the system. In both systems, an important source of contention is for the information passing mechanisms, whether a shared blackboard or a shared communication channel.

The view of cooperation afforded by these mechanisms is based on agents simply responding to information and publicizing their results. These mechanisms

have limited application in loosely-coupled systems, since the costs and delays of communication make the broadcasting of all results impractical. To cooperate effectively, loosely-coupled agents need more local intelligence and more sophisticated control. If the loosely-coupled agents are as simple as KSs or ACTORs and need to frequently exchange simple information, then they would spend most of their time waiting for messages from each other. The agents should therefore have more local intelligence so that they perform more local computation and interact less frequently. The less frequent interactions should also be more directed: to make effective use of their communication channels, the agents should make better decisions about what information to communicate and where to communicate it. Research in closely-coupled systems therefore developed an initial view of control that, through its failure in loosely-coupled systems, helped direct later efforts to finding more effective control mechanisms.

Control Through Contracts

Consider a network of loosely-coupled agents with various resources. If one of these agents receives a large problem to solve, then it has two choices: it can apply its own resources and solve the problem as best it can by itself; or it can decompose the large problem into smaller subproblems and convince other agents to pursue these subproblems. To make the best use of network resources, the agents should cooperatively solve large problems by assigning subproblems to suitable agents and working concurrently on these subproblems.

The contract-net protocol developed by Smith develops a framework for cooperating in this manner [Smith, 1980]. Given a task to perform, a node first determines whether it could break the task into subtasks that could be performed concurrently. If it forms such subtasks, or it is locally unable to perform the initial task, then the node must coordinate with others to decide where to transfer tasks. It employs the contract-net protocol to announce the tasks that could be transferred and request that nodes that could perform any of those tasks submit bids. A node that receives a task announcement message replies with a bid for that task, indicating how well it believes it can perform the task. The node that announced the task collects these bids and awards the task to the "best" bidder. The contract-net protocol allows nodes to broadcast bid-requests to all others or to focus bid-requests to a likely subset of nodes. Nodes can also communi-

cate about their availability, so that focusing information (and decisions about whether it is worth advertising a task in the first place) can be based on dynamic views of the network.

The contract-net protocol promotes control based on negotiating over task assignments to form contractor-contractee relationships [Davis and Smith, 1983]. These relationships determine how nodes will act and interact, and allow them to coordinate their activities to work together effectively. Because nodes exchange information about tasks and availability, they make dynamic control decisions about how they will cooperatively pursue tasks. Thus, in applications where the principal mode of interactions between nodes fits into the contracting model, the contract-net protocol is an effective approach for controlling cooperation. Unfortunately, there are applications which do not fit cleanly into this model of cooperation: tasks may be inherently distributed among nodes, and coordination is not a matter of decomposing and assigning tasks but instead is a matter of recognizing when distributed tasks (or partial results) are part of some larger overall task (or result) and, when this is the case, how to interact to achieve the larger task (or result).

Control Through Organization

The organizational approach to coordination developed by Corkill and Lesser stems from a different view of cooperative problem solving [Corkill, 1983; Corkill and Lesser, 1983; Lesser and Corkill, 1983]. In domains where subproblems of some larger problem are inherently distributed, the cooperating nodes do not need to coordinate subproblem assignments, but they do have to coordinate how they combine subproblem solutions. When the nodes decompose problems and assign subproblems themselves, they have knowledge about how the problems were decomposed and so how the subproblem solutions should be recombined. In domains where subproblems are inherently distributed, nodes do not know what subproblems other nodes are working on and how those subproblems are related. To cooperate, these nodes exchange partial solutions in a functionally-accurate, cooperative manner [Lesser and Corkill, 1981] so that despite some incorrect control decisions they eventually converge on overall problem solutions. Without any knowledge about the possible activities of other nodes, nodes can do no better than to broadcast partial solutions in the hope that they will eventually

converge on overall solutions if they exchange enough information. However, in loosely-coupled networks, such a communication-intensive approach would be infeasible. The organizational approach to coordination was devised to deal with such situations.

An organization indicates the general interests, responsibilities, and capabilities of the nodes. Each node has a copy of this organization, and uses the organization to guide its local decisions. For example, given local tasks (to process local data), a node gives preference to tasks that coincide with its organizational roles. Once it has formed some partial solution, it can also use the organization to identify nodes that could possibly use (extend, improve) this partial solution based on their organizational roles. An appropriate organization therefore promotes cooperation by guiding nodes into working on complementary tasks and selectively exchanging partial results. However, because an organization represents common knowledge shared by the nodes, forming an organization can be a costly and time consuming process. Organizations are thus relatively static, and must be general enough to allow acceptable coordination in a range of situations that could be encountered.

The organizational approach to cooperation establishes control by providing nodes with static, shared organizational knowledge and with the ability to use this knowledge when making decisions about their local activities and about what they communicate and where. The approach is limited, however, because of its inability to dynamically respond to specific situations: without a communication protocol for exchanging coordination information, the nodes cannot recognize cases where they could cooperate more effectively. Later extensions to the organizational approach (and preliminary steps in the research described in this dissertation [Durfee *et al.*, 1985a]) allowed nodes to exchange information to refine the organization based on current circumstances, but even so the nodes could not cooperate with complete flexibility, such as transferring tasks among themselves.

Control Through Multi-Agent Planning

In a multi-agent planning approach to cooperation, the nodes work together to form a multi-agent plan that specifies the future actions and interactions for each of them. Typically, one of the nodes acts as the group planner, and each of the

other nodes sends this node all pertinent information. The planning node forms a multi-agent plan and distributes this plan among the nodes. Since this plan is based on a global view of the problem, all important interactions between nodes can be predicted. Such prediction can be crucial in domains such as air-traffic control where it is imperative that detrimental node interactions (vehicle collisions) are predicted and avoided [Cammarata *et al.*, 1983]. Unfortunately, achieving a global view of the problem might be extremely costly both in communication resources and in time, and the performance of the entire network depends on the planning node and would be compromised if that node fails. Multi-agent planning by a single planning node may thus be infeasible in many realistic situations.

When multi-agent planning is done in a distributed manner, there may be no single node with a global view of network activities. Detecting and dealing with interactions among agents is much more difficult in this case. The general approach is to provide each agent with a model of other agent's plans (for example, Corkill's MODEL nodes [Corkill, 1979], Georgeff's process models [Georgeff, 1984], Konolige's belief subsystems [Konolige, 1984], and Rosenschein and Genesereth's exchange of tentative multi-agent plans [Rosenschein and Genesereth, 1987]). Interactions between plans are usually reconciled by having nodes synchronize critical regions of the plans.

How nodes coordinate in multi-agent planning systems is explicitly specified in the multi-agent plan that identifies all actions and interactions of nodes. The strength of this approach is that the nodes form a plan for a given situation and can therefore cooperate effectively in that situation. However, in dynamic domains where situations change over time, the multi-agent planning approach would be less effective: whenever the situation changes, the nodes would have to stop their activities and recompute (in either a centralized or distributed fashion) a new multi-agent plan. Depending on globally shared plans for coordination can thus have its drawbacks. To work in dynamic domains, nodes need the ability to cooperate despite having inconsistent views of planned actions and interactions, and should be able to form and communicate about plans asynchronously as problem solving proceeds.

Finally, the multi-agent planning approach to cooperation has been combined with other approaches in a two stage coordination process for controlling fleets of vehicles [Steeb *et al.*, 1986]. Each node (vehicle) has a copy of a pre-established

type of organization, where the organization has a single coordinating node that plans for the entire group. However, the assignment of organizational roles is not static: although they know that they will have a leader, the nodes must communicate to determine which node will assume that role. Thus, in the first phase of coordination, nodes exchange information about their current status to converge on a choice for leader. Once accomplished, then the second phase of coordination involves the nodes sending information to the leader and the leader forming a multi-agent plan, possibly by assigning sub-planning tasks to the other nodes. Therefore, although coordination is predominantly based on multi-agent planning, other techniques are fit in around this approach to increase its flexibility, resulting in a hybrid system for controlling cooperation.

Formalisms for Control of Cooperation

Several research efforts have been directed toward developing a more formal foundation for views of cooperation. In general, these have been grounded in predicate logic, focusing on the beliefs, goals, intentions, and actions of nodes. For example, Georgeff has proposed formal representations for reasoning about interacting plans and for representing how agents view each other to develop multi-agent plans [Georgeff, 1983; Georgeff, 1984]. More recently, Georgeff and Lansky have extended these ideas [Lansky, 1985]. Another formal approach for agents to represent each other's beliefs has been developed by Konolige [Konolige, 1984]. Halpern and Moses have explored questions of common knowledge that are very relevant to DAI [Halpern and Moses, 1984]. For example, they have shown that agents may never converge on common views in certain distributed environments.

Rosenschein and Genesereth have similarly been developing formal models of cooperative (and not so cooperative) activity [Rosenschein and Genesereth, 1985; Rosenschein, 1985]. Much of their work is based on game theoretic views, where the payoff an agent receives for an action depends on the actions concurrently being taken by other agents. Assuming that each agent knows about how every possible combination of actions will affect the payoffs of each of the agents, Rosenschein and Genesereth study issues in what agents need to know about how other agents make their decisions (are they rational?) in order to coordinate their decisions. They have also examined issues in converging on common plans in distributed environments, concerned not so much with mechanisms for

forming these plans as with identifying situations where convergence is possible [Rosenschein and Genesereth, 1987].

Formal models of cooperation allow researchers to mathematically prove theories about what cooperating agents can and cannot do, and about how assumptions about their domains and characteristics affect their capabilities. As a result, formal models are useful for delimiting problems and identifying research issues. On the other hand, bridging the gap between theories of cooperation and implemented mechanisms for cooperation is not a simple matter, and as a result, formalisms currently provide little help for developing practical systems.

1.3.2 Distributed Operating Systems

Distributed control is also of crucial importance to distributed operating systems research. Among the issues that are of concern in distributed computing systems are control of resources to avoid contention and deadlock, control of scheduling decisions to maximize concurrency and meet performance needs, and control of memory to insure integrity of distributed data. Therefore, distributed operating systems research is concerned with many of the same issues as distributed AI, but usually from a different perspective because tasks are typically considered to be independent. Recently, research has been conducted on domains where tasks are more interdependent (related by precedence) and the link between distributed operating systems and DAI has become more pronounced [Zhao, 1986; Zhao *et al.*, 1987].

This section discusses important general issues and approaches to control in distributed operating systems without going into details about specific systems. For a more information on distributed computing systems, the reader is referred to the survey by Stankovic [Stankovic, 1984] and to almost any issue of IEEE Transactions on Computers.

Control of Network Resources

Tasks in a computer system require resources such as memory, files, devices, and CPU time. The operating system must manage and allocate resources so that tasks that need those resources to complete have access to them. Avoiding contention for resources is important because tasks that contend for resources might

prevent each other from proceeding and lead to deadlock. Detecting contention and avoiding deadlock becomes particularly difficult in distributed computing systems since information about tasks and resource needs is distributed among the computing elements.

Several approaches have been taken for controlling access to resources in a network. In more centralized approaches, there is a single network manager for a type of resource that coordinates all accesses to that resource. The problem with such an approach is that the manager can be a bottleneck (since all accesses must pass through it) and dependence on a single manager makes the system unreliable. This second drawback can be reduced by having backup managers for the type of resource—but then not only is there a bottleneck for clearing access to a resource with the current manager but there is also the overhead of having that manager update the backup managers.

More decentralized approaches allow access to resources to be granted based on local views. This reduces the time costs and bottleneck problems, but might not prevent contention for a resource (because local views may be inconsistent). In this case, the mechanisms must also be capable of detecting contentions that may lead to deadlock and recovering from these problems, usually by aborting and restarting some of the tasks. Coordinating access to network resources based on local views therefore is useful in domains where recovery is relatively inexpensive and infrequent compared to the cost of maintaining global views of resource access.

Control Through Distributed Scheduling

A related area of research is in distributed scheduling techniques. Tasks arrive at nodes over time, and associated with these tasks may be their resource needs, expected computation times, priorities, and deadlines. A node will attempt to schedule its tasks to maximize some performance criteria, such as completing as many high-priority tasks to meet deadlines as possible. In the network of nodes, however, it may very well be the case that some nodes have more high-priority tasks to complete over a given interval of time than others, and transferring tasks among the nodes may allow the overall network to better meet the performance criteria. Distributed scheduling is thus an important form of distributed control in such systems.

For reasons of reliability and to avoid bottlenecks, a centralized network task scheduler is not a practical means for scheduling tasks in the network. Instead, schedulers at each node should control local activity but should also coordinate to find possible task transfers that improve network performance. The distributed schedulers therefore attempt to move tasks around by contracting them out: a local scheduler that determines it cannot meet the needs of a local task might contract that task out to another, less overburdened node. It is at this point that distributed operating system and DAI techniques overlap—contracting out tasks to improve network performance is a common feature to both, and DAI techniques have been borrowed and extended for use in distributed scheduling systems [Ramamritham and Stankovic, 1984; Stankovic *et al.*, 1985].

Control of Data Consistency

Controlling data consistency is much like controlling access to resources. A piece of data could reside at only a single node which controls how it is manipulated, but then this node could become a bottleneck (if that data is particularly popular) and reliability would be reduced (if that node fails then the data is lost). On the other hand, if the data is replicated at several nodes, then they must coordinate changes to that data, otherwise the data at the nodes might be manipulated differently and the nodes would have inconsistent information. Maintaining data integrity in a distributed system therefore relies on having nodes manipulate local copies of data and propagating these changes to each other.

Control of data consistency thus can take two principal forms. Either nodes could coordinate changes to data so that they all modify their copies of data at the same time (only when all commit to changing their data do they actually do so), or they could make local changes and then detect whether in fact inconsistencies have arisen. In the later case, nodes need the ability to negotiate about which of the different views should be adopted, and then activities based on inconsistent views must be retracted. Thus, coordination could incur the overhead for continually insuring against inconsistencies, or could incur the less frequent (but possibly more substantial) overhead of correcting the effects of any inconsistencies that may occur. The choice over which approach is better is domain dependent.

1.3.3 Natural Metaphors for Cooperation

The best examples of cooperation occur in natural systems, including social, biological, and interpersonal cooperation [Chandrasekaran, 1981]. In this section we briefly review some of the approaches to controlling cooperation exhibited by natural systems.

Social Metaphors for Cooperation

Cooperation is an integral part of social interaction. A group of agents (people, companies) can often achieve more by working together than they can by working alone. In an economy, for example, agents can interact by exchanging goods and services: by finding a suitable exchange, they each benefit because the utility each assigns to what it trades away is less than the utility it assigns to what it gets in return. A market represents a forum for making coordination decisions, a place where agents can communicate to make deals. Decisions about cooperation are (usually) made locally, not imposed from some other authority. Control in an economy thus emerges from local decisionmaking and is usually communication intensive. Only after a series of messages and input from the more global market database do the agents arrive at a decision. Nonetheless, economic models provide one view of how independent agents can choose to cooperate, and these models have been applied to computer systems [Kurose *et al.*, 1985].

Another important model for social interaction is game theory [Axelrod, 1984; Shubik, 1982]. Given information about possible actions and payoffs for those actions, agents can reason about how their local choices can affect and can be affected by the choices of others. They can recognize situations where cooperation (or competition) will give them the highest payoffs. Game theory uses formal representations, and various properties of interactions can be developed based on certain assumptions. In fact, these game theoretic approaches have been used in DAI formalisms to understand various types of cooperation [Rosenschein and Genesereth, 1985]. However, game theory generally provides descriptive information (why some control decisions are better than others) rather than prescriptive information (how to make good control decisions) except in very trivial domains.

Organizational theory also provides views of cooperation [Fox, 1983; Malone and Smith, 1984; March and Simon, 1958]. Organizations assume that the par-

ticipants are part of some overall team, and the purpose of an organization is to define the types of actions each should take and the patterns of interactions between them. The organization is thus established to serve a particular purpose, to coordinate agents working together in some basic way, and gives structure to the decisions that agents make that leads them to work together more effectively. Control is thus imposed on the agents through knowledge of their roles and the roles of others that constrain how they can act and interact. The work of Malone addresses issues common to human and computer organizations and helps bridge the gap between these types of organizations [Malone and Smith, 1984].

Biological Metaphors for Cooperation

While the social metaphors are concerned with cooperation between intelligent agents, biological metaphors address cooperation between much simpler agents. Therefore, it differs from the thrust of this research and we only briefly address it here by mentioning some representative work. First, the work by Dawkins views genes as self-interested agents that, through mutation and combination, can be thrown together into cooperative groups [Dawkins, 1976]. Cooperation in this domain is thus accidental, not intentional, and control of cooperation is not an issue. As a second example, neurons and neuron-like elements interact through their connections, and they can change the form of their interaction by modifying these connections [Barto, 1985]. Rewards are propagated among the agents to reinforce patterns of activity that led to favorable responses (based on environmental feedback). Thus, cooperation is controlled by having agents make local adjustments to their connections which, over many iterations, eventually generates a connection structure (an organization) that leads to good performance by the overall network. Again, cooperation is not intentional (the agents are too simple to recognize how their decisions affect other agents).

Natural Language and Cooperation

Research in natural language processing, and particularly in dialogues, addresses many issues concerning cooperation and building common views. To communicate effectively, agents must use models of each other to plan their speech acts [Cohen, 1978; Cohen and Levesque, 1986]. The work by Grosz, for example, is

concerned with discourse between potentially cooperating agents, and considers how a dialogue between agents exchanges sufficient information for them to develop a shared plan of action [Grosz and Sidner, 1985]. Coordination is achieved through discourse (to share information) and by assumptions that agents share about each other's goals and planning abilities that allow them to infer each other's (and hence their shared) plans. The common (or assumed common) knowledge structures their interactions so that they can coordinate without exchanging extensive amounts of information.

Stories also deal with how different characters interact, and these interactions are colored by a desire to cooperate, compete, conflict, etc. To understand a story, therefore, requires the ability to infer and represent the beliefs and goals of the characters. The approach by Bruce and Newman, for example, uses models of characters when attempting to interpret their actions [Bruce and Newman, 1978]. Although the focus is more on how to recognize relationships (including cooperative relationships) than on how to develop them, many of the issues in representing and reasoning about goals and beliefs are shared by other areas of research in cooperation.

1.3.4 Where Partial Global Planning Fits In

Our approach synthesizes and extends many of the previous approaches to DAI by building a unified framework for dynamically coordinating nodes. Rather than using contracting mechanisms for some situations and multi-agent planning mechanisms for others, our approach uses partial global plans as a versatile coordination mechanism that permits dynamic control of coordination in each of these various styles. This dynamic control is coupled with the more static organizational techniques so that nodes have a basic structure for interactions which they can modify and refine dynamically.[2]

The emphasis of this approach is on providing sophisticated local control

[2]Each technique for coordination must have some underlying organization to provide a framework for the more dynamic mechanisms. For example, in the contracting style the underlying organization is the preferences that each node has for bidding on tasks. If nodes have incompatible preferences, so that nodes capable of performing some tasks do not bid on those tasks, then some tasks may remain uncompleted and the network cannot converge on overall solutions. The network should reorganize, causing nodes to change how they bid on tasks so that network performance improves. Similarly, the multi-agent planning approach assumes an underlying organization about the planning roles of nodes (or of how they will decide on the planning roles).

that leads to making intelligent coordination decisions. An important aspect in controlling cooperation in social systems is the ability of the agents to make assumptions about each other—that they have so much in common that they can infer most of each other's goals and beliefs without exchanging any information at all (often simply by hypothesizing what their goals and beliefs would be if they were in another's situation). Providing artificial agents with better local control mechanisms and thereby increasing their self-awareness can lead to better cooperative reasoning by those nodes.

Once a node has intelligent local control, it can then go on to recognize and intelligently reason about coordination with others. In particular, it should *plan* how it will coordinate with others. Distributed operating system research provides specific techniques for coordinating local decisions, but these techniques must be improved on significantly before distributed computing systems can flexibly be applied to a wider range of domains. As research in distributed computing systems addresses these issues, it will become increasingly important to integrate planning into control and to improve the local capabilities of computing elements so that they can plan (albeit uncertainly) their future actions and interactions and can recover when those plans change. In short, distributed computing and DAI research will evolve toward each other.

Though it has ramifications to distributed computing systems research in general and might even shed light on aspects of natural forms of cooperation, our research is predominantly interested in extending DAI research. Moreover, from the previous discussion it should be apparent that our approach to cooperation borrows attributes of several DAI approaches, attempting to integrate and improve upon them. We need to address issues in contracting out tasks, in multi-agent planning, and in organizing agents not only to guide decisions about problem solving but also to guide decisions about how they should coordinate their problem solving. Many of these ideas are not new, and in particular Davis outlined many of the same issues [Davis, 1981]. Our research differs, however, in that it goes beyond speculation about how agents might coordinate their plans—we implement an approach that lays out exactly how coordination occurs, and we experimentally evaluate whether, in fact, the approach is practical.

1.4 Reading this Dissertation

This dissertation can be divided into four major parts. The first part is the introductory material and includes this chapter and Chapter 2 which introduces the domain and testbed in which we have experimented with our ideas. The second part covers local planning and is comprised of Chapters 3, 4, and 5, where Chapter 3 describes how local goals are recognized, Chapter 4 discusses how local plans are constructed and executed, and Chapter 5 provides experimental results and evaluation. The third part, Chapters 6, 7, and 8, is concerned with partial global planning: Chapter 6 discusses how more global goals are identified, Chapter 7 describes how partial global plans are formed and pursued, and Chapter 8 presents experimental results and evaluates the mechanisms. Finally, the fourth part, Chapter 9, contains concluding remarks and outlines avenues for future research.

A chapter by chapter breakdown is:

Chapter 1 has described the problem being addressed, introduced the partial global planning approach to dynamic coordination, presented some major research issues that we address, and outlined how our new approach relates to previous research.

Chapter 2 introduces the experimental concerns by outlining the problem domain and by describing the important aspects of the experimental testbed that acts as the foundation for our implemented mechanisms.

Chapter 3 presents our techniques for recognizing long-term, local goals by exploiting relationships in the data to process (tasks) to build up a hierarchical representation that is uniquely suited for making control decisions.

Chapter 4 describes how a node can incrementally plan actions to pursue the local goals and to resolve uncertainty about which goals to pursue, addressing issues in generating, monitoring, updating, repairing, and modifying these plans in a dynamic environment.

Chapter 5 summarizes a number of experiments that indicate the benefits, costs, and limitations of the local planning mechanisms, and uses these

results to evaluate the effectiveness of introducing planning into the control activities of a node.

Chapter 6 discusses how nodes can summarize and communicate about their local plans to model each other, how they can use these models to identify more global goals that they could cooperatively work toward, and how they are organized to decide where this information will be gathered.

Chapter 7 details the partial global planning mechanisms, telling how a node can interleave information about several nodes' plans to develop of view of their concurrent activities, how it can build expectations about future interactions and communication, how it can modify partial global plans to involve new nodes by sending them tasks, and how it can exchange partial global plans with others to converge on consistent views of coordination.

Chapter 8 experimentally tests these new mechanisms, indicating how they promote different styles of cooperation, how they balance predictability and responsiveness, how they achieve various goals of cooperation, and how their additional overhead is (usually) acceptable; these results are then used to evaluate the mechanisms and to suggest some improvements.

Chapter 9 summarizes what has been accomplished and its contributions, and outlines important directions for future research to improve and extend the mechanisms and the approach.

There are also three appendices: the first summarizes the experimental facilities used; the second presents an annotated trace to illustrate the experimental output, and the third provides a glossary of important terms.

The chapters that get into the meat of the mechanisms, Chapters 3, 4, 6, and 7, are all structured to first provide background information, then give an overview of the mechanisms developed in the chapter, then present those mechanisms in detail along with detailed examples, and then finally generalize the mechanisms to point out how they could be used in other domains. These chapters thus cycle through overview to detail to generalizations and back again. The purpose of these cycles is to put details and generalizations for important subgroups of mechanisms together to give the reader a more complete view of them.

A reader interested in the thrust of the research need read no further. Readers interested in the basics of the approach can find what they are looking for by reading Chapter 1, Chapter 2, the sections overviewing the mechanisms in Chapters 3, 4, 6, and 7, and finally Chapter 9. A reader interested in the implementation details can read the information on the testbed in Chapter 2 followed by the detailed descriptions of Chapters 3, 4, 6, and 7, while someone interested in how well the mechanisms worked could read Chapters 5 and 8. Those interested solely in local control techniques should read Chapters 2, 3, 4 and 5, while those interested in distributed problem solving should concentrate more on Chapters 2, 6, 7, and 8. However, because the mechanisms and concepts of later chapters build on those of earlier chapters, and because the complete set of mechanisms and concepts together blend into an overall approach to control and coordination, we believe that the reader that perseveres through the entire text will be richer, and not just wearier, for the effort.

This is as strange a maze as e'er men trod,
And there is in this business more than nature
Was ever conduct of. Some oracle
Must rectify our knowledge.
—Shakespeare *(The Tempest)*

Chapter 2

Distributed Problem Solving and the DVMT

A *distributed problem solving network* is composed of semi-autonomous problem solving nodes that can communicate with each other. Nodes work together to solve a single problem by individually solving interacting subproblems and integrating their subproblem solutions into an overall solution. Because each node may have a limited local view of the overall problem, nodes must share subproblem solutions; cooperation thus requires intelligent local control decisions so that each node performs tasks which generate useful subproblem solutions. The use of a global "controller" to make these decisions for the nodes is not an option because it would be a severe communication and computational bottleneck and would make the network susceptible to complete collapse if it fails. Because nodes must make these decisions based only on their local information, well-coordinated or *coherent* cooperation is difficult to achieve [Davis and Smith, 1983; Lesser and Corkill, 1981].

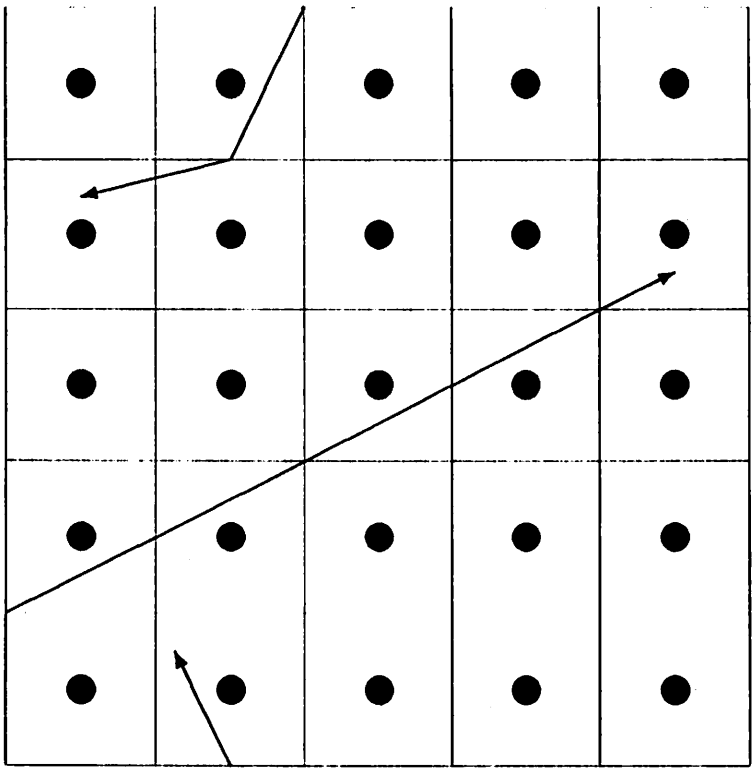
The *distributed vehicle monitoring testbed* (DVMT) simulates a distributed

problem solving network so that approaches for distributed problem solving can be developed and evaluated. In this chapter, we introduce the DVMT: the problem domain, the problem solving knowledge and architecture of a node, the local problem solving actions, the network interactions, and some examples of distributed vehicle monitoring problems. In the final section, we motivate the need for local and partial global planning in the DVMT.

2.1 The Experimental Domain

The experimental domain is vehicle monitoring. Acoustic sensors are dispersed in a two-dimensional area, where each sensor has a limited range. By distributing the sensors throughout the area, each part of the area is sensed by at least one sensor. For example, in Figure 4, the sensors are distributed throughout the overall area, and each sensor is responsible for a particular subregion. As vehicles pass through the overall area, they move through the subregions of various sensors, so each sensor only detects part of the overall vehicle path. By combining the data from each sensor, the vehicle can be tracked through the entire region.

Two basic approaches can be taken to correlating the sensor data. In a centralized approach, all of the sensors are connected to a single processor that is responsible for interpreting all of the data. In a distributed approach, several processors are distributed among the sensors and subsets of the sensors are connected to each processor. The distributed approach has a number of advantages: it exploits parallelism since the processors can work on different data concurrently; it reduces communication since the large amounts of raw data are sent a shorter distance (to a nearby processor) and only abstract views are passed longer distances; and it increases reliability since much of the overall area can be monitored despite the loss of a processor. The major disadvantage of the distributed approach is the difficulty in coordinating the problem solving: since the processor's have limited views of the overall problem, they lack the context for deciding how and where their individual pieces of a solution should be combined. Because of the difficulties in coordinating the processors, the distributed vehicle monitoring task is a fertile domain for studying control of cooperation.



A simple view of a vehicle monitoring task. Each dot represents an acoustic sensor and the small square surrounding each is its range. Together, the sensors cover the overall area and vehicles moving through the area (indicated by arrows) pass within the ranges of different sensors at different times.

Figure 4: A Vehicle Monitoring Task.

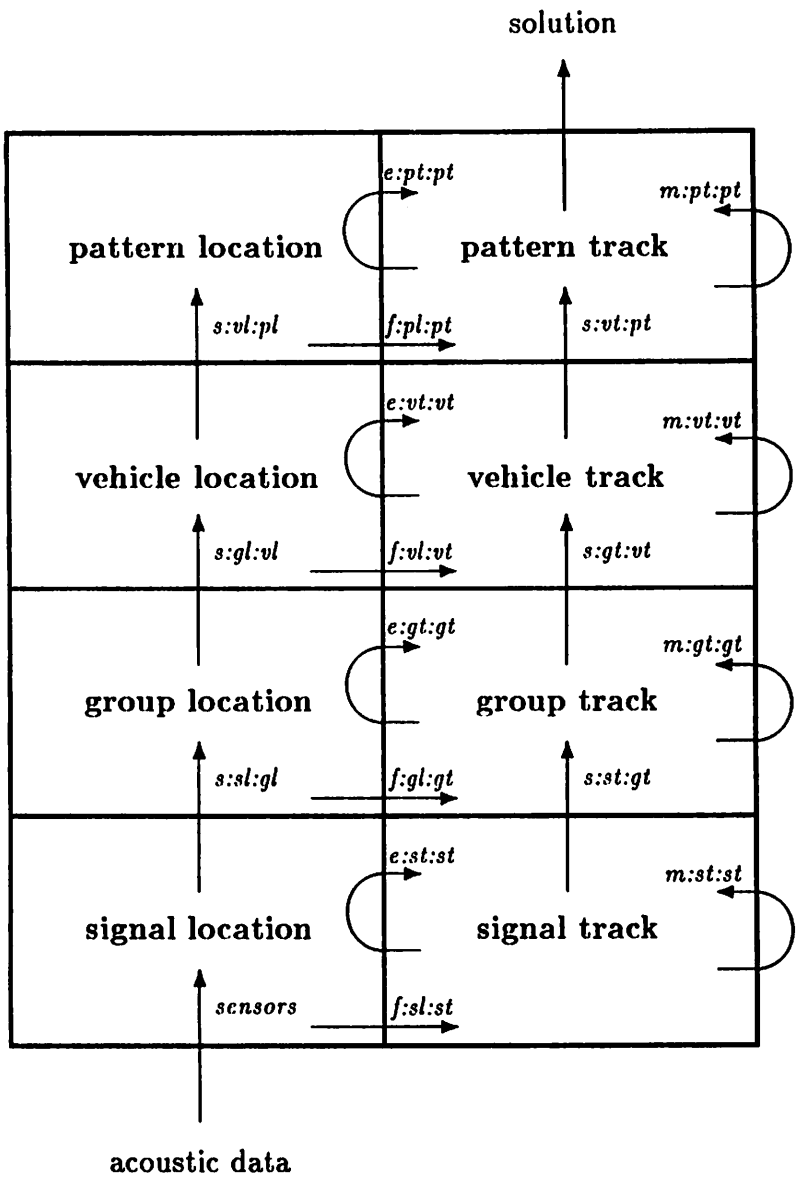
2.2 The Problem Solving Knowledge

In the DVMT, a vehicle monitoring node applies simplified signal processing knowledge to the acoustically sensed data in an attempt to identify, locate, and track patterns of vehicles moving through the two-dimensional space [Lesser and Corkill, 1983]. The vehicles' characteristic acoustic signals are detected at discrete time intervals, and these signals indicate the types of vehicles passing through the area and their approximate locations at each sensed time. An acoustic sensor's range and accuracy are limited, and the raw data it generates can be errorful, causing non-existent (ghost) vehicles to be "identified" and causing actual vehicles to be located incorrectly, misidentified, or missed completely. A node applies signal processing knowledge to correlate the data, attempting to recognize and eliminate incorrect noisy sensor data as it integrates the correct data into solution tracks.

Each node has a blackboard-based architecture, with knowledge sources and levels of abstraction appropriate for vehicle monitoring. A *knowledge source* (KS) performs the basic tasks of extending and refining *hypotheses*, where a hypothesis represents a partial interpretation of some signal data. A hypothesis is characterized by one or more *time-locations* (where the vehicle was at discrete sensed times), by an *event-class* (classifying the frequency or vehicle type), by a *belief* (the confidence in the accuracy of the hypothesis), and by a *blackboard-level* (corresponding to the amount of processing performed on the data).

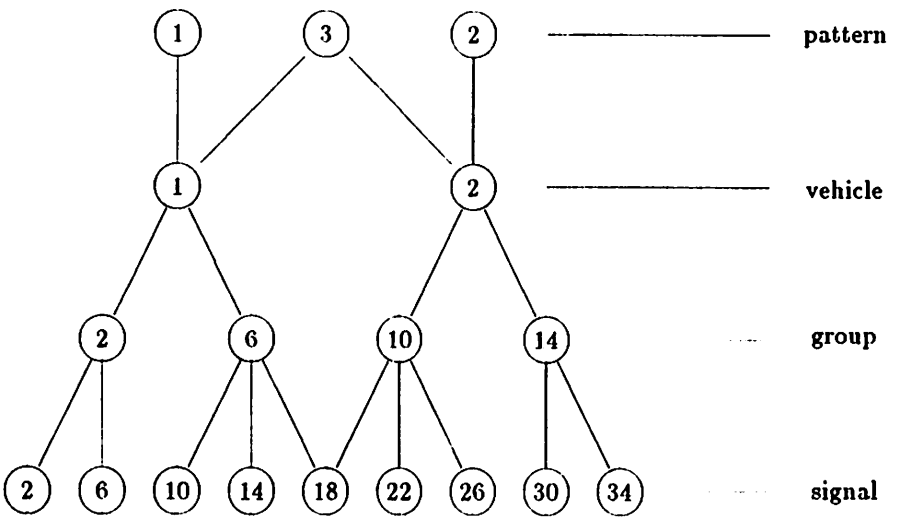
The blackboard has four principal blackboard-levels: **signal** (for low-level analyses of the sensory data), **group** (for collections of harmonically related signals), **vehicle** (for collections of groups that correspond to given vehicle types), and **pattern** (for collections of spatially related vehicle types such as vehicles moving in a formation). Each of these blackboard-levels is split into a blackboard-level for location hypotheses (which have one time-location) and a blackboard-level for track hypotheses (which have a sequence of time-locations). In total, nodes have eight blackboard-levels with appropriate KSs for combining hypotheses on one level to generate more encompassing hypotheses on the same or on a higher level (Figure 5).

The KSs can be divided into three basic types: synthesis, integration (or extension), and communication. A synthesis KS uses knowledge about how different



KSs combine hypotheses to form more encompassing hypotheses on the same or higher levels. KS names have the form *k:ib:ob*, where *k* is the type of KS, *ib* is the (highest) blackboard-level of the input hypotheses, and *ob* is the blackboard-level of the output hypotheses. Synthesis (*s:*) KSs generate higher level hypotheses out of compatible lower level hypotheses. Formation (*f:*) KSs form a track hypothesis from two combinable location hypotheses. Extension (*e:*) KSs combine a track hypothesis with a compatible location hypothesis to extend the track. Merge (*m:*) KSs combine two shorter track hypotheses that are compatible into a single longer track. The sensors KS creates signal location hypotheses out of sensed data. Communication KSs are not shown.

Figure 5: Blackboard-levels and Knowledge Sources.

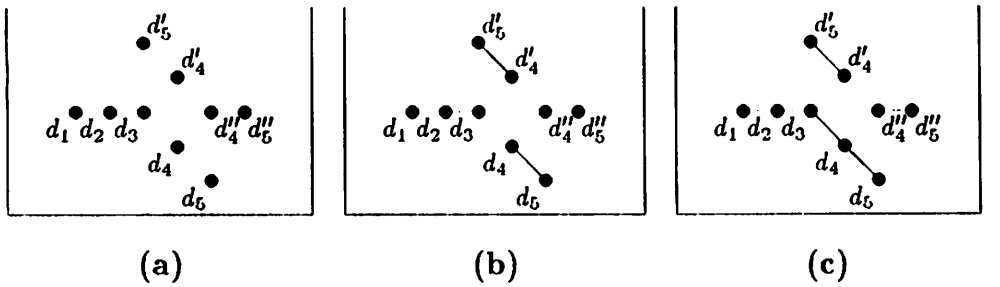


The signal grammar is represented graphically. For example, event-class 1 at the vehicle blackboard-level is composed of event-classes 2 and 6 at the group blackboard-level.

Figure 6: An Example Signal Grammar

distributions of frequencies are indicative of certain types of vehicles. It finds combinations of compatible data and filters out noisy data to identify likely vehicles based on a signal grammar. An example grammar is shown in Figure 6. To form a hypothesis at the vehicle blackboard-level with event-class 1, the node should first form hypotheses at the group blackboard-level with event-classes 2 and 6. In turn, to form these hypotheses it must use hypotheses with relevant event-classes at the signal blackboard-level (signal event-classes 2 and 6 lead to group event-class 2, for example). By looking for suitable combinations of hypotheses based on event-classes, the synthesis KSs can filter out data at spurious frequencies and can increase the confidence in hypotheses with a large degree of support (a large proportion of the supporting hypotheses at corroborating event-classes are present).

An integration KS (formation, extension, or merge) takes several hypotheses that each indicate where a particular type of vehicle may have been at some sequence of sensed times, and uses vehicle movement (velocity and acceleration) constraints to combine the hypotheses into a single hypothesis with a longer track. For example, after applying synthesis KSs to data at sensed times 1–



In (a), the individual data locations are shown (where d_i is data for sensed time i. In (b), some locations at adjacent sensed times have been combined into tracks, and the node then attempts to combine d_1–d_3 with each of the other tracks. Because of velocity and acceleration contraints, only one combination is possible, shown in (c).

Figure 7: An Example of Integrating Partial Tracks.

5 (Figure 7a), the node then uses integration KSs to join points together into longer tracks. After it has formed some of the partial tracks, the node might have the situation shown in Figure 7b. It attempts to combine the track d_1–d_3 with the possible tracks for sensed times 4–5. It cannot join d_1–d_3 with d''_4–d'''_5 because of velocity constraints: according to knowledge about how fast vehicles can move, a vehicle could not move from d_3 to d''_4 in a single time unit. Similarly, it cannot join d_1–d_3 with d'_4–d'_5 because of acceleration constraints: although a vehicle could get from d_3 to d'_4 in one time unit, it could not then turn fast enough to get to d'_5. Thus, the only extension to d_1–d_3 possible is to d_4–d_5 (Figure 7c).

Communication KSs (which are not shown in Figure 5) exist for each blackboard-level and allow nodes to exchange information from their blackboards. Transmission KSs access a hypothesis on the blackboard and build a message out of its attributes. This message is then sent to some other node. Reception KSs construct new hypotheses from received messages and insert these hypotheses on the blackboard. How the communication KSs decide what information to exchange with other nodes depends on the organizational structure, and we will describe this further later in this chapter.

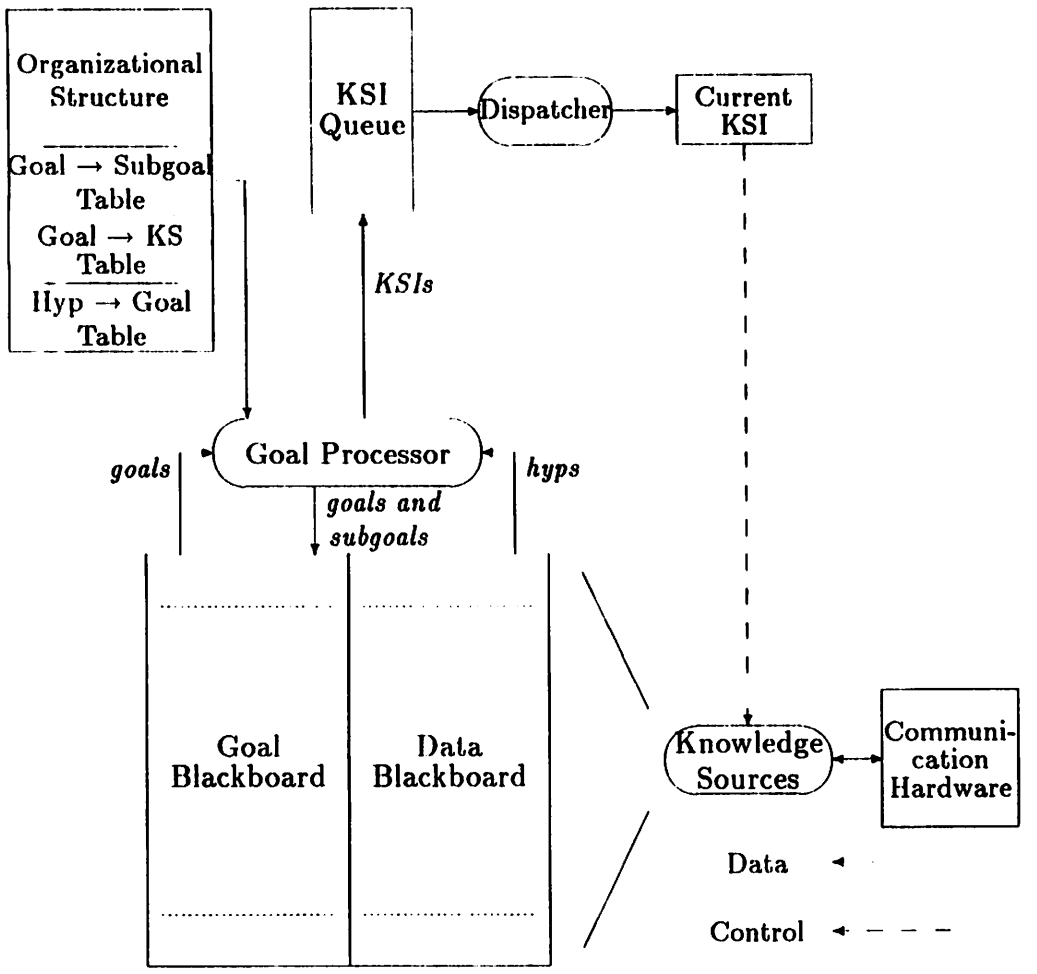
2.3 Control of Problem Solving

At any given time, a node must make control decisions about what KS to apply to which hypotheses. Since the node's objective is to form credible hypotheses at the top (pattern) blackboard-level, the control mechanisms should invoke KSs that extend and refine the best (highest belief) hypotheses.

A *knowledge source instantiation* (KSI) represents the potential application of a particular KS to specific hypotheses. Each node maintains a queue of pending KSIs and, at any given time, must rank the KSIs to decide which one to invoke next. When the next action must be taken, the dispatcher retrieves the most highly-rated KSI and invokes it. We use an extended Hearsay-II architecture (Figure 8) that lets nodes reason more fully about the intentions or *goals* of the KSIs [Corkill *et al.*, 1982]. When a hypothesis is formed, the node builds goals to explicitly represent how the hypothesis can be extended and abstracted. The goals are stored on a separate goal blackboard and are given importance ratings. Goal ratings, KSI ratings, and hypothesis beliefs have values in the range of 0 to 10000 (the DVMT belief/rating range), where 0 represents no confidence and 10000 represents complete confidence.[1]

The *goal processor* performs the bulk of the control reasoning. Given new goals to satisfy, the goal processor steps through possible KSs and applies their preconditions to find any that could potentially achieve those goals. If a KS precondition indicates that it might generate the desired hypotheses to satisfy goals, the goal processor forms a KSI. A KSI is rated, where this rating is based both on the estimated beliefs of the hypotheses it may produce (estimated by the KS precondition) and on the ratings of the goals it is expected to satisfy. By recognizing interactions between goals, the goal processor can modify KSI ratings. For example, two different hypotheses might cause the creation of two goals that are trying to achieve the same thing—their objectives overlap. Because a KSI that satisfies one may satisfy both goals, the rating of such a KSI is increased so that it is given precedence over other KSIs. The goal processor can also form subgoals of some goals. As an example, the node might form a goal to extend a track at a high blackboard-level, but the data that would extend the track

[1] This is for efficiency reasons, so numerical calculations are done on integers rather than real numbers between 0 and 1.



The basic blackboard-based node architecture is shown.

Figure 8: The Problem Solving Architecture of a Node.

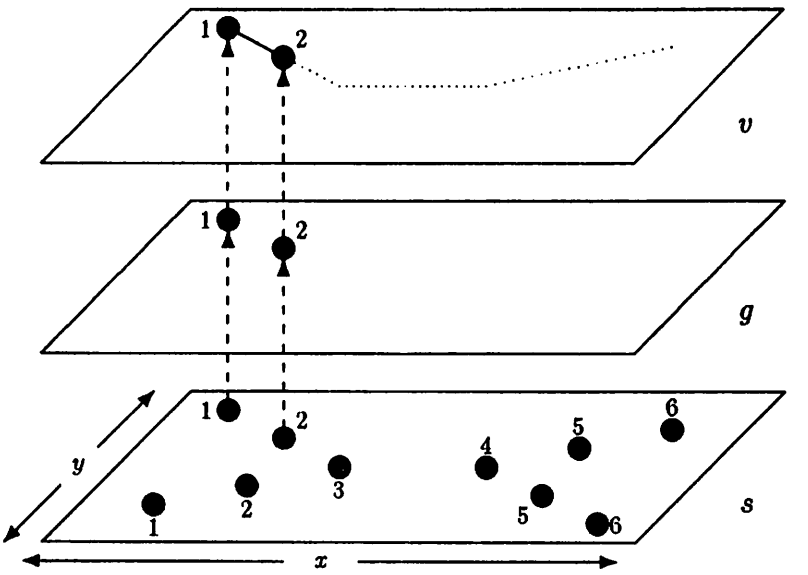
only exists at low blackboard-levels. The goal processor generates subgoals of the high-level goal, where a subgoal represents an intention to develop data that might lead to achieving the high-level goal. The goal processor identifies other goals that overlap with these subgoals, and since the subgoals are highly rated, the KSIs to satisfy the overlapping goals have their ratings increased. Thus, goal processing improves control decisions by increasing the ratings of KSIs that may lead to the achievement of important goals such as extending a highly believed hypothesis.

Control and problem solving are interleaved. After a KS is executed, the goal processor generates subgoals and KSIs. When goal processing is complete, the dispatcher extracts the most highly-rated KSI from the queue and executes the appropriate KS. Thus, the DVMT simulates the nodes' activities as follows.[2] Each node begins by transforming any sensed data into a set of signal location hypotheses (using the sensors KS). With each new signal hypothesis, the node generates goals to improve upon it and forms KSIs to achieve these goals. After it has created all of its signal hypotheses, a node then chooses a KSI to invoke. The KSI invocation may cause the creation of new hypotheses, which stimulate the generation of more goals, which in turn may cause more KSIs to be formed. The node then chooses another KSI and the cycle repeats until the node generates acceptable solutions. The criteria for deciding whether a hypothesis at the pattern blackboard-level represents an acceptable solution is statically defined before problem solving begins: the user specifies some minimum belief, expected spatial and temporal characteristics, etc.[3]

A snapshot of the contents of the signal, group, and vehicle blackboard-levels while solving a sample problem is shown in Figure 9, where each blackboard-level is represented as a surface with spatial dimensions x and y. At the signal blackboard-level s, there are 10 hypotheses, each for a single time-location (the time is indicated for each). Two of these hypotheses have been synthesized to the group blackboard-level g. In turn, these hypotheses have been synthesized to the vehicle blackboard-level v, where they have been connected into a single track hypothesis (graphically, the two locations are linked). Problem solving

[2]More detailed descriptions can be found elsewhere [Corkill, 1983; Lesser and Corkill, 1983].

[3]In many cases, the user specifies exactly the solution that the node should find, but this information is treated as an "oracle" that only tells the node when a solution has been found—the information cannot be used to guide a node's decisions.



Blackboard-levels are represented as surfaces containing hypotheses (with associated sensed times). Hypotheses at higher blackboard-levels are synthesized from lower level data, and a potential solution is illustrated with a dotted track at blackboard-level v.

Figure 9: An Example Problem Solving State.

proceeds from this point by having the goal processing component form goals (and subgoals) to extend this track to time 3 and instantiating KSIs to achieve these goals. The most highly-rated pending KSI is then invoked and triggers the appropriate KS to execute. New hypotheses are posted on the blackboard, causing further goal processing and the cycle repeats until an acceptable track incorporating data at each time is created. One of the potential solutions is indicated at blackboard-level v in Figure 9.

2.4 Coordination and Organization of Nodes

Each node is an independent problem solver and makes its own control decisions. When different nodes have data for tracking the same vehicle, however, they must cooperate to share information so that they form an overall solution. To coordinate, the nodes must have knowledge about each other so that they know how to interact. In the DVMT, the nodes share common knowledge in the form

of an organizational structure.

An *organizational structure* specifies a set of long-term responsibilities and interaction patterns for the nodes. This information guides the local control decisions of each node and increases the likelihood that the nodes will behave coherently by providing a global strategy for network problem solving. In the current implementation of the DVMT, we assume that the organizational structure is established at network creation and is never altered [Pattison *et al.*, 1985].

The organizational structure is implemented in the DVMT as a set of data structures called *interest areas*. As implied by its name, an interest area specifies the node's interest (represented as a set of parameters) in a particular area of the partial solution space (characterized by blackboard-levels, times, locations, and event-classes). A node's interest areas specify its general roles and responsibilities in network problem solving. The goal processor uses the interest areas to modify the ratings of goals, so that goals to generate hypotheses in desirable areas of the blackboard have their ratings increased. Since the goal rating is a factor in rating KSIs, the interest areas can influence node activity, but because there are other factors in rating KSIs (such as the expected beliefs of the output hypotheses), a node still preserves a certain level of flexibility in its local control decisions. The organizational structure thus provides guidance without dictating local decisions, and can be used to control the amount of overlap and problem solving redundancy among nodes, the problem solving roles of the nodes (such as "integrator", "specialist", and "middle manager"), the authority relations between nodes, and the potential problem solving paths in the network [Corkill and Lesser, 1983].

Because nodes know of each other's interest areas, the communication KSIs can decide which hypotheses (and goals) could be of interest to other nodes. The organization defines communication blackboard-levels (to enforce communication of only more fully processed information). When it forms a new hypothesis on a communication blackboard-level, the node checks this hypothesis against its view of the other nodes' interest areas, and if another node may be able to use (extend or refine) the hypothesis, a KSI to send it is formed. When this KSI becomes the most highly-rated KSI on the queue (a separate queue is maintained for communication KSIs [Durfee *et al.*, 1984]), the hypothesis is sent.

Because it affects both the local control decisions of nodes and decisions about how they interact through communication, an organizational structure can guide

nodes into coordinating effectively. When it assigns appropriate responsibilities (interest areas) to nodes, the organization can lead to acceptable network behavior over the long term. However, an organization that is specialized for one short-term situation may be inappropriate for another. Because we assume that network reorganization is costly and time consuming, and since specific problem characteristics cannot be predicted beforehand, an organizational structure should thus be chosen which can achieve acceptable and consistent performance in the long-term rather than being very good in a limited range of situations and very bad in others [Durfee *et al.*, 1985a].

2.5 Specifying Problem Solving Environments

Information about a particular network configuration and about the problem characteristics are contained in an *environment file*. The contents of an environment file (which are described in more detail elsewhere [Corkill, 1983]) are:

global tuning parameters: control parameters indicating such things as the relative weight of goal and data ratings in KSI ratings and whether subgoaling should be performed;

signal grammar: how hypotheses with different event-classes are related at the various blackboard-levels;

movement constraints: the maximum velocity and acceleration of vehicles;

node specifications: the nodes that comprise the network;

interest areas specifications: the problem solving roles of the nodes;

knowledge source assignments: the KSs assigned to each node;

subgoaling specifications: how hypotheses at higher blackboard-levels should be subgoaled to lower blackboard-levels;

sensor assignments: which sensors send data to which nodes;

communication information: the communication delays and error rate between nodes;

solution specifications: the characteristics of desired solutions;

signal data: the simulated sensor data received by each sensor and when it is simulated to arrive.

To experiment with a particular distributed problem solving situation, the user can build an environment file with the suitable characteristics. Because different aspects of the environment can be changed independently, the user can flexibly experiment with many aspects of the distributed vehicle monitoring domain, including the size of the network, the sensor configurations, the control strategies, the signal data to be interpreted, the rate at which that data arrives, and the communication topology. It is this flexibility that makes the DVMT such a valuable research tool: by modifying aspects of the environment, one can simulate an extremely wide range of distributed problem solving scenarios, including situations that would not likely occur in the vehicle monitoring domain but which may be of interest in other domains. The DVMT is thus more general than a testbed for studying distributed vehicle monitoring. In fact, because the goal of our research is to study more general issues distributed problem solving, the emphasis in the DVMT has been on providing opportunities for experimenting with distributed problem solving situations, and has not been on accurately simulating the vehicle monitoring task itself.

2.6 Network Simulation

The DVMT runs on a serial machine, and must interleave node activities to simulate the concurrent actions of the nodes [Durfee *et al.*, 1984]. The DVMT does this by cycling through the nodes and performing a *node execution* for each. A node execution corresponds to a node invoking a single local KS, and includes the activities that go into choosing that KS and that are triggered as a result of that KS. In the DVMT, the execution of a KS takes a certain amount of simulated time. Each time it executes a KS, therefore, a node updates its simulated clock to reflect this time need. At any given time, the DVMT performs a node execution for the node that is furthest behind.

In a simplified view sufficient for understanding the later chapters, a node execution has four basic phases:

1. The sensor information is checked and any KSIs corresponding to the simulated arrival of new sensor data are formed and placed on the queue;

2. Reception KSIs are invoked until all of the data that should arrive before the next local KS is invoked has been incorporated and processed (it might trigger new KSIs);

3. The most highly-rated local KSI is invoked, causing new hypotheses to be placed on the blackboard and triggering goal processing to form new goals and KSIs;

4. Transmission KSIs are invoked to send out any relevant hypotheses formed by the local KSI.

A node execution has a duration corresponding to the amount of simulated time that the node takes to execute. Because this duration depends on the KSI invoked and is unaffected by the control activities (how many goals and subgoals were formed, how many KSI preconditions were run), the simulated time needs of nodes in the DVMT do not completely reflect the node's activities during the node execution. Instead, the duration of the KSI is simply calculated as $ax + b$ where a and b are constants associated with KSs (defined in the environment file) and x is the number of hypotheses produced by the KSI. The DVMT's current inability to simulate the time needs for control limit our ability to measure control overhead in the experiments in chapters 5 and 8.

2.7 Problem Solving Examples

To more completely understand problem solving in the DVMT, we now go through two examples in some detail. In the first example, we consider how a single node processes the data shown in Figure 10a. Data is received for 5 sensed times, and data for two locations exists for sensed time 4. Although not represented in the figure, we assume that each of the data points actually has 5 hypotheses in the same location, where these hypotheses have different event-classes. The event-classes are 2, 6, 10, 14, and 18, corresponding to the signals caused by a vehicle of type 1 (Figure 6). Furthermore, note that the data at times 1, 2, 3, and 5 are strongly sensed, while d_4 is weakly sensed and d_4' is moderately sensed. We

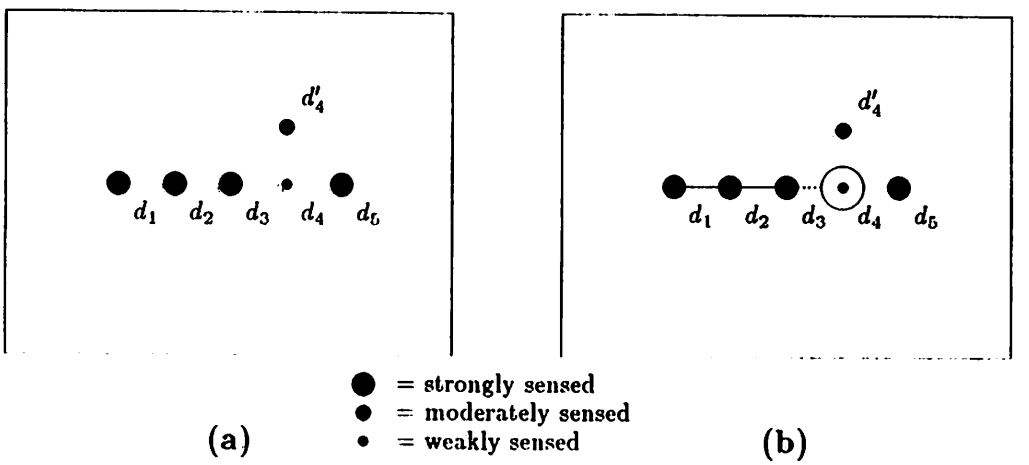


Figure 10: Single Node DVMT Problem Solving Example.

assume that all KSs take 1 time unit to execute, and that the interval between discrete sensed times equals one time unit (although in chapters 5 and 8, we experiment with larger intervals so that nodes can run several KSs before they receive data for the next sensed time).

The sensor KSs run over the first 5 time intervals, from simulated time 1 to simulated time 6. For each of these KSs, the goal processing generates goals based on the new hypotheses and instantiates KSIs to achieve these goals. Because we only allow the node to form tracks at the vehicle and pattern blackboard-levels (by not giving it certain KSs), the node forms KSIs to generate group-level hypotheses from the signal-level hypotheses rather than to generate tracks from these hypotheses. At time 6, the node invokes the most highly-rated of these KSIs, and forms a group-level hypothesis with group event-class 2 out of data in d_1. At time 7, it does the same for data in d_2, at time 8 it does the same for data in d_3, and at time 9 it does the same for data in d_5. It skips over the data for sensed time 4 because it is less strongly sensed (and so less highly believed). The node then invokes KSIs to form the group-level hypothesis with group event-class 6: at time 10 it works in d_1, at time 11 in d_2, at time 12 in d_3, and at time 13 in d_5. Using the group-level hypotheses, it forms vehicle-level hypotheses with vehicle event-class 1: at time 14 in d_1, at time 15 in d_2, at time 16 in d_3, and at time 17 in d_5.

The node then begins to combine the data into tracks. When it had formed the vehicle-level hypothesis at d_2 (time 15), the goal processor built a goal to

combine it with data at d_1 (formed at time 14). Since the KS precondition for forming the track out of separate locations found a vehicle-level hypothesis at d_1 that could be used, the goal processor instantiated a KSI to form d_1–d_2. Similarly, at time 16 a KSI is formed to combine d_3 with d_2. At time 18, the node invokes the first KSI and forms d_1–d_2. The goal processor instantiates a KSI to combine this result with d_3, but because the KSI to form d_2–d_3 was on the queue first, it is invoked at time 19. With this result, the goal processor forms a goal to extend it to d_1, and and forms a KSI to merge these two short tracks. At time 20, the node invokes the KSI to extend d_1–d_2 into d_3 and forms d_1–d_3, and forming this track causes the goal processor to delete the KSI to merge the two short tracks since it would generate redundant results. The node also has d_5 at the vehicle-level, but cannot combine it with anything yet because the data at adjacent sensed times is not at a suitable blackboard-level.

When the node forms d_1–d_3, it generates a goal to extend this track into sensed time 4. Using the vehicle movement constraints, the goal processor is able to identify an area of allowable extensions to this track, shown in Figure 10b. When the goal processor forms subgoals of this goal, these subgoals overlap with goals to process d_4 but not with goals to process d_4'. Before subgoaling, the KSIs to process d_4' were more highly rated than those to process d_4, but because the subgoals increase KSI ratings the opposite is true after subgoaling. Thus, at time 21 the node invokes the KSI to form the group-level hypothesis for group event-class 2 in d_4, and at time 22 it forms the hypothesis for group event-class 6. The node forms the vehicle-level hypothesis for vehicle event-class 1 in d_4 at time 23, and when it does it forms goals to combine this hypothesis with data at both earlier and later sensed times. At time 24, the node forms d_1–d_4. At time 25, it forms d_4–d_5. The node builds a goal to combine these and at time 26 invokes a KSI to generate d_1–d_5. This hypothesis is at the vehicle blackboard-level, while solution specifications indicate that overall problem solutions must be pattern-level hypotheses. The node invokes a KSI to synthesize the track to the pattern-level at time 27. This KSI is completed at time 28, so the simulated solution time is 28.

This one-node example shows the basic activities of a node. It also illustrates how subgoaling can improve performance: without subgoaling, the node would have worked on d_4' before d_4. Because of vehicle movement constraints, it would

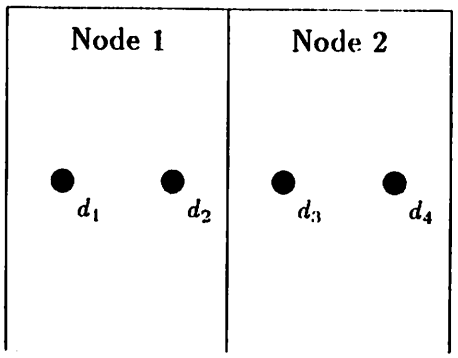


Figure 11: Multi-Node DVMT Problem Solving Example.

have failed when trying to combine d_4' with the other tracks, and would eventually have pursued d_4. Subgoaling allows it to base expectations on highly believed hypotheses and to use these expectations to focus low-level activity. However, the experiment did not reflect the costs of performing subgoaling, since this extra processing does introduce overhead.

We next turn to a simple two-node environment, shown in Figure 11. Each node receives data for just 2 sensed times, and we assume once again that each data point represents 5 hypotheses. KSs again take 1 time unit to execute. We assume that nodes can exchange track hypotheses at the vehicle-track blackboard-level (exchanging only high level tracks can reduce communication).

Both nodes invoke sensor KSs over the first 4 time intervals, even though each node does not necessarily get data for all 4 times. Until it runs the sensor KS for a particular sensed time, a node does not know what if any data has been sensed at that time, so it must run sensor KSs for every sensed time. The problem solving thus begins at time 5. At time 5, node 1 forms the group-level hypothesis at group event-class 2 in d_1 while node 2 forms a similar hypothesis for d_3. Similarly, at time 6 node 1 forms the hypothesis for group event-class 2 in d_2 while node 2 does the same in d_4. At time 7, node 1 forms the hypothesis at group event-class 6 in d_1 while node 2 makes the corresponding hypothesis in d_3, and at time 8 the nodes work in d_2 and d_4 concurrently. The nodes then develop vehicle-level hypotheses for vehicle event-class 1: at time 9, node 1 works in d_1 while node 2 works in d_3, and at time 10 nodes 1 and 2 work in d_2 and d_4 respectively.

At time 11, the nodes form tracks from their vehicle-location hypotheses.

Node 1 builds d_1–d_2 while node 2 builds d_3–d_4. They complete these hypotheses at time 12. Each then checks the goal for extending its local hypothesis against the organizational structure, and determines that the other node is responsible for the area in which the track should be extended. Each node thus sends its hypothesis to the other. Because of communication delays, they will not receive results from each other until time 14. When they invoke local KSIs at time 12, therefore, the nodes cannot continue on their highly-rated data. Because they received data for signal-level hypotheses at signal event-class 18, and because this event-class supports vehicles of type 2 as well (see Figure 6), the nodes begin invoking KSIs to build a vehicle track of event-class 2. When they had formed group-level hypotheses with event-class 6, they also formed hypotheses with event-class 10 (the synthesis KS formed all possible hypotheses from the signal-level hypotheses with event-classes 10, 14, and 18). Because there were no signal-level hypotheses with event-class 22 or 26, these group-level hypotheses have low belief (they lack full support). Moreover, the nodes cannot build any group-level hypotheses with event-class 14 since they have no suitable signal data. At time 12, therefore, node 1 builds the vehicle-level hypothesis with vehicle event-class 2 in d_1, and this has very low belief because it lacks support. Node 2 builds a similar hypothesis for d_3. At time 13, they do the same for d_2 and d_4 respectively.

At time 14, the nodes receive the hypotheses they sent to each other. Each forms goals from its received hypothesis, and forms a KSI to combine them. At time 14, the nodes concurrently form the vehicle-track hypothesis d_1–d_4. Once again, they then invoke synthesis KSs at time 15 to form pattern track hypotheses. Since these KSs end at time 16, both nodes have formed the solution at time 16.

This example shows a very simple example of cooperation among the nodes. Each works independently on its data, but when it forms a transmittable result (at the vehicle-track blackboard-level) it checks the organizational structure to decide whether it should send this hypothesis to the other. An appropriate organizational structure can thus lead nodes to exchange sufficient results so that they can converge on overall solutions.

2.8 Limitations of the DVMT

As mentioned above, the DVMT is a flexible research tool for simulating and experimenting with a wide range of distributed problem solving situations. However, the DVMT does have limitations. Some of these limitations stem from its relatively static view of coordination (through organization) and from the way that nodes make control decisions. As we describe more fully in the next section, the principal focus of this research is on improving control and coordination in the DVMT. However, other limitations that are beyond the scope of this research nonetheless influence how we evaluate the new mechanisms we introduce.

One of these limitations, mentioned before, is the inability to simulate the costs of control. Potentially very costly control activities such as subgoaling and invoking KS preconditions are not adequately represented within the simulated runtime calculations for KSs. We are therefore unable to simulate different relative costs of problem solving and control to determine when control is and is not worthwhile. Because the costs of control are important to its practicality, the empirical evaluation in this research compares the *actual* computation costs of experiments with and without various control mechanisms. The overall computational runtime (CPU time) represents the total time spent on problem solving and control. Thus, although we cannot simulate situations where control is more or less costly, we can use the the computation time to recognize, with the mechanisms as implemented, whether the overhead added by the additional control mechanisms is offset by the savings in time spent on problem solving and goal processing because of better control decisions.

Another limitation is in the representation and calculation of confidence values (beliefs) for hypotheses. The calculations of belief are relatively simple, and a hypothesis's belief is often computed as a weighted average of its supporting hypotheses's beliefs. In particular, the belief computations lack the ability to factor in "independent verification" when computing beliefs. For example, say two nodes form identical hypotheses in an area where their independent sensors overlap. If they exchange these hypotheses, each will set the belief of this hypothesis as the maximum of the beliefs of the two versions. The belief calculations thus do not allow nodes to increase their confidence in results because of corroboration, or for that matter to decrease the beliefs of results because of "negative" evidence.

The DVMT does not reflect the advantages of working together to independently verify results, so we cannot experimentally test coordination mechanisms in such scenarios.

Therefore, the DVMT does have some limitations that affect its use as a tool for experimentation and evaluation in the context of this research. However, these limitations are fairly minor compared to the flexibility that the DVMT does provide: despite its limitations, the DVMT provides a rich framework for experimenting with a wide range of issues in distributed problem solving, and this research makes use of much, but certainly not all, of this versatility.

2.9 How This Work Builds on the DVMT

Local control in the DVMT is based on a combination of data-driven and goal-directed forces. Because nodes typically process their best data in a data-driven manner, they are likely to eventually form highly-believed results. Because they also can subgoal important goals to trigger short sequences of actions (KSIs) to achieve those goals, they are somewhat goal-directed as well. However, the goal-directed component basically works toward relatively short-term goals (single extensions to a hypothesis). As a consequence, a node is never working toward any particular long-term goals: it takes actions that show promise in the near-term under the assumption that near-term promise leads to long-term gains. Because nodes make control decisions with such a short-sighted view, it is common for a node to jump among different partial solutions, doing a little work on one and then moving on to another when the first looks less promising.

Coordination through a static organization can also lead to erratic behavior. Because they only know of each other's general roles and responsibilities, nodes might not coordinate effectively: they can work at cross-purposes (on different overall solutions); they can unnecessarily duplicate each other's work (where their responsibilities overlap); they can distract each other into doing less important activities (by transmitting partial results of lowly-rated overall results, causing other nodes to work on extending those results); and they can fail to adapt to inefficient situations (where some nodes sit idle while others are overburdened). Although the network will coordinate better with a static organization than with no organization at all, the organization might not accommodate the needs of a

particular situation, and the short-term network behavior may be ineffective.

These methods of local control and coordination depend on eventual convergence: although local problem solving might jump among solution paths, the drive toward forming highly believed results will eventually lead a node into generating useful results; and although network activity might be erratic and poorly coordinated, the organizational structure slowly pushes nodes into converging on acceptable solutions. With short-sighted views of control and with general organizations, one cannot expect any more than such functionally-accurate behavior.

When experimenting with the DVMT, however, it became obvious that we need not settle for control and coordination at this level. Nodes could cooperate more effectively if they could communicate about their local activities and *plan* how they should work together. But before they can plan as a group, they need to be able to plan their local activities so that they can exchange plans to converge on more global plans. And before they can plan their local activities, nodes need to develop more long-term views of what solutions they should work toward forming. Because nodes in the DVMT initially could not do any of these things, we have developed mechanisms for identifying local goals though clustering data, then for forming local plans, then for recognizing more global goals, and then for building more global plans--each set of mechanisms building upon the previous ones. These mechanisms are covered in turn in the next chapters.

The first step is to identify possible solutions to work toward. In a blackboard-based problem solver such as a node in the DVMT, this first step is difficult. Because the problem solver explicitly represents complete solutions only *after* it has solved the problem, any planning mechanisms must generate a new representation that allow them to recognize potential solutions in advance. This representation should be formed quickly and inexpensively to minimize the overhead for developing this view, but it still should be adequate for identifying important solutions to work toward. The first phase of this research thus concentrates on using approximate knowledge to cheaply form a representation that allows a node to recognize its long-term goals, and this is addressed next.

"Cheshire Puss," she began ... "would you tell me, please, which way I ought to go from here?"

"That depends a good deal on where you want to get to," said the Cat.

"I don't much care where—" said Alice.

"Then it doesn't much matter which way you go," said the Cat.

"—so long as I get somewhere," Alice added as an explanation.

"Oh, you're sure to do that," said the Cat, "if you only walk long enough."

-Lewis Carrol *(Alice in Wonderland)*

Chapter 3

Identifying Local Goals Through Clustering

A problem solver that more clearly defines its goals can make better decisions about the activities it should pursue. The problem solver can use the specific attributes of a more well-defined goal to constrain the possible steps leading up to that goal: it can back-propagate the desired characteristics of the goal to formulate a specific sequence of subgoals leading to the solution, and in turn can use these well-defined subgoals as context for determining what actions to take. In short, a *goal-directed* problem solver can identify and perform actions that lead toward its well-defined long-term goals, whereas a *data-driven* problem solver takes actions that seem promising based on its current state without any view of the long-term significance of those actions.

In this chapter, we describe mechanisms that allow a blackboard-based problem solver, which typically performs data-driven problem solving, to identify its long-term goals. By building a suitable representation of its problem situation—a representation specifically intended to improve its control decisions—the problem solver can recognize the areas where it should expend effort and how working in those areas can affect its future activities as it pursues its long-term goals. However, it must balance the benefits of having this view against the costs of forming it, and we focus on how a node can build an adequate view by applying approximate knowledge. After presenting background information, this chapter gives an overview of how these mechanisms for building a useful representation work in a problem solver for vehicle monitoring. This overview is followed by a more detailed description of the mechanisms and some examples of how they work. The final section discusses the more general aspects of the mechanisms and how similar mechanisms might be developed in other domains.

3.1 Background

Although goal-directed and data-driven problem solvers both begin with goals to achieve, the specificity of their goals is what sets them apart. A data-driven problem solver begins with the vague and often implicit goal of developing "good" solutions, where goodness is often measured as some rating (belief, confidence) of how well the data associated with the initial problem situation combines to lead to the solution. Because the problem solver essentially begins with no prior knowledge about which possible solutions are more or less likely, it takes any actions that might form promising combinations of data, possibly generating a very large number of such combinations as it explores a variety of solution paths. A goal-directed problem solver has more specific, explicit goals that let it prefer some solution paths to others, so it works more purposefully toward desirable solutions.

Goal-directed problem solving is more purposeful and usually less costly than data-driven problem solving, but it assumes that more specific goals are in fact available to the problem solver. These goals might be provided by the user (usually human) desiring a problem solution, but of course this would defeat the whole purpose of having an autonomous problem solver. Alternatively, the prob-

lem solver might enumerate each of the possible goals and attempt to achieve each in turn. This approach is used in MYCIN, for example, where to diagnose a patient's diseases the problem solver sequentially attempts to confirm whether the patient has each of the diseases known to it [Shortliffe, 1976]. Exhaustively considering each possible goal in this way can be costly, and alternative approaches have combined aspects of goal-directed and data-driven reasoning, attempting to use the data in the problem situation to prune the set of goals considered. For example, Aikins used "prototypical" rules that could hypothesize whether each goal (disease) was feasible based on a rough view of the situation (patient symptoms) [Aikins, 1980]. The NEOMYCIN system extended this idea by using prototypical rules to trigger possible diagnoses, and then reasoning about the relationships between diagnoses when trying to differentiate between them as it worked toward solutions [Clancey and Letsinger, 1981]. As another example, in Hayes-Roth's blackboard architecture for control, the gross characteristics of the problem situation trigger one out of a small number of problem solving strategies to be adopted [Hayes-Roth, 1985].

These methods for combining goal-directed and data-driven reasoning are useful in domains where the set of possible goals can be enumerated (so criteria for when to consider each can be specified) and where this set is relatively small (since each must have its criteria checked in the course of problem solving). Unfortunately, there are many types of problems where the set of possible goals is very large (and possibly not enumerable). For these problems, the specification of goals demands more sophisticated and flexible techniques that can exploit information in the problem situation to generate a set of possible goals for that situation rather than choosing from a previously established set of possible goals.

After initially developing partial solutions, a problem solver can use expectations about how these partial solutions could be enlarged to form specific goals. Through goal processing (forming subgoals), the problem solver can trigger sequences of actions that achieve the goals (enlarge the partial solutions), thereby coupling data-driven and goal-directed processing [Corkill *et al.*, 1982]. Unfortunately, the subgoal sequences represent relatively short-term intentions—single extensions to partial solutions—rather than long-term views of potential overall solutions. Moreover, before subgoals can be formed, a certain amount of purely data-driven processing must generate the initial high-level partial solutions to be

extended. To develop long-term goals about potential overall solutions, and to develop them as early as possible, requires different mechanisms.

To generate a set of possible goals in a complex situation, a problem solver must transform its problem situation (data) into a form that is specifically intended to let it see these goals. For example, a problem solver attempting to interpret an image (perhaps represented as an array of pixel data) usually cannot simply match the data against predefined templates because the set of possible objects and combinations of objects seen in a non-trivial environment can be enormous. The problem solver may be expected to eventually identify each object in the image, but if it can start with a more specific goal such as "determine if the image is a house in a forest" then it has context that lets it focus more quickly on looking for features that fit into this goal. To find this goal in the first place, the problem solver could simplify the data, averaging the attributes of spatially adjacent pixels into a smaller, less exact representation of the same information. By successively doing this, a "visions cone" is created where the most abstracted image data (at the top of the cone) can be matched against simple knowledge of images [Hanson and Riseman, 1978]. For example, if the most abstracted image has a band of blue pixels above a band of green pixels, the problem solver might develop a goal to prove whether the image is in fact of an outdoor scene with blue sky above green forest. By abstracting the data based on knowledge about images (that nearby pixels are more likely to correspond to the same object), the problem solver builds a representation that helps it identify possible goals to work toward—a representation that improves its control decisions.

A vehicle monitoring problem solver must use similar techniques to recognize its local goals. The set of possible goals (to form solutions for each of the different ways that vehicles could move through its sensed area) is extremely large, so rather than explore each possible goal in turn, the problem solver should abstract the data for its current situation to identify potential goals that it might work toward. Conceptually, therefore, the vehicle monitoring problem solver's techniques for identifying its goals are very similar to those of an image interpretation problem solver. However, abstraction in image interpretation can be based on the spatial relationships between pixel data, while the data used in vehicle monitoring has a richer set of relationships, such as temporal, spatial, harmonic (event-class), and signal strength (belief).

By building a suitable abstraction hierarchy, a problem solver can not only identify possible long-term goals, but also can find relationships among these goals. For example, if two goals call for different interpretations of the same data, the goals represent competing solutions since both cannot simultaneously be true. By reasoning about such relationships, the problem solver can approach solving the problem in a more informed way.

Thus, recognizing possible long-term goals in this type of domain exploits relationships in the problem situation (the data). Our mechanisms thus assume that such relationships exist. This chapter emphasizes how a problem solver can use such relationships to identify long-term goals in the vehicle monitoring domain, and more generally in other domains with rich sets of relationships in their data.

3.2 An Overview of the Clustering Mechanisms

Transforming the node's problem situation into a suitable representation for control requires domain knowledge to recognize relationships—in particular, long-term relationships—in the data. This transformation is accomplished by incrementally clustering data into increasingly abstract groups based on the attributes of the data: the hypotheses can be clustered based on one attribute, the resulting clusters can be further clustered based on another attribute, and so on. The transformed representation is thus a hierarchy of clusters where higher-level clusters abstract the information of lower-level clusters. More or less detailed views of the problem situation are found by accessing the appropriate level of this abstraction hierarchy, and clusters at the same level are linked by their relationships (such as having adjacent time frames or blackboard-levels, or corresponding to nearby spatial regions).

A set of knowledge-based clustering mechanisms for vehicle monitoring has been implemented, each of which takes clusters at one level as input and forms output clusters at a new level. Each mechanism uses different domain-dependent relationships, including:

- **temporal relationships:** the output cluster combines any input clusters that represent data in adjacent time frames and that are spatially near enough to satisfy simple constraints about how far a vehicle can travel in

one time unit.

- **spatial relationships:** the output cluster combines any input clusters that represent data for the same time frames and that are spatially near enough to represent sensor noise around a single vehicle.
- **blackboard-level relationships:** the output cluster combines any input clusters that represent the same data at different blackboard-levels.
- **event-class relationships:** the output cluster combines any input clusters that represent data corresponding to the same event-class at a higher blackboard-level.
- **belief relationships:** the output cluster combines any input clusters that represent data with similar beliefs.

The abstraction hierarchy is formed by sequentially applying the clustering mechanisms. The order of application depends on the *bias* of the problem solver: since the order of clustering affects which relationships are most emphasized at the highest levels of the abstraction hierarchy, the problem solver should cluster to emphasize the relationships it expects to most significantly influence its control decisions. This emphasis helps it make control decisions that lead it to efficiently form good solutions.

To illustrate clustering, consider the clustering sequence in Figure 12, which has been simplified by ignoring many cluster attributes such as event-classes, beliefs, volume of data, and amount of pending work; only a cluster's blackboard-levels (a cluster can incorporate more than one) and its time-regions (indicating a region rather than a specific location for a certain time) are discussed. Initially, the situation is nearly identical to that in Figure 9, except that for each hypothesis in Figure 9 there are now two hypotheses at the same sensed time and slightly different locations. In Figure 12a, each cluster c_n^l (where l is the level in the abstraction hierarchy) corresponds to a single hypothesis, and the graphical representation of the clusters mirrors a representation of the hypotheses. By clustering based on blackboard-level, a second level of the abstraction hierarchy is formed with 19 clusters (Figure 12b). As is shown graphically, this clustering "collapses" the blackboard by combining clusters at the previous abstraction level that correspond to the same data at different blackboard-levels.

In Figure 12c, clustering by spatial relationships forms 9 clusters. Clusters at the second abstraction level whose regions were close spatially for a given sensed time are combined. Finally, clustering by temporal relationships in Figure 12d combines any clusters at the third abstraction level that correspond to adjacent sensed times and whose regions satisfy weak vehicle velocity constraints.

The highest level clusters, as illustrated in Figure 12d, indicate four rough estimates about potential solutions: a vehicle moving through regions $R_1R_2R_3R_4R_5R_6$, through $R_1R_2R_3R_4R'_5R'_6$, through $R'_1R'_2R_3R_4R_5R_6$, or through $R'_1R'_2R_3R_4R'_5R'_6$. The problem solver could use this view to improve its control decisions about what short-term actions to pursue. For example, this view allows the problem solver to recognize that all potential solutions pass through R_3 at sensed time 3 and R_4 at sensed time 4. By boosting the ratings of KSIs in these regions, the problem solver can focus on building high-level results that are most likely to be part of any eventual solution.

In some respects, the formation of the abstraction hierarchy is akin to a rough pass at solving the problem, as indeed it must be if it is to indicate where the possible solutions may lie. However, abstraction differs from problem solving because it ignores many important constraints needed to solve the problem. Forming the abstraction hierarchy is thus much less computationally expensive than problem solving, and results in a representation that is too inexact as a solution but is suitable for control. For example, although the high-level clusters in Figure 12d indicate that there are four potential solutions, three of these are actually impossible based on the more stringent constraints applied by the KSs. The high-level view afforded by the abstraction hierarchy therefore does not provide answers but only rough indications about the long-term promise of various areas of the solution space, and this additional knowledge can be employed by the problem solver to make better control decisions as it chooses its next task.

(a)

Cluster	Time-regions	Blackboard-levels	Subclusters
c_1^1	$(1x_1y_1)(2x_2y_2)$	v	–
c_2^1	$(1x_1y_1)$	g	–
c_3^1	$(2x_2y_2)$	g	–
c_4^1	$(1x_1y_1)$	s	–
c_5^1	$(1x_1'y_1')$	s	–
$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_{23}^1	$(6x_6'''y_6''')$	s	–

(b)

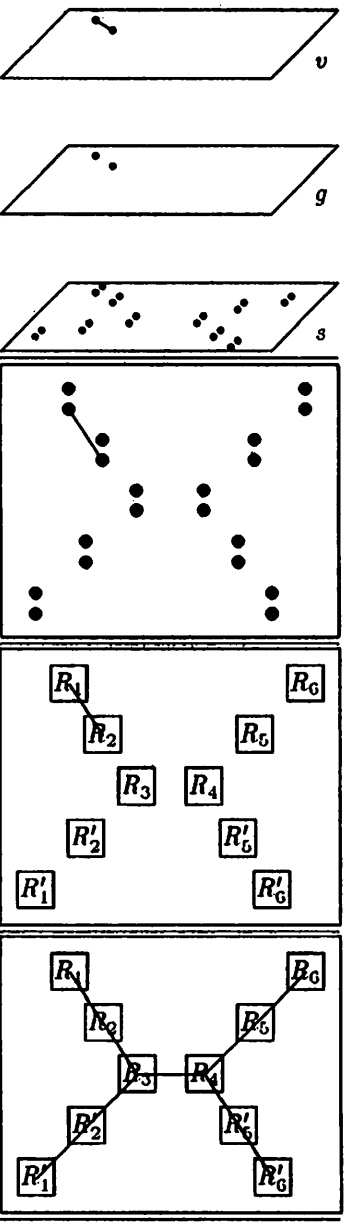
Cluster	Time-regions	Blackboard-levels	Subclusters
c_1^2	$(1x_1y_1)(2x_2y_2)$	s, g, v	$c_1^1, c_2^1, c_3^1, c_4^1, c_8^1$
c_2^2	$(1x_1'y_1')$	s	c_5^1
c_3^2	$(2x_2'y_2')$	s	c_6^1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_{19}^2	$(6x_6'''y_6''')$	s	c_{23}^1

(c)

Cluster	Time-regions	Blackboard-levels	Subclusters
c_1^3	$(1R_1)(2R_2)$	s, g, v	c_1^2, c_2^2, c_3^2
c_2^3	$(1R_1')$	s	c_3^2, c_4^2
c_3^3	$(2R_2')$	s	c_6^2, c_7^2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_9^3	$(6R_6')$	s	c_{18}^2, c_{19}^2

(d)

Cluster	Time-regions	Blackboard-levels	Subclusters
c_1^4	$(1R_1)(2R_2)(3R_3)$ $(4R_4)(5R_5)(6R_6)$	s, g, v	$c_1^3, c_4^3, c_5^3,$ c_6^3, c_8^3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_4^4	$(1R_1')(2R_2')(3R_3)$ $(4R_4)(5R_5')(6R_6')$	s	$c_2^3, c_3^3, c_4^3,$ c_5^3, c_7^3, c_9^3



A sequence of clustering steps is illustrated both with tables (left) and graphically (right). c_i^l represents cluster i at level l of the abstraction hierarchy. In (a), each cluster is a hypothesis. These are clustered by blackboard-level to get (b); note that graphically the levels have been collapsed into one. These clusters are then grouped by spatial relationships to form (c), which in turn is clustered by temporal relationships to form (d).

Figure 12: An Example of Incremental Clustering.

3.3 The Clustering Mechanisms in Detail

The mechanisms for developing an abstract view of the problem situation evolved considerably over the course of this research. Initially, the abstraction hierarchy was intended to emphasize "problem solving situations" that could be planned for. Thus, the hierarchy was constructed until such a situation was recognized, at which point a planning mechanism developed for that situation was employed to find actions (KSIs) suitable for the situation. The mechanisms were therefore extensions to earlier research that abstracted the problem state and made plans in response to particular situations [Durfee *et al.*, 1985a], where the extensions added flexibility in how abstractions and plans were formed.

As the more sophisticated planning techniques that are outlined in later chapters were introduced, many of the mechanisms initially employed in forming the abstraction hierarchy became unnecessary, either because other planning mechanisms took on tasks initially performed during abstraction or because the planning mechanisms, in their current state of development, assumed that the abstraction hierarchy had a certain form, so that the flexibility for abstracting in different ways became unused. This section describes the abstraction data structures and the mechanisms for forming clusters and the clustering hierarchy as currently implemented in the DVMT's planner, and does not address how the mechanisms evolved or how their capabilities could be extended.

3.3.1 The Clustering Data Structures

The clustering hierarchy is composed of individual clusters (Figure 13), where each cluster summarizes the characteristics of some set of hypotheses and indicates how this set of hypotheses is related to other sets (clusters). The clusters are grouped together into levels of abstraction, depending on how much abstraction has been applied, and clusters point to related clusters on the same level and on adjacent levels.

To begin with, the clustering hierarchy is simply a list:

$$(clusters_n \quad clusters_{n-1} \quad \ldots \quad clusters_i \quad \ldots \quad clusters_1)$$

where $clusters_i$ is a set of clusters at level i of the clustering hierarchy. In fact, $clusters_i$ is a pointer to the first cluster in level i of the clustering hierarchy, and

it points to the next cluster, and so on. The linked-list of clusters is not ordered in any particular way (ordered in the order they were generated); using linked-lists simplifies how a level of the clustering hierarchy is represented and maintained.

The structure of an individual cluster is based on how it can be related to other clusters. In the previous section that gave an overview of clustering are listed a set of relationships: temporal, spatial, blackboard-level, event-class, and belief. For some of these relationships, either two clusters are related or they are not—two clusters either represent the same data at different blackboard-levels or they do not. Other relationships depend on additional knowledge, and often knowledge used by the KSs as well. For example, how close the locations of hypotheses at adjacent sensed times must be for them to be temporally related (that is, how close must they be to conceivably be the same vehicle moving over time).

The parameters controlling such relationships are contained in the cluster-data structure. These parameters represent approximate knowledge about the domain, and for vehicle monitoring they are:

forward-backward-distance Maximum distance between regions covered by clusters at adjacent sensed times for those clusters to be related temporally (the clusters could represent the same vehicle moving over time).

near-to-distance Maximum distance between regions covered by clusters at the same sensed times for those clusters to be related spatially (the clusters could represent spatially displaced data about the same vehicle at a given sensed time).

belief-range For hypotheses whose time-locations and blackboard-levels match, specifies how close their beliefs must be for them to be grouped together in the same cluster.

A cluster then has two principal types of information: information about the data that it summarizes and information about its relationships with other clusters (Figure 13). The information about the data includes the time-regions covered by the hypotheses (their combined time-locations), their blackboard-levels, their event-classes, and the range of beliefs they cover. The information about relationships includes clusters related by time (both forward (later) and backward (earlier)), by space (near-to clusters), by blackboard-level (levels above and

below), and by belief (greater and lesser belief). Also, a cluster is linked to other clusters at the same level of the clustering hierarchy, and is related to clusters at adjacent levels of the clustering hierarchy, pointing to its sub-clusters (clusters at the level below that this cluster subsumes) and its super-clusters (clusters at the level above that subsume this cluster).

data-list	A list of the hypotheses summarized in the cluster.
blackboard-levels	A list of the blackboard-levels covered by the data.
time-regions	The combined time-locations of the data.
event-classes	A list of the event-classes of the data.
belief-range	A range (lowest-highest) of beliefs covered by the data.
past-work	A list of previously invoked KSIs that were stimulated by the hypotheses in the data-list.
pending-work	A list of not yet invoked KSIs that were stimulated by the hypotheses in the data-list.
vehicle-event-classes	A list of event-classes at a high blackboard-level (the vehicle level) that could potentially be derived from the event-classes of the data.
forward-cluster-list	A list of clusters where each contains data that could extend the cluster's data forward in time.
backward-cluster-list	A list of clusters where each contains data that could extend the cluster's data backward in time.
near-to-cluster-list	A list of clusters where each contains data that is for the same sensed time and for nearby locations as some of the cluster's data.
greater-cluster-list	A list of clusters where each has similar blackboard-level and time-region characteristics to the cluster and where the belief-range of each is greater than the cluster's belief-range.
lesser-cluster-list	A list of clusters where each has similar blackboard-level and time-region characteristics to the cluster and where the belief-range of each is less than the cluster's belief-range.
above-cluster-list	A list of clusters where each summarizes some hypotheses that may be derived from (are on the adjacent blackboard-level above) some hypotheses in the cluster's data-list.
below-cluster-list	A list of clusters where each summarizes some hypotheses that may have been used to derive (are on the adjacent blackboard-level below) some hypotheses in the cluster's data-list.
next-cluster	The next cluster in the doubly-linked list of clusters.
previous-cluster	The previous cluster in the doubly-linked list of clusters.
super-cluster-list	A list of clusters at the next higher level of the clustering hierarchy where each subsumes this cluster.
sub-cluster-list	A list of clusters at the next lower level of the clustering hierarchy where each is subsumed by this cluster.

Figure 13: The Contents of a Cluster.

3.3.2 Forming the Initial Clusters

The base-clusters (clusters at the bottom of the clustering hierarchy) are constructed directly from the hypotheses on the blackboard. The initial set of these clusters is formed when the node has some hypotheses on its blackboard and must make control decisions about which KSs to invoke to process those hypotheses.

The base-clusters group hypotheses together to a very small extent. To be clustered together at this level, hypotheses must have identical time-location-lists, be at exactly the same blackboard-level, have beliefs within the range specified in the cluster-data structure, and have a common vehicle-event-class. This last requirement means that the clustered hypotheses indicate that the same type of vehicle may have been detected. To determine vehicle-event-classes for a given hypothesis, a simplified version of the signal grammar that maps each event-class at an arbitrary blackboard-level to a set of possible event-classes at the vehicle blackboard-level is employed. This simplified version of the grammar knowledge ignores aspects such as how different event-classes collaborate and how much support they give each other, but is sufficient for grouping hypotheses that could potentially support each other.

The node maps through all of the hypotheses on its data blackboard, and for each first determines the vehicle-event-classes of that hypothesis. For each vehicle-event-class, the node determines whether the hypothesis fits into an already existing cluster (with matching vehicle-event-classes, time-locations, levels, and a belief-range that could be extended to fit the hypothesis's belief without exceeding the limits expressed in the cluster-data). If so, the cluster is updated accordingly (its data-list includes the hypothesis, its event-classes include the hypothesis's event-class, its belief-range is extended suitably). Otherwise, a new cluster is formed for the hypothesis and vehicle-event-class combination, the values for its characteristics are taken straight from the hypothesis, and it is inserted in the list of clusters at this clustering level. The algorithm is summarized in Figure 14.

After all of the clusters have been formed, the next step is to determine their relationships to each other. For each pair of clusters, their attributes are compared to determine whether any of the relationships hold. Figure 15 gives examples of how relationships between clusters are found. Cluster-1 and cluster-2 share some time-regions, have vehicle-event-classes in common, but have different

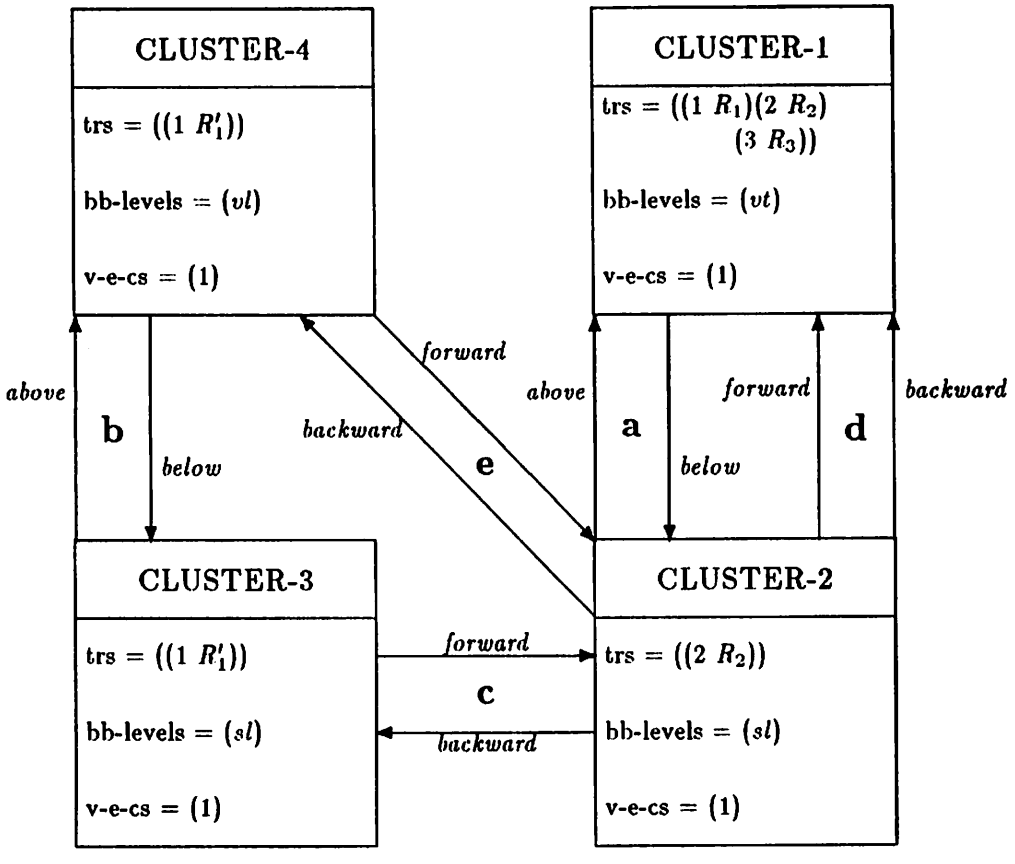
For each hypothesis:

1. Find possible vehicle-event-classes
2. For each possible vehicle-event-class:
   (a) Find existing clusters with vehicle-event-class
   (b) If hypothesis matches cluster attributes insert it into cluster and update cluster
3. For each vehicle-event-class where hypothesis did not match existing cluster, generate a cluster for the hypothesis and vehicle-event-class.

Figure 14: Building the Initial Clusters from Hypotheses.

blackboard-levels, so they are linked by blackboard-level relationships indicating that cluster-1 is *above* cluster-2 and cluster-2 is *below* cluster-1. (Figure 15, relationship a). Similarly, cluster-4 is above cluster-3 and cluster-3 is below cluster-4 (Figure 15, relationship b).

Once cluster can extend another forward in time if the time-regions of the clusters overlap (have one or more time-regions in common) or abut (the time of one cluster's last time-region is adjacent to the time of the other cluster's first time-region and their regions fall within the forward-backward-distance specified in the cluster-data structure). In Figure 15, relationship c, the time-regions of cluster-2 abut with those of cluster-3, and since the regions at their adjacent time-regions are within the forward-backward-distance and their vehicle-event-classes intersect, cluster-3 points to cluster-2 as a forward cluster and cluster-2 points to cluster-3 as a backward cluster. In the same way, cluster-4 points to cluster-2 as a forward cluster and cluster-2 points to cluster-4 as a backward cluster. Relationships need not be symmetric, however. For example, in Figure 15, relationship c, cluster-1 and cluster-2 have overlapping time-regions. Cluster-2 points at cluster-1 as a forward cluster (since cluster-1 shares time-region (2 R_2) and also has time-region (3 R_3)), but cluster-1 does not point to cluster-2 as a backward cluster since cluster-2 cannot extend cluster-1's time-regions backward in time. Similarly, cluster-2 points to cluster-1 as a backward cluster but the reciprocal relationship does not hold. Finally, note that cluster-1 is not related temporally



Four clusters are shown, indicating some of their attributes and the relationships between them (marked with letters).

Figure 15: Sample Relationships Between Clusters.

with either cluster-3 or cluster-4 because of its different region at sensed time 1.

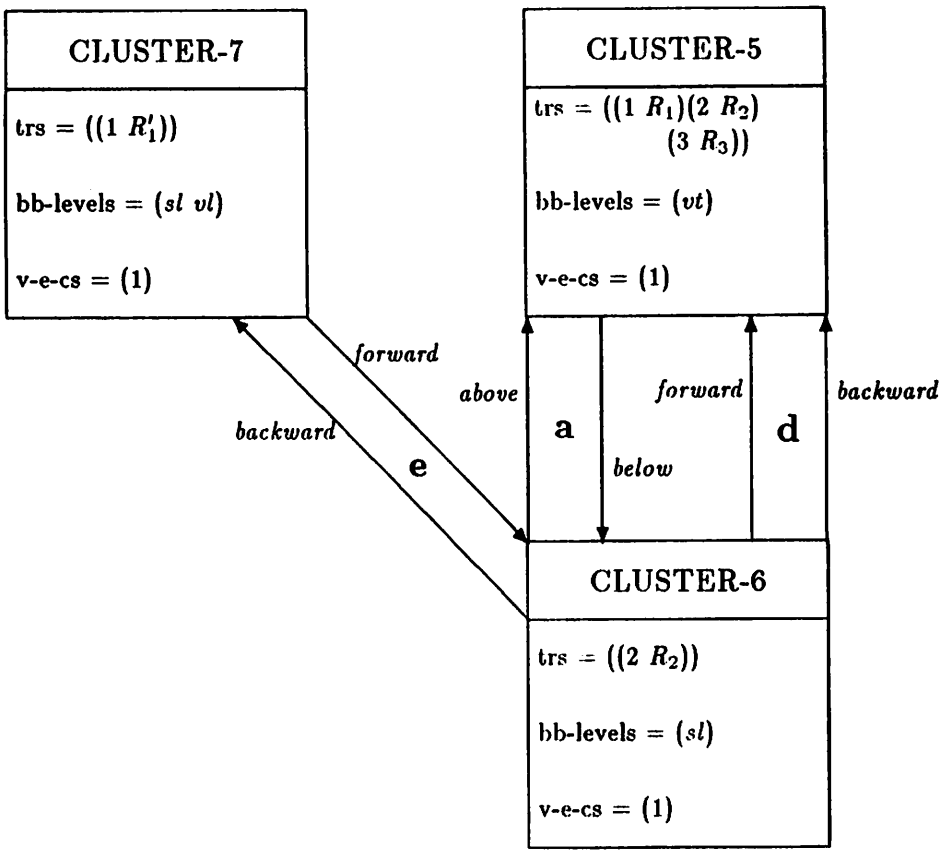
Because there are on the order of n^2 pairs of clusters given n clusters, and because the relationships between clusters may be asymmetric, the determination of relationships can be a costly procedure as the number of clusters increases. Even by implementing the procedures to check less costly requirements for a relationship first (so that failure to relate clusters will be found as cheaply as possible), the volume of comparisons between clusters can be quite large. Fortunately, only the relationships between the base-clusters need to be found in this way—clusters at higher levels of the clustering hierarchy can inherit relationships from the lower levels as will be described shortly.

3.3.3 Forming the Clustering Hierarchy

The clustering hierarchy is generated by successively grouping together clusters at one level to form new clusters at the next higher level.[1] A set of functions for merging clusters based on various relationships has been implemented, where each function takes as input the cluster-set for one level and returns a cluster-set for the next level. In essence, each function concentrates on one relationship that differentiates clusters from each other in the input cluster-set and "ignores" that relationship in the output cluster-set, merging clusters together which are no longer different when the particular relationship is ignored. Each function concentrates on different relationships, and the current set of functions are merge-by-level, merge-by-time, merge-by-time-split, merge-by-near-to, and merge-by-event-class.

Merge-by-level. This merges clusters that correspond to the same information at different blackboard-levels, where these clusters point at each other in their above-cluster and below-cluster relationships. The purpose of merging by level is to no longer differentiate between clusters based on the blackboard-level of their data. Clusters linked by the above-below relationship may only be clustered together if their other relationships are consistent. For example, in Figure 15, cluster-3 and cluster-4 can be merged by blackboard-level relationships

[1] Even when the attributes of a cluster are unchanged from one level of the clustering hierarchy to the next, we still form a new cluster because the new cluster will have different relationships to other clusters.



The clusters in Figure 15 have been clustered based on blackboard-level relationships. The resulting clusters at the new level of the clustering hierarchy and their relationships are shown.

Figure 16: Clusters Merged by Level.

since their other pointers are consistent (they both point to cluster-2 as a forward cluster), but cluster-2 and cluster-1 cannot be merged by blackboard-level since their forward-backward pointers are inconsistent. More specifically, if cluster-1 and cluster-2 were merged, their merged time-regions ((1 R_1) (2 R_2) (3 R_3)) could not be related to the time-regions of cluster-3 or cluster-4 since they have different regions for sensed time 1. Merging them would thus lose the possible combination of cluster-3 and cluster-2 to form time-regions ((1 R_1') (2 R_2)). The result of applying merge-by-level to the clusters in Figure 15 is shown in Figure 16.

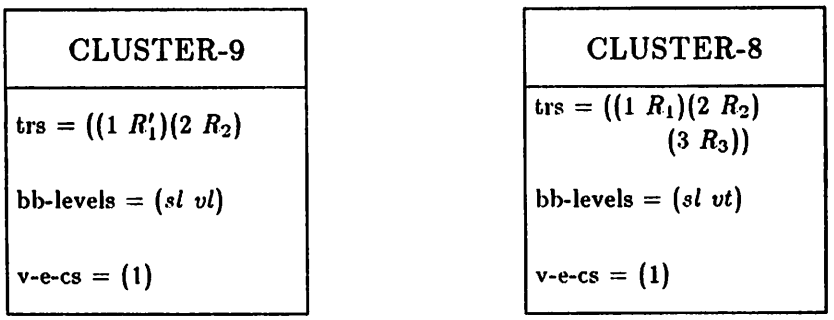
Merge-by-time. This merges clusters that are linked *unambiguously* by forward-backward relationships. If cluster-a can only be extended forward (back-

ward) by cluster-b, and cluster-b either can be extended backward (forward) by cluster-a or cannot be extended backward (forward) by any clusters, then the clusters can be combined unambiguously into a single cluster with a possibly longer set of time-regions. The purpose of merging by time is to recognize sequences of time-regions (tracks) that could not be part of alternative sequences. For example, if in Figure 16 there were no cluster-7, then cluster-5 and cluster-6 could be merged by time. Similarly, if there were no cluster-5, cluster-6 and cluster-7 could be merged by time. However, because cluster-6 is temporally related to both cluster-5 and cluster-7, and because cluster-5 and cluster-7 have incompatible time-regions, merge-by-time will not change the situation shown in Figure 16: the cluster-set returned by this function in this case mirrors the cluster-set it received as input.

Merge-by-time-split. This merges clusters that are linked by forward-backward relationships and finds all possible sequences of time-regions—all possible tracks—by splitting up ambiguous sequences into a set of specific, unambiguous sequences.[2] In the situation of Figure 16, for example, the function takes the three clusters as input and returns two clusters as output: one cluster that combines cluster-7 and cluster-6 and another cluster that combines cluster-5 and cluster-6. The result is shown in Figure 17. Note that the two resulting clusters, cluster-8 and cluster-9, are not related: they represent alternative tracks. Furthermore, the function to merge-by-time-split can in the worst case (a highly ambiguous situation) create a very large number of output clusters because the same input cluster could map to a number of output clusters depending on how many possible sequences of time-regions it can participate in.

Merge-by-near-to. This merges clusters which point at each other by the near-to (spatial) relationship—a symmetric relationship—and which have consistent other relationships such as forward-backward relationships (as were considered in the merge-by-level function). The purpose of merging by near-to is to ignore minor spatial offsets in the data (possibly caused by inexact detection by the sensor). Note that this function only considers the "nearness" of pairs: if cluster-

[2]This description accounts for the name of this clustering mechanism in the DVMT. More generally, the mechanism represents the need to generate alternative clusters when a single cluster can be part of multiple, incompatible combinations.



The clusters in Figure 16 have been clustered based on temporal relationships. The resulting clusters at the new level of the clustering hierarchy and their relationships are shown.

Figure 17: Clusters Merged by Time-split.

a is near cluster-b which is near cluster-c, then the three are clustered together even if cluster-a is not near cluster-c. The assumption underlying this simple (and inexpensive) clustering criterion is that noisy data will center around actual signals rather than being distributed randomly in space.

Merge-by-event-class. This merges clusters that have consistent time-regions and overlapping blackboard-levels but that have different vehicle-event-classes (recall that hypotheses with event-classes that could potentially contribute to the same vehicle-event-class are clustered in the base-clusters). The purpose of merging by event-class is to ignore the particular type of the vehicle being tracked.

The clusters returned by one of these functions represent the next level of the clustering hierarchy. Before they can be passed to the next function to be merged by another relationship, they must be related to each other. Recall that the base-clusters are related to each other by considering how each pair is related. Above the bottom level of the clustering hierarchy, however, such a costly set of comparisons need not be performed. Instead, the relationships between the new clusters are inherited from their sub-clusters. For example, to find the forward-clusters of a new cluster, the new cluster's sub-clusters are found, the forward-

clusters of the sub-clusters are retrieved, and the super-clusters of these clusters are found. If there are any super-clusters, then the new cluster will point at each as a forward-cluster (except if one of the super-clusters is the same as the new-cluster, which can occur when clusters are merged by time or time-split).

The clustering hierarchy is thus formed by applying the functions in some order, and the form that the hierarchy takes depends on that order. At higher levels of the clustering hierarchy, more relationships in the data are ignored so that those relationships that have yet to be merged are emphasized. That is, the mechanisms that use the clustering hierarchy typically start at the highest, most abstract level and work their way down to lower levels until the desired relationship in the data is recognized. Therefore, relationships clustered later are more prominent (seen at higher levels) in the clustering hierarchy. It is thus important to generate the clustering hierarchy with a view to the relationships that are most important to planning. If the planner were rewritten to focus on other relationships, however, the clustering mechanisms would simply have to be applied in a different order to form a suitable clustering hierarchy.

The planner, whose implementation is described in the next chapter, emphasizes building solution tracks by processing data in an area (time-region) and then using the results to extend any solution track currently being developed. The most important relationships therefore are temporal and spatial, with blackboard-level and event-classes taking a back seat. The clustering functions are applied in the order: merge-by-event-class, merge-by-blackboard-level, merge-by-near-to, merge-by-time, and merge-by-time-split. By eliminating event-class, then blackboard-level, and then near-to relationships, the number of clusters that have to be considered during the more costly temporal merging steps can be significantly reduced. Merging by time then groups together clusters that form unambiguous partial tracks, and finally merging by time-split generates clusters representing potential complete tracks. These clusters correspond to *potential* solutions that the problem solver can work toward, but because clusters are developed using only approximate domain knowledge, they may not represent viable solutions. Clustering thus allows nodes to cheaply develop rough views of the data that they use to improve control decisions, but they must use their KSs to generate actual solutions from the data. As will be seen in the next chapter, having information about unambiguous partial tracks can be useful for finding

areas where potential solution tracks share information, and so, places to initially investigate that avoid commitment to a single potential solution.

3.3.4 Updating the Clustering Hierarchy

In any dynamic environment, problem information will change over time. A vehicle monitoring problem solver, for example, might get new data from its sensors or might receive data from other problem solvers. To adequately represent the problem situation, therefore, the clustering hierarchy must be updated when relevant new information arrives at the node.

In the preliminary research, every new hypothesis was incorporated into the abstraction, whether the hypothesis was formed from local activity or from received information from sensors or other problem solvers [Durfee *et al.*, 1985a]. In the current implementation, the mechanisms do not need to incorporate hypotheses formed from local activity into the clustering hierarchy, because these hypotheses do not alter the scope of potential solutions to be pursued. That is, the locally generated hypotheses all fall within the scope of clusters already in the clustering hierarchy. Hence, the clustering hierarchy is only updated when new information is received from the sensors or from another problem solver.

The initial approach to updating the clustering hierarchy was simply to discard the existing hierarchy and to build a completely new one. Although a simple strategy, this approach proved very expensive: although the new information might alter certain clusters and relationships between clusters, most of the clustering hierarchy was often unaffected by the new information. Discarding all of this still valid information and deriving it all over again was very inefficient, so new mechanisms were added for appropriately modifying the base-clusters and then propagating any changes to higher levels of the abstraction hierarchy.

The first step is to map through the new hypotheses and either incorporate each into existing clusters or form new clusters. The functions used are the same as those described in Section 3.3.2 above. The next step is to take any new or modified clusters and determine how they relate to each of the other clusters. Thus, once again a pairwise comparison is made between base-clusters but this time only pairs involving new or modified clusters need be considered (since relationships between pairs of unaltered clusters will not change). When the set of base-clusters has been fully updated, there will be some set of new-or-

modified-clusters, including clusters whose data-lists have not changed but whose relationships with other clusters have.

The clustering functions for merging based on various relationships are then once again employed, but this time they are given a list of the new-or-modified-clusters of the input cluster-set. When merging clusters to build the updated output cluster-set, only the new-or-modified-clusters are considered. These clusters might in turn cause new-or-modified-clusters to be formed in the output cluster-set, and this new list of new-or-modified-clusters is passed on to the next clustering function. Therefore, when updating the clustering hierarchy, the mechanisms only work with any clusters that have changed, leaving the old clusters and relationships alone. In this way, the amount of effort expended to update the clustering hierarchy is much less than starting from scratch and rediscovering many of the relationships in the data.

3.3.5 Identifying Local Goals

The clusters at the highest level of the clustering hierarchy—clusters that correspond to potential solution tracks—represent the long-term goals of the problem solver. An *alternative-goal* is developed for each of these clusters, and the alternative-goal points to the cluster upon which it is based and points to any *competing* alternative-goals. Two alternative-goals are competing if they specify alternative solutions which cannot both be true at the same time. For example, if two tracks have some common time-regions, then both cannot be simultaneously valid solutions because that would imply two vehicles being in the same place at the same time. To recognize competing alternative-goals, the mechanisms simply compare the sub-clusters of the alternative-goals' clusters, and if they have common sub-clusters then the alternative-goals are competing.

An alternative-goal is a goal for local problem solving. The clustering mechanisms allow the problem solver to identify the alternative-goals. The planning mechanisms outlined in the next chapter determine what sequence of actions should be taken to recognize which alternative-goals are most important to achieve and to achieve them.

3.3.6 Some Clustering Examples

Some simple examples of clusters and clustering were presented in Section 3.3.3. In this section are more complete examples for sample situations that are still relatively simple but that demonstrate important features of the clustering mechanisms. The first example shows a clustering hierarchy for a small set of typical initial hypotheses, and then how new hypotheses that extend the potential tracks are incorporated. The second, slightly more elaborate example shows how a clustering hierarchy changes when new data joining previously unrelated clusters is incorporated.

Clustering Example 1

A problem solver receives sensory information and creates hypotheses at the signal blackboard-level (*sl*) based on this information. A vehicle passing through the sensed area typically generates a set of signals—frequency event-classes—and a separate hypothesis is generated for each. For example, if a vehicle of type 1 passes through the area, it generates several frequencies corresponding to signal blackboard-level event-classes 2, 6, 10, 14, and 18 (see Figure 6), usually all detected at the same location. Each of these different signals is represented as a separate hypothesis at the signal blackboard-level.

The base-clusters group together hypotheses with the same time and location attributes and that could contribute to a compatible vehicle type. Therefore, the base-clusters in Figure 18 that have vehicle-event-class 1 each have five hypotheses in their data-list. However, because a hypothesis at signal event-class 18 is also a characteristic of vehicles of type 2, a separate cluster is made for this possibility at each time location. Therefore, two base clusters are formed for each of the three distinct time-regions in Figure 18.

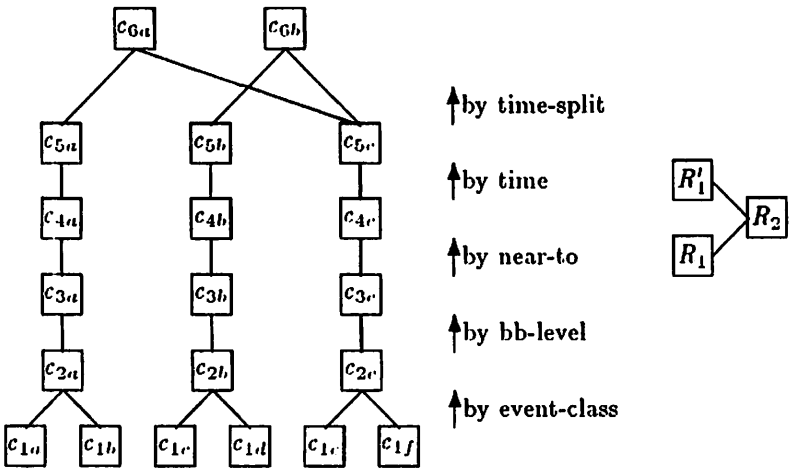
The first step in clustering, merge-by-event-class, brings together the pairs of clusters for separate vehicle-event-classes, and the vehicle-event-classes attribute of these clusters reflects both possibilities. The next step, merge-by-level generates copies of the clusters at the next level since all of the clusters contained data at the same blackboard-level (*sl*). Merge-by-near-to is applied next, and again generates copies of the clusters at the next clustering hierarchy level, this time because there is no spatial ambiguity. No unambiguous sequences of clusters over

time exist (because (2 R_2) can be linked to either cluster for sensed time 1), so merge-by-time copies the clusters to the next level. Finally, merge-by-time-split generates two clusters, one for each of the potential tracks for sensed times 1 and 2.

Now consider what happens when new sensor information arrives for sensed time 3, as shown in Figure 19. Once again, a distribution of frequencies are detected so 5 hypotheses are formed at the *sl* blackboard-level. However, a sensor error has caused some of the hypotheses to be spatially displaced. The new hypotheses, when incorporated into the base-clusters, trigger the formation of 3 new clusters: one for the hypotheses in region R_3' that are relevant for vehicle-event-class 1, one for hypotheses in region R_3'' that are relevant for vehicle-event-class 1, and one for the hypothesis with signal event-class 18 which is relevant for vehicle-event-class 2.

These new base-clusters are then used to update the clustering hierarchy. The merge-by-event-class function combines the two clusters for the same region but with different vehicle-event-classes. The merge-by-level function again has no effect in this situation. The merge-by-near-to function combines the clusters for R_3' and for R_3'' because the two regions are so close spatially that in all likelihood they probably correspond to the same vehicle. Their regions are merged into the larger region R_3. Merge-by-time combines the cluster for sensed time 2 with the new cluster for sensed time 3 because there is no ambiguity: based on the data, a vehicle at (2 R_2) could only have gone to (3 R_3) and a vehicle at (3 R_3) could only have come from (2 R_2). Finally, merge-by-time-split takes the new cluster proposing a track for sensed times 2 and 3 and builds a cluster for each of the two ways the track could be extended backward to sensed time 1.

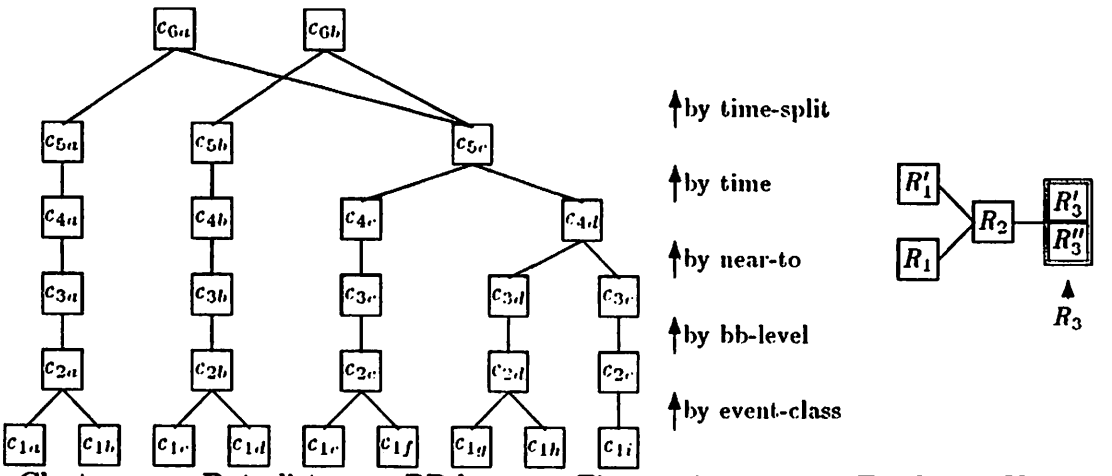
This example therefore illustrates how all of the clustering functions work (except merge-by-level which is covered in the next example) and how initial hypothesis data is transformed into a clustering hierarchy where the top-level clusters represent potential solutions to work toward. The example also indicates how new data that extends the potential solution tracks is incorporated into the clustering hierarchy.



Cluster	Data-list	BB-levs	Time-regions	Ev-cls	V-e-cs
c_{1a}	$(h_1\ h_2\ h_3\ h_4\ h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1)
c_{1b}	(h_5)	(sl)	$((1\ R_1))$	(18)	(2)
c_{1c}	$(h_6\ h_7\ h_8\ h_9\ h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1)
c_{1d}	(h_{10})	(sl)	$((1\ R_1'))$	(18)	(2)
c_{1e}	$(h_{11}\ h_{12}\ h_{13}\ h_{14}\ h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1)
c_{1f}	(h_{15})	(sl)	$((2\ R_2))$	(18)	(2)
c_{2a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{2b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{2c}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{3a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{3b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{3c}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{4a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{4b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{4c}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{5a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{5b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{5c}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{6a}	$(h_1 - h_5,\ h_{11} - h_{15})$	(sl)	$((1\ R_1)\ (2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{6b}	$(h_6 - h_{15})$	(sl)	$((1\ R_1')\ (2\ R_2))$	(2 6 10 14 18)	(1 2)

Clustering hierarchy is shown graphically above left, and the relationships between regions is shown above right. For each cluster is given the hypotheses it summarizes, their blackboard-levels, time-regions, and event-classes, and the vehicle-event-classes that the clustered data supports.

Figure 18: Clustering Example 1 Initial Hierarchy.



Cluster	Data-list	BB-levs	Time-regions	Ev-cls	V-e-cs
c_{1a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1)
c_{1b}	(h_5)	(sl)	$((1\ R_1))$	(18)	(2)
c_{1c}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1)
c_{1d}	(h_{10})	(sl)	$((1\ R_1'))$	(18)	(2)
c_{1e}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1)
c_{1f}	(h_{15})	(sl)	$((2\ R_2))$	(18)	(2)
c_{1g}	$(h_{18} - h_{20})$	(sl)	$((3\ R_3'))$	(10 14 18)	(1)
c_{1h}	(h_{20})	(sl)	$((3\ R_3'))$	(18)	(2)
c_{1i}	$(h_{16}\ h_{17})$	(sl)	$((3\ R_3''))$	(2 6)	(1)
c_{2a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{2b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{2c}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{2d}	$(h_{18} - h_{20})$	(sl)	$((3\ R_3'))$	(10 14 18)	(1 2)
c_{2e}	$(h_{16}\ h_{17})$	(sl)	$((3\ R_3''))$	(2 6)	(1)
c_{3a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{3b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{3c}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{3d}	$(h_{18} - h_{20})$	(sl)	$((3\ R_3'))$	(10 14 18)	(1 2)
c_{3e}	$(h_{16}\ h_{17})$	(sl)	$((3\ R_3''))$	(2 6)	(1)
c_{4a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{4b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{4c}	$(h_{11} - h_{15})$	(sl)	$((2\ R_2))$	(2 6 10 14 18)	(1 2)
c_{4d}	$(h_{16} - h_{20})$	(sl)	$((3\ R_3))$	(2 6 10 14 18)	(1 2)
c_{5a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{5b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{5c}	$(h_{11} - h_{20})$	(sl)	$((2\ R_2)\ (3\ R_3))$	(2 6 10 14 18)	(1 2)
c_{6a}	$(h_1 - h_5,\ h_{11} - h_{20})$	(sl)	$((1\ R_1)\ (2\ R_2)\ (3\ R_3))$	(2 6 10 14 18)	(1 2)
c_{6b}	$(h_6 - h_{20})$	(sl)	$((1\ R_1')\ (2\ R_2)\ (3\ R_3))$	(2 6 10 14 18)	(1 2)

Clustering hierarchy is shown graphically above left, and the relationships between regions is shown above right. For each cluster is given the hypotheses it summarizes, their blackboard-levels, time-regions, and event-classes, and the vehicle-event-classes that the clustered data supports.

Figure 19: Clustering Example 1 Hierarchy Incorporating New Data.

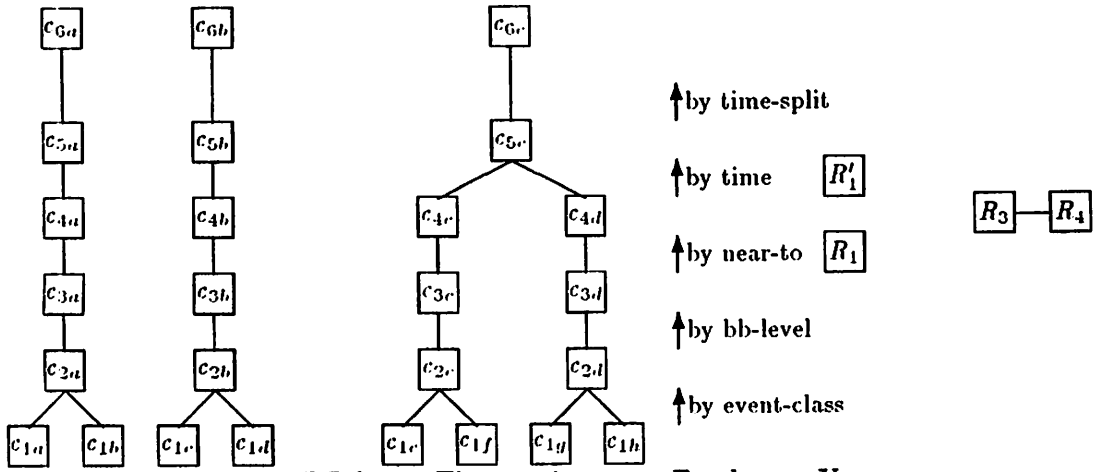
Clustering Example 2

Sometimes, when new hypotheses arrive, they not only extend potential solution tracks but also combine tracks that were previously considered unrelated. For example, Figure 20 shows a situation where, once again, five signal hypotheses for each of for sensed times and regions are clustered. The base-clusters again group hypotheses for the same times, regions, and blackboard-levels based on their vehicle-event-classes. Merge-by-event-class combines the pairs for separate vehicle-event-classes, merge-by-level cannot merge any further, and merge-by-near-to similarly cannot join any clusters together. Merge-by-time combines the clusters for sensed times 3 and 4 since their temporal combination is unambiguous. Merge-by-time-split cannot merge clusters any farther, and the result is three, unrelated top level clusters.

After some time has elapsed, the node receives new information: a hypothesis at the vehicle blackboard-level, where the hypothesis indicates a short track for sensed times 2 and 3. This track fills the gap between the local data for sensed time 1 and the data for sensed time 3, as is shown in Figure 21. The base-clusters are updated by generating a new cluster (since the received hypothesis does not fit any existing base-clusters), which is related to the clusters for sensed times 1, 3, and 4. Merge-by-event-class has no effect on the new cluster, but merge-by-level combines the new cluster with the existing cluster at sensed time 3—they have common time-regions and their relationships to other clusters are consistent. The modified existing cluster is then used by merge-by-near-to which cannot combine it with any clusters. Merge-by-time combines the modified cluster with the existing cluster for sensed time 4, and generates a modified cluster indicating the unambiguous track for sensed times 2 through 4. Finally, merge-by-near-to can combine this modified cluster with the (previously unrelated) clusters at sensed time 1, forming two top level clusters representing potential solution tracks.

The addition of new information therefore may not extend the clustering hierarchy but instead might collapse it. By providing information that relates previously unrelated clusters—by filling in gaps between clusters—the new information can reduce the number of top level clusters. The clustering mechanisms at times need to increase the scope of existing clusters, eliminate other clusters that are obsolete, and propagate the changes to affect relationships between the

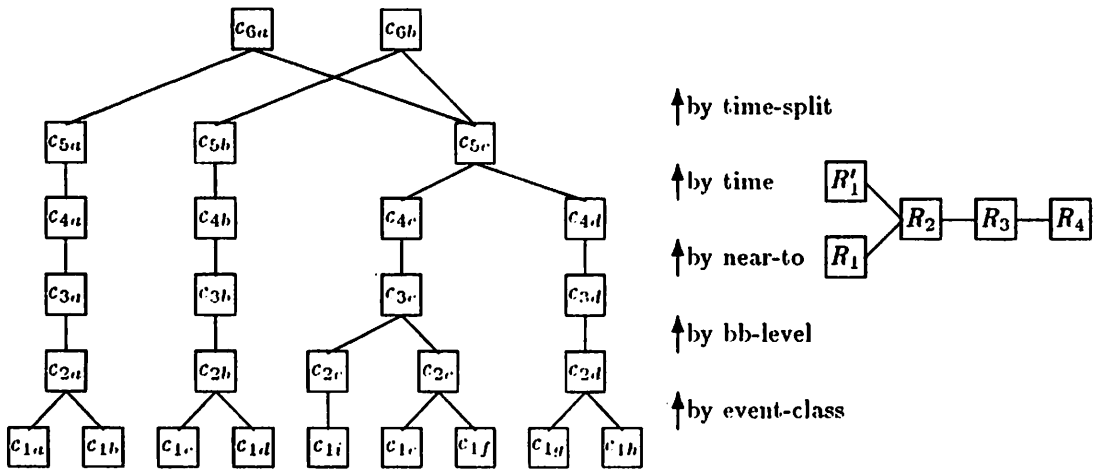
clusters that remain. In turn, as top level clusters are created, modified, and deleted, the set of alternative goals must be updated in the same way: by creating new, modifying existing, and deleting obsolete alternative goals when the top level clusters change.



Cluster	Data-list	BB-levs	Time-regions	Ev-cls	V-e-cs
c_{1a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1)
c_{1b}	(h_5)	(sl)	$((1\ R_1))$	(18)	(2)
c_{1c}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1)
c_{1d}	(h_{10})	(sl)	$((1\ R_1'))$	(18)	(2)
c_{1e}	$(h_{11} - h_{15})$	(sl)	$((3\ R_3))$	(2 6 10 14 18)	(1)
c_{1f}	(h_{15})	(sl)	$((3\ R_3))$	(18)	(2)
c_{1g}	$(h_{16} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1)
c_{1h}	(h_{20})	(sl)	$((4\ R_4))$	(18)	(2)
c_{2a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{2b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{2c}	$(h_{11} - h_{15})$	(sl)	$((3\ R_3))$	(2 6 10 14 18)	(1 2)
c_{2d}	$(h_{18} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1 2)
c_{3a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{3b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{3c}	$(h_{11} - h_{15})$	(sl)	$((3\ R_3))$	(2 6 10 14 18)	(1 2)
c_{3d}	$(h_{18} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1 2)
c_{4a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{4b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{4c}	$(h_{11} - h_{15})$	(sl)	$((3\ R_3))$	(2 6 10 14 18)	(1 2)
c_{4d}	$(h_{16} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1 2)
c_{5a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{5b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{5c}	$(h_{11} - h_{20})$	(sl)	$((3\ R_3)\ (4\ R_4))$	(2 6 10 14 18)	(1 2)
c_{6a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{6b}	$(h_6 - h_{10})$	(sl)	$((1\ R_1'))$	(2 6 10 14 18)	(1 2)
c_{6c}	$(h_{11} - h_{20})$	(sl)	$((3\ R_3)\ (4\ R_4))$	(2 6 10 14 18)	(1 2)

Clustering hierarchy is shown graphically above left, and the relationships between regions is shown above right. For each cluster is given the hypotheses it summarizes, their blackboard-levels, time-regions, and event-classes, and the vehicle-event-classes that the clustered data supports.

Figure 20: Clustering Example 2 Hierarchy.



Cluster	Data-list	BB-levs	Time-regions	Ev-cls	V-e-cs
c_{1a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1)
c_{1b}	(h_5)	(sl)	$((1\ R_1))$	(18)	(2)
c_{1c}	$(h_6 - h_{10})$	(sl)	$((1\ R'_1))$	(2 6 10 14 18)	(1)
c_{1d}	(h_{10})	(sl)	$((1\ R'_1))$	(18)	(2)
c_{1e}	$(h_{11} - h_{15})$	(sl)	$((3\ R_3))$	(2 6 10 14 18)	(1)
c_{1f}	(h_{15})	(sl)	$((3\ R_3))$	(18)	(2)
c_{1g}	$(h_{16} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1)
c_{1h}	(h_{20})	(sl)	$((4\ R_4))$	(18)	(2)
c_{1i}	(h_{21})	(vt)	$((2\ R_2)\ (3\ R_3))$	(1)	(1)
c_{2a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{2b}	$(h_6 - h_{10})$	(sl)	$((1\ R'_1))$	(2 6 10 14 18)	(1 2)
c_{2c}	$(h_{11} - h_{15})$	(sl)	$((3\ R_3))$	(2 6 10 14 18)	(1 2)
c_{2d}	$(h_{18} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1 2)
c_{2e}	(h_{21})	(vt)	$((2\ R_2)\ (3\ R_3))$	(1)	(1)
c_{3a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{3b}	$(h_6 - h_{10})$	(sl)	$((1\ R'_1))$	(2 6 10 14 18)	(1 2)
c_{3c}	$(h_{11} - h_{15}, h_{21})$	$(sl\ vt)$	$((2\ R_2)\ (3\ R_3))$	(1)	(1 2)
c_{3d}	$(h_{18} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1 2)
c_{4a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{4b}	$(h_6 - h_{10})$	(sl)	$((1\ R'_1))$	(2 6 10 14 18)	(1 2)
c_{4c}	$(h_{11} - h_{15}, h_{21})$	$(sl\ vt)$	$((2\ R_2)\ (3\ R_3))$	(1)	(1 2)
c_{4d}	$(h_{16} - h_{20})$	(sl)	$((4\ R_4))$	(2 6 10 14 18)	(1 2)
c_{5a}	$(h_1 - h_5)$	(sl)	$((1\ R_1))$	(2 6 10 14 18)	(1 2)
c_{5b}	$(h_6 - h_{10})$	(sl)	$((1\ R'_1))$	(2 6 10 14 18)	(1 2)
c_{5c}	$(h_{11} - h_{21})$	$(sl\ vt)$	$((2\ R_2)\ (3\ R_3)\ (4\ R_4))$	(1)	(1 2)
c_{6a}	$(h_1 - h_5, h_{11} - h_{21})$	$(sl\ vt)$	$((1\ R_1)\ (2\ R_2)\ (3\ R_3)\ (4\ R_4))$	(1)	(1 2)
c_{6b}	$(h_6 - h_{21})$	$(sl\ vt)$	$((1\ R'_1)\ (2\ R_2)\ (3\ R_3)\ (4\ R_4))$	(1)	(1 2)

Clustering hierarchy is shown graphically above left, and the relationships between regions is shown above right. For each cluster is given the hypotheses it summarizes, their blackboard-levels, time-regions, and event-classes, and the vehicle-event-classes that the clustered data supports.

Figure 21: Clustering Example 2 Hierarchy Incorporating New Data

3.4 The Clustering Mechanisms in General

The clustering hierarchy serves two major purposes. First, it groups together related data to indicate where potential solutions may lie. In a problem solver that begins with only vague goals, the identification of potential solutions can help resolve uncertainty about what partial solutions to develop and how to develop them. The second (and related) purpose of the clustering hierarchy is to identify important subgoals (sub-solutions) that provide the problem solver with even more information about how to develop the complete potential solutions. When forming the overall potential solutions, the clustering mechanisms automatically generate likely sub-solutions (smaller groupings of related data) at lower levels of the clustering hierarchy. Identifying potential solutions and their important subgoals is the first step toward planning effective sequences of actions.

These mechanisms are necessary in the vehicle monitoring domain because a vehicle monitoring problem solver begins with a quantity of data to be processed and little direction other than to develop a "good" interpretation of that data. Not all domains are like this, however. The attributes of the domain that make clustering appropriate are:

- The problem solver begins with a vague goal (such as to develop a "good" interpretation of some data).
- The set of specific goals is so large that enumerating them and trying each in turn is infeasible.
- The set of specific goals is so large that enumerating them and applying some predefined criteria for each to decide whether to pursue it is infeasible.
- The set possible subgoals of each goal is so large that, even if it were given specific goals, the problem solver lacks the information it needs to determine how to satisfy those goals (what subgoals to pursue).
- The problem situation provides sufficient information for the problem solver to identify a small set of likely goals and subgoals by applying approximate domain knowledge.

Domains that do not have these attributes probably will not need the clustering mechanisms described in this chapter. Given specified goals and the ability

to break them down into subgoals, a problem solver could bypass these mechanisms and go straight to the planning mechanisms covered in the next chapter. However, in many real-world domains the problem solver cannot start with a complete list of possible specific goals: in an unpredictable world, the set of situations the problem solver may encounter cannot be predicted, and hence the set of possible specific goals it will need to achieve cannot be enumerated. In these circumstances, the problem solver must have the ability to develop specific goals "on the fly" based on the situation it faces. If the problem solver develops solutions by investigating how different pieces of problem information are related, it can use clustering mechanisms like those described in this chapter.

The most important prerequisite for building similar mechanisms in another domain is identifying the relationships in the problem information to exploit in clustering. In a blackboard-based problem solver, the inherent organization of the blackboard provides many useful relationships: certain attributes of the units placed on the blackboard are used as indices that facilitate storing and retrieving the units. In a well-designed blackboard architecture, these attributes correspond to the important relationships between units.[3] In the DVMT, for example, the hypothesis blackboard has blackboard-level, time, and location indices: these same relationships (blackboard-level, temporal, and spatial) are exactly those emphasized in the clustering hierarchy.

Problem solvers that do not employ a blackboard architecture can also have important relationships between pieces of problem information. For example, a problem solver to schedule tasks (perhaps in a job-shop) can use relationships between tasks: temporal relationships (some tasks may need to be performed before others, some tasks may have to execute simultaneously); spatial relationships (tasks may be spatially distributed); and resource relationships (tasks may compete for processing resources)—just to name a few. To build a high-level view of such a situation, the problem solver can exploit these relationships to cluster information (tasks) into related groups to recognize the important long-term goals of problem solving (such as how a group of tasks all contribute to a final product).

[3]The principal function of a KS in an interpretation task is to combine related hypotheses into larger, more encompassing hypotheses. A well-designed blackboard architecture will store hypotheses so that a KS can efficiently retrieve related hypotheses, so the hypotheses are stored based on their attributes that determine their relationships.

Once the relationships have been identified, functions to generate the set of base-clusters and functions to merge clusters based on certain relationships can be implemented very much like those described above. Although the concepts involved in developing these mechanisms is straightforward, the implementation is not always simple. For example, the merge-by-time-split function described above is complicated by the bookkeeping involved in keeping track of what combinations of time-regions have been developed and the desire to avoid developing redundant combinations (by combining the same clusters in a different order) while not missing any new combinations. Similarly, the functions to update the clustering hierarchy are conceptually straightforward, but keeping track of new and modified clusters and propagating changes to relationships as clusters are formed, modified, and deleted complicate the functions. The description of the functions implemented as part of the DVMT provide the conceptual framework; the rest is a matter of software engineering, understanding the complexity of algorithms, and good programming practice.

Forming a clustering hierarchy requires the clustering functions to be applied in some order. The order used in the DVMT, as described in this chapter, was chosen based on knowledge about how problem solving is done by the system and how the planning mechanisms described in the next chapter are expected to behave. In fact, there may be no order that consistently generates the most useful clustering hierarchy for every problem situation, and our clustering mechanisms were developed to be applicable in any order. In a given application domain, therefore, the decisions about what relationships to emphasize in the clustering hierarchy must be made by the implementer of the problem solver using an understanding of what aspects of a problem situation are most important for making intelligent local decisions. Fortunately, by implementing clustering mechanisms that can be applied in any order (like those described in this chapter), different clustering hierarchies can be easily developed to gain experience and insight into which is the most effective.

The planning mechanisms described in subsequent chapters can use local goals independently of how they are generated by a node or supplied to the node from an outside source. However, forming a clustering hierarchy is an appropriate technique for recognizing goals in domains such as data interpretation, where identifying specific goals and their subgoals cannot otherwise be accomplished.

The mechanisms and data structures described in this chapter provide an example of how clustering hierarchies can be formed in a particular domain, and this example helps suggest a framework for implementing clustering hierarchies in other domains. Finally, by identifying more specific local goals and their subgoals, the problem solver can then proceed to develop plans for determining which of the goals are worth pursuing and how to go about pursuing them.

It is a bad plan that admits of no modification. –Publius

Chapter 4

Planning Local Problem Solving

Once the problem solver has identified a set of *possible* specific local goals, it should plan activities that not only achieve goals, but that also resolve uncertainty about which goals to pursue. Some of these goals might represent alternative and conflicting solutions, and others, on further investigation, might prove to be incompatible with the problem information. By taking actions in pursuit of the goals, the problem solver can work toward acceptable solutions, and by ordering its actions appropriately, the problem solver may be able to more quickly recognize which goals are worthwhile.

An intelligent planner should not simply pursue goals one at a time, but should also reason about how the goals interact in order to develop a problem solving strategy that lets it effectively resolve uncertainty about which goals to pursue at the same time that it pursues them. By working on subgoals shared by several goals, the planner can use subgoal solutions to better identify which of the larger goals are more important. The planner can also expend the minimum effort on the initial exploration of a plan by performing less costly actions first and then using the results of these actions to decide whether continued work on the plan is warranted. When it cannot decide between goals that have no common subgoals, the planner should work on subgoals that are most likely to help distinguish between the overall goals.

This chapter describes such a planner. As part of a DVMT node, the planner uses the information provided by the clustering hierarchy to determine how it should go about building the best solutions in the minimum amount of time. The next section discusses various approaches to planning and characterizes our approach. After the background information, we give an overview of the planning mechanisms. Following that is a section that describes the mechanisms in detail, including examples of planning. The last section generalizes the mechanisms and outlines how they can be applied in other domains.

4.1 Background

If a planner has a complete and certain model of its environment and actions, then a plan should achieve its predicted results. The simplest type of planner assumes that only it changes the environment and that its actions have completely predictable effects on the environment. In essence, such a planner assumes that the actual environment always behaves exactly as its internal model of the environment does. Many early planners such as STRIPS were based on this assumption [Fikes and Nilsson, 1971]. Even with this assumption, planning can be a complex problem because of the potential interactions between goals (or subgoals) to be achieved: one action might undo the effects of an earlier action, causing previously achieved subgoals to be no longer satisfied. Later planning systems, such as INTERPLAN [Tate, 1975] and Waldinger's system [Waldinger, 1977], reordered goals to avoid harmful plan interactions; other planners, like NOAH [Sacerdoti, 1977] and NONLIN [Tate, 1977], only imposed an order on goals when interactions between plans to achieve the goals independently indicated that a particular ordering was necessary. In all of these systems, plans were formed assuming that the environment and actions could be modeled with complete certainty, so that choosing a plan step requires the planner only to explore the ramifications of its possible actions.

When the effects of an action cannot be predicted with certainty, or when events beyond the planner's control can change the environment, then the planner cannot assume that the plan it forms will necessarily be successful. A planner like those discussed above develops a plan based on its internal models, and can monitor the actual execution of the plan to detect deviations between what

was expected and what actually occurred. When the plan deviates, the planner creates a new plan or alters the remainder of the old plan to account for the unanticipated environment. As examples, NOAH [Sacerdoti, 1977] and SIPE [Wilkins, 1984] develop plan representations that facilitate execution monitoring and replanning, and Wesson describes a system that monitors and revises plans [Wesson, 1977]. When possible deviations are predictable, the planner can even plan in advance for different contingencies [Drummond *et al.*, 1987].

Sacerdoti has recognized that more advanced planners should integrate plan generation, execution, and repair [Sacerdoti, 1979]. In unpredictable and uncertain environments, it may be pointless to plan actions for the distant future [Chien and Weissman, 1975; Davis, 1981; Feldman and Sproull, 1977; McCalla *et al.*, 1982]. The planner should adopt a "wait and see" approach in these situations by only planning as far in the future as it can reasonably predict. After executing its partial plan (alternatively, its plan to achieve some set of subgoals), the planner can reconnoiter before developing its next partial plan. By interleaving planning and execution, the planner wastes less time forming plans that are unlikely to come to fruition; by maintaining a view of its overall goals, the planner can choose an effective sequence of partial plans to achieve these goals. Monitoring and repair are still needed since the partial plans may themselves deviate from expectations.

A planner that works in a dynamically changing world must therefore plan both reactively and strategically. *Reactive* planning means that the planner monitors the situation and takes appropriate actions in reaction to the current situation, taking into account both expected and unexpected features of the situation. Reactive planning is thus data-driven. *Strategic* planning means that the planner maps out, at some level, an entire sequence of actions that will take it from its initial situation to some goal situation, and is thus goal-directed. To work effectively, a planner must balance reactive with strategic planning: if overly reactive, the planner will make short-sighted decisions in response to its current situation and may fail to work purposefully toward its long-term goals; but if overly strategic, the planner will lean toward following a planned sequence of actions that may no longer be suitable given its current situation.

The planners developed by AI researchers have fallen somewhere in the range between being purely reactive and being purely strategic. At the purely reactive

end are planners that simply map situations to actions, where planning is reduced to "if the situation is a and the goal is b , then take actions x, y, and z." These planners are arguably not planning at all since they simply follow such rules without the long-term view of how actions will lead to achieving goals that is usually associated with planning. At the purely strategic end are planners that assume that only they can affect their environment and that effects of their actions are completely predictable. Such planners map out detailed actions to achieve all their goals, recognizing before they begin any interactions between goals that force actions to be taken in a particular order. When planning complex sequences of actions, such as molecular genetics experiments for example, a planner may only be capable of developing plans that cannot tolerate any unexpected situations, due to the strong interactions between planned actions. The planner therefore must plan strategically [Stefik, 1981], and need not be reactive (since any need to react means that the only acceptable plan has failed).

The planner described in this chapter falls between the two extremes. The domain is predictable to the extent that the planner can formulate strategic plans at a high level, but unpredictable enough that it should not expend effort planning detailed actions too far in the future.[1] Because there are generally a number of ways of achieving each goal, the planner has the flexibility to react to unexpected situations by modifying its plans to achieve goals in a different way. With this flexibility, the planner can find plans that may pursue several of its goals simultaneously, thereby helping it resolve its uncertainty about which goals are in fact important. In short, planning involves not so much finding the only sequence of actions as finding the most appropriate sequence, where the appropriateness of a plan can change as the situation changes in unexpected ways.

Because the planner must control a problem solver that interacts with some "outside" world (other problem solvers--possibly human--that need solutions within some time), the planner must also have rudimentary capabilities for reasoning about time. In planners that do not associate actions with their durations,

[1]This can be compared with Wesson's approach of forming detailed plans up to some cut-off time in the future, but then pursuing only some of the planned actions before it replans [Wesson, 1977]. By planning far enough into the future, his system tries to recognize future situations that should be resolved by near-term actions. To keep this view of the more long-term future, his system occasionally looks ahead farther into the future. In contrast to this approach, our mechanisms generate a less detailed view of the entire plan, and use this view to plan short-term actions that lead to worthwhile results.

it is the order of plan steps that defines the temporal relationships between actions, and predictions about "when" an action will occur can only be expressed in terms of the actions that must precede it. To better interact with other problem solvers (consumers of a solution), the problem solver needs not only to plan its actions but to predict when those actions will take place so that it can meet deadlines or synchronize in some way with the outside world. Cheeseman, for example, describes a multi-agent planner that uses temporal reasoning to avoid resource conflicts and deadlocks [Cheeseman, 1984]. Other research in planning has incorporated temporal reasoning into planners [Allen, 1984; Lowrance and Friedman, 1977; McDermott, 1982; Vere, 1983]. For example, these planners can develop plans that allow agents to prevent the predicted actions of other agents from occurring [Allen, 1984], or to base planning steps around expected events [Vere, 1983]. The planner described in this chapter uses predictions about the time-costs for achieving different subgoals as a factor in ordering how it will attempt to achieve them. It also uses these predictions to estimate when actions will take place and results will be formed, so that it can respond to externally imposed deadlines (times when results are needed) by recognizing when its plans may exceed deadlines and altering those plans appropriately.

The planner described in this chapter is therefore useful in domains with the following characteristics:

- Unpredictable changes to the problem situation can occur at any time.
- There are several ways to achieve goals (different orderings of subgoals or different sets of subgoals), although some ways may be better (faster, cheaper) than others, so that when plans fail it is possible to form new plans to achieve desired goals.
- The planner is initially uncertain about which goals to pursue and how to pursue them, but by taking appropriate actions it can reduce this uncertainty.
- The possible goals may have related subgoals so the planner could reason about how subgoals should be pursued to further several goals.
- The problem solver must interact with other problem solvers, and so must reason about the time needs of actions to meet deadlines and to coordinate

network problem solving.

4.2 An Overview of the Planning Mechanisms

The planner must intelligently order the problem solving actions. Even with the high-level view provided by the clustering mechanisms, uncertainty remains about whether each long-term goal can actually be achieved, about whether an action that might contribute to achieving a long-term goal will actually do so (since long-term goals are inexact), and about how to most economically form a desired result (since the same result can often be derived in different ways). The planner reduces control uncertainty in two ways. First, it orders the intermediate goals for achieving long-term goals so that the results of working on earlier intermediate goals can diminish the uncertainty about how (and whether) to work on later intermediate goals. Second, the planner forms a detailed sequence of steps to achieve the next intermediate goal: it determines the least costly way to form a result to satisfy the goal. The planner thus sketches out long-term intentions as sequences of intermediate goals, and forms detailed plans about the best way to achieve the next intermediate goal.

A long-term vehicle monitoring goal to generate a track consisting of several time-locations can be reduced to a series of *intermediate-goals* (i-goals) where each i-goal represents a desire to extend the track satisfying the previous i-goal into a new time-location.[2] To determine an order for pursuing the possible i-goals, the planner currently uses three domain-independent heuristics:

Heuristic-1: *Prefer common intermediate-goals.* Some intermediate-goals may be common to several long-term goals. If uncertain about which of these long-term goals to pursue, the planner can postpone its decision by working on common intermediate-goals and then can use these results to better distinguish between the long-term goals. This heuristic is a variation of least-commitment [Stefik, 1981].

Heuristic-2: *Prefer less costly intermediate-goals.* Some intermediate-goals may be more costly to achieve than others. The planner can quickly estimate

[2]In general terms, an i-goal in any interpretation task is to process a new piece of information and to integrate it into the current partial interpretation.

the relative costs of developing results in different areas by comparing their corresponding clusters at a high level of the abstraction hierarchy: the number of event-classes and the spatial range of the data in a cluster roughly indicates how many potentially competing hypotheses might have to be produced. This heuristic causes the planner to develop results more quickly. If these results are creditable they provide predictive information, otherwise the planner can abandon the plan after a minimum of effort.

Heuristic-3: *Prefer discriminative intermediate-goals.* When the planner must discriminate between possible long-term goals, it should prefer to work on intermediate-goals that most effectively indicate the relative promise of each long-term goal. When no common intermediate-goals remain, therefore, this heuristic triggers work in the areas where the long-term goals differ most.

These heuristics are interdependent. For example, common i-goals may also be more costly, as in one of the experiments described in the next chapter. The relative influence of each heuristic can be modified parametrically.

Having identified a sequence of i-goals to achieve one or more long-term goals, the planner can reduce its uncertainty about how to satisfy these i-goals by planning in more detail. If the planner possesses models of the KSs that roughly indicate both the costs of a particular action and the general characteristics of the output of that action (based on the characteristics of the input), then the planner can search for the best of the alternative ways to satisfy an i-goal. We have provided the planner for our vehicle monitoring problem solver with coarse KS models that allow it to make reasonable predictions about short sequences of actions to find the sequences that best achieve i-goals.[3] To reduce the effort spent on planning for an uncertain future, the planner only forms detailed plans for the next i-goal: since the results of earlier i-goals influence decisions about how and whether to pursue subsequent i-goals, the planner avoids expending effort forming detailed plans that may never be used.

Given the abstraction hierarchy in Figure 12, the planner recognizes that achieving each of the four long-term goals (Figure 12d) entails i-goals of tracking the vehicle through these regions. Influenced predominantly by Heuristic-1, the

[3] If the predicted cost of satisfying an i-goal deviates substantially from the crude estimate based on the abstract view, the ordering of the i-goals may need to be revised.

planner decides to initially work toward all four long-term goals at the same time by achieving their common i-goals. A detailed sequence of actions to drive the data in R_3 at the signal blackboard-level to the vehicle blackboard-level is then formulated. The planner creates a plan whose attributes (and their values in this example) are:

- the long-term goals to which the plan contributes (in the example, there are four);
- the predicted, underspecified time-regions of the eventual solution (in the example, the time regions are $(1 \;\; R_1 or R_1')(2 \;\; R_2 or R_2')(3 \;\; R_3)$ ...);
- the predicted vehicle type(s) of the eventual solution (in the example, there is only one type of vehicle considered);
- the order of i-goals (in the example, begin with sensed time 3, then time 4, and then work both backward to earlier times and forward to later times);
- the blackboard-level for tracking, depending on the available knowledge sources (in the example, this is the vehicle blackboard-level);
- a record of past actions, updated as actions are taken (initially empty);
- a sequence of the specific actions to take in the short-term (in the example, the detailed plan is to process data in region R_3);
- a rating based on the number of long-term goals being worked on, the effort already invested in the plan, the ratings of the KSIs corresponding to the detailed short-term actions, the beliefs of the partial solutions previously formed by the plan, and the predicted beliefs of the partial solutions to be formed by the detailed activities.

As each predicted action is consecutively pursued, the record of past actions is updated and the actual results of the action are compared with the general characteristics predicted by the planner. When these agree, the next action in the detailed short-term sequence is performed if there is one; otherwise the planner develops another detailed sequence for the next i-goal. In our example, after forming results in R_3 at a high blackboard-level, the planner forms a sequence

of actions to do the same in R_4. When the actual and predicted results disagree (since the planner's models of the KSs may be inaccurate), the planner must modify the plan by introducing additional actions that can get the plan back on track. If no such actions exist, the plan is aborted and the next most highly-rated plan is pursued. If the planner exhausts its plans before forming a complete solution, the node discontinues planning and pursues any pending highly-rated KSIs in an effort to eventually generate these results using its simpler control mechanisms.

The planner thus generates, monitors, and revises plans, and interleaves these activities with plan execution. In our example, the common i-goals are eventually satisfied and a separate plan must be formed for each of the alternative ways to proceed. After finding a partial track combining data from sensed times 3 and 4, the planner decides to extend this track backward to sensed time 2. The long-term goals indicate that work should be done in either R_2 or R_2'. A plan is generated for each of the two possibilities, and the more highly-rated of these plans is followed. Note, however, that the partial track already developed can provide predictive information that, through goal processing, can increase the rating of work in one of these regions and not the other. In this case, constraints that limit a vehicle's turning rate are used when goal processing (subgoaling) to increase the ratings of KSI's in R_2', thus making the plan to work there next more highly rated.[4]

The planner and goal processing thus work in tandem to improve performance. The goal processing uses a detailed view of local interactions between hypotheses, goals, and KSIs to differentiate between alternative actions. Goal processing can be computationally wasteful, however, when it is invoked based on strictly local criteria. Without the knowledge of long-term reasons for building a hypothesis, the problem solver simply forms goals to extend and refine the hypothesis in all possible ways. These goals are further processed (subgoaled) if they are at certain blackboard-levels, again regardless of any long-term justification for doing so. With its long-term view, the planner can drastically reduce the amount of goal processing. As it pursues, monitors, and repairs plans, the planner identifies areas where goals and subgoals could improve its decisions and selectively invokes

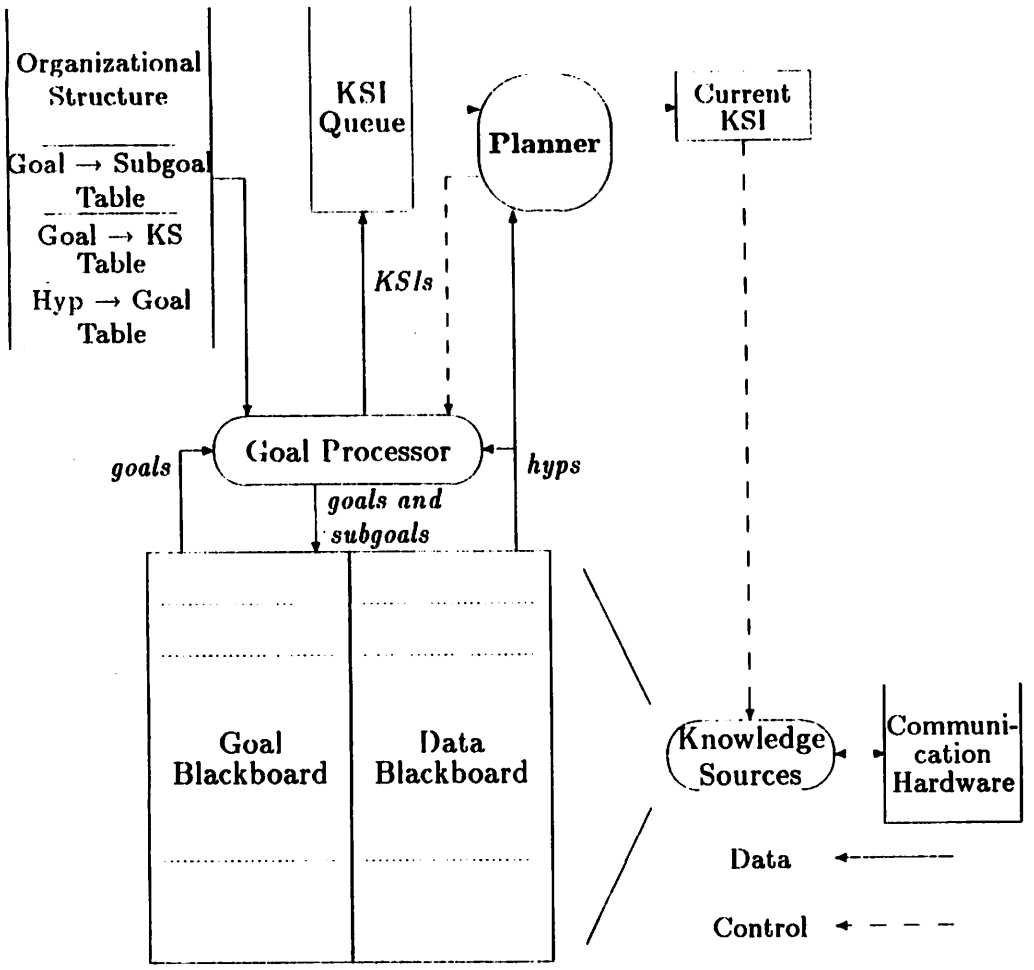
[4] In fact the turn to R_2 exceeds these constraints, as does the turn to R_5', so that the only track that satisfies the constraints is $R_1' R_2' R_3 R_4 R_5 R_6$.

goal processing to form only those goals that it needs.

4.3 The Planning Mechanisms in Detail

The planner is responsible for controlling problem solving, and so must decide what KSI should be invoked at any given time. Without the planner, a node has a KSI dispatcher that simply invokes the most highly-rated KSI at any time (Figure 8). The planner replaces this dispatcher so that a more intelligent choice of KSIs can be made (Figure 22). Moreover, because the planner controls the goal processing, it can trigger the creation of goals (and in turn KSIs) that it believes would be useful. The planner thus uses hypotheses, goals, and KSIs to plan sequences of actions to achieve goals and to invoke specific KSIs that correspond to the planned actions.

The architecture of the planner is sketched in Figure 23. The mechanisms that use hypotheses to build a clustering hierarchy and for using these clusters to recognize alternative-goals were described in the previous chapter. In this section, the other important components of the planner are described in extensive detail. First, the plan data structures are introduced in Section 4.3.1. The plan generator uses information about the alternative-goals and the clustering hierarchy to create plans. The process of creating plans—which involves computing a variety of plan attributes—is described in Section 4.3.2. Because planning and execution are interleaved, the subsequent section, Section 4.3.3, discusses the mechanisms for incrementally updating plans as they are being executed, for monitoring and repairing plans, and for determining what KSI should be invoked for a plan. Next, because a node works in a dynamic world and thus the clustering hierarchy and alternative-goals can change, the other activity of the plan generator—to modify plans based on new problem situations—is discussed in Section 4.3.4. The node might also face deadlines for generating solutions, so the plan executor also has responsibility for updating plans so that they meet deadlines, and the mechanisms for this are described in Section 4.3.5. Section 4.3.6 puts all of these pieces together, outlining the sequence of activities the planner performs when it must determine the KSI that should be invoked next. Finally, Section 4.3.7 presents a detailed example of planning.



The node architecture incorporating the planner is shown. Note that the planner uses hypotheses, KSIs, and goals (through the goal processor) to generate plans and find the current KSI.

Figure 22: The Modified Problem Solving Architecture of a Node.

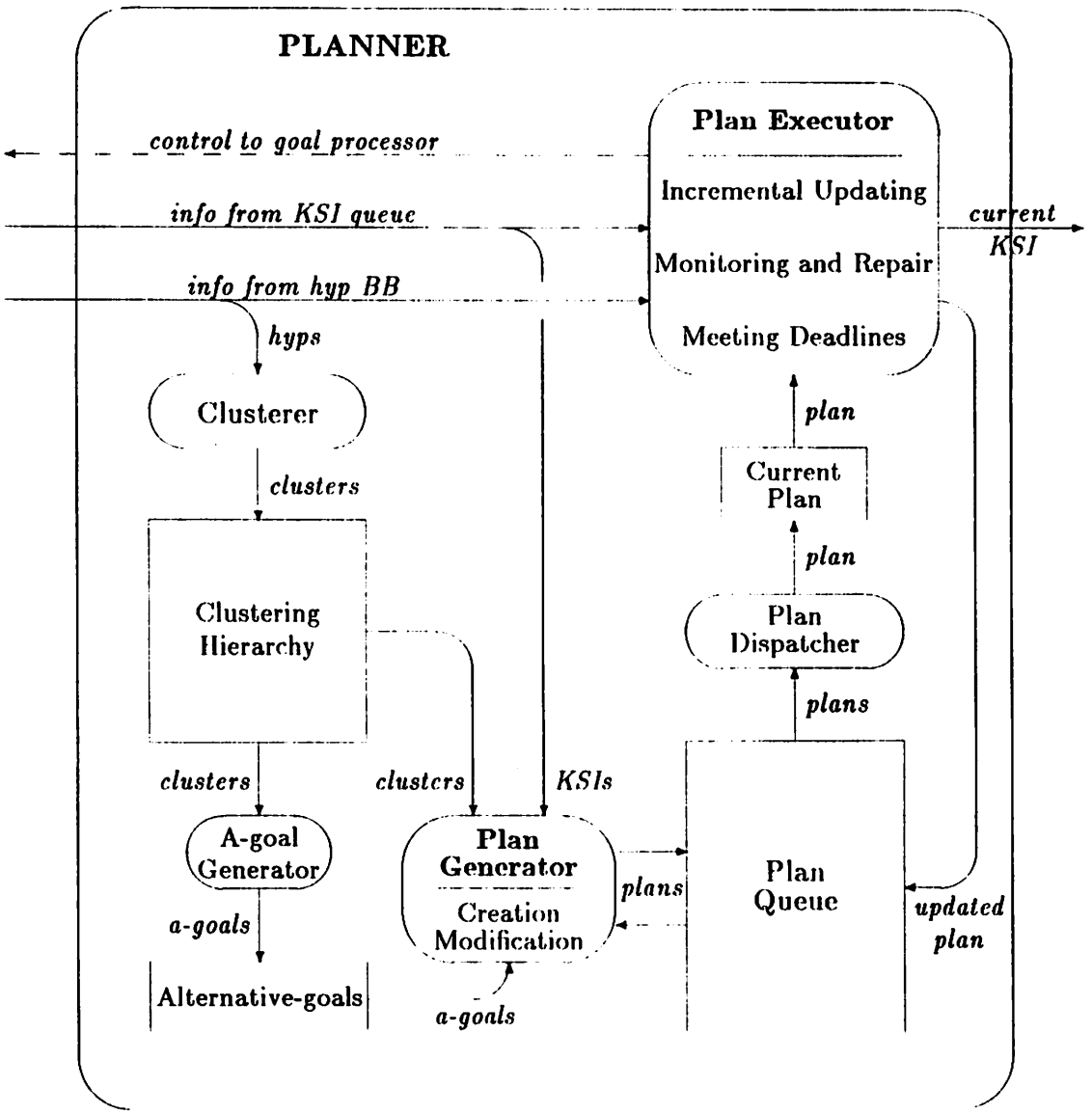


Figure 23: The Architecture of the Planner.

4.3.1 The Plan Data Structures

The planner forms a plan to achieve one or more alternative-goals, and the attributes of a plan are summarized in Figure 24. The plan is given a unique name of the form plan:nn:xxx where nn is the name of the current node (its number) so that different nodes cannot have plans of the same name, and where xxx is a number generated by a counter that the node keeps which is incremented each time a new plan is formed. The simulated node time when the plan is created (or later updated) is also recorded.

A single plan can pursue more than one alternative-goal if it works in areas shared by the goals, although once problem solving in those areas is complete the plan is divided into several plans, one for each subset of the original plan's alternative-goals. The combined time-regions of the alternative-goals summarize the objective track for the plan. For example, the objective track might be ((1 (R_1))(2 (R_2 R_2'))(3 (R_3 R_3'))), where the track indicates that at time 1 the vehicle was in region R_1, at time 2 the vehicle was in region R_2 *or* R_2', and so on. The plan also has an objective of tracking certain types of vehicles, indicated by a list of vehicle-event-classes. Finally, the plan maintains a list of competing-plans—plans that are working on competing alternative-goals.

For each plan, the planner forms strategic information about how the goals will be achieved: the order in which partial solutions will be formed and how they will be integrated into tracks. An *intermediate-goal* (i-goal) represents the desire to extend a previous result (track) into a new area (time-region). The *long-term* plan information sketches out a time-order (the order in which i-goals will be pursued), determines the tracking-level (the level of the blackboard where hypotheses for individual locations or for partial tracks will be integrated into larger partial tracks), records time-predictions (about how long each i-goal is expected to take and how long the plan is expected to take as a whole), and develops an expectation about the general activities performed for each i-goal. The mechanisms for determining the long-term information are described in Section 4.3.2.

A plan also has *short-term* information, containing details about how i-goals have been or will be achieved. The short-term:actions[5] is a list of pending detailed

[5]The data structures often are themselves composed of substructures. We specify a particular field in a substructure by indicating the larger structure(s) as well, using the form larger-structure:smaller-structure. For example, plans have short-term information, and this information

name	A unique name (symbol) pointing at the plan structure	
creation-time	When the plan was created or last updated	
objective	alternative-goals	A list of the alternative-goals to achieve
	track	The expected track for the plan, formed by combining alternative-goals' time-regions
	v-event-classes	The vehicle-event-classes (types of vehicles) being tracked
	competing-plans	A list of plans that have competing alternative-goals as objectives
long-term	time-order	The order for pursuing i-goals, where each i-goal is identified by its data's sensed time
	tracking-levels	The blackboard-level(s) where hypotheses will be integrated into tracks
	time-predictions	The predicted or actual time needs of each i-goal and for the entire plan
	activity-per-time	The basic activities (KSs) executed to achieve an i-goal
short-term	actions	The detailed actions remaining to be done to achieve the current i-goal
	record	The detailed actions already taken for the current i-goal and any past i-goals
	used-clusters	The base-clusters used to achieve each past i-goal and the current i-goal
prediction	result-belief	The predicted belief of the eventual result
	runtimes	The predicted or actual time costs for each i-goal
	prediction-time	When the prediction was formed
rating	rating	The overall rating for the plan
rating-factors	prediction	The predicted belief of the eventual result
	fraction-completed	The fraction of the overall result that has been completed
	alternatives	The number of alternative-goals the plan is concurrently pursuing
	next-result	The predicted belief of the next i-goal's result
	max-ksi	The maximum rating of the KSIs in the short-term actions

Figure 24: The Attributes of a Plan.

actions that will be taken to satisfy the next i-goal. Because unexpected changes to the problem information might alter whether and how the planner should pursue later i-goals, detailed actions are only developed for the next i-goal rather than for all future i-goals. The short-term information also maintains a record of actions that were taken in the past to achieve i-goals so that the planner can refer to past actions and their results. When executing a plan, the node pops the first action off the pending actions list, takes that action, and then pushes the action onto the record list.

An action has the form:

action = (KS-type KSI duration result)

where KS-type is a list of possible KSs that could be invoked, KSI is the specific KSI to be invoked (if it has been instantiated), duration is the expected duration of the action, and result indicates the expected attributes of a hypothesis generated by the action, and has the form:

result = (belief-range time-regions bb-level event-classes volume sat-hyps).

A hypothesis generated by the action is expected to have: a belief falling within the expected range; time-locations, where each time-location falls within one of the expected time-regions; a blackboard-level identical to the expected level; and an event-class that matches one of the expected event-classes. The volume attribute represents an expectation of how many hypotheses are expected to be generated by the action that will match the expected attributes. When the action is taken, a list of hypotheses that satisfy the expectations is inserted into the sat-hyps slot, so that past actions (in the short-term:record) will point to the hypotheses that they generated. The details for generating and manipulating actions and action-results are given in Section 4.3.2.

The short-term information about a plan also indicates the base-clusters "used" for each i-goal. This is described in more detail in the next section, but the basic idea is that the planner should keep track of what clusters' data it has used to attempt to satisfy an i-goal so that if it fails with some data it knows what data it has not tried yet.

is itself a structure containing fields for actions, record, and used-clusters. The short-term:actions are thus the actions field in the plan's short-term information.

A plan has prediction information. It maintains a list of hypotheses at the tracking-level that have some time-locations matching the plan's objective track. The planner uses this list when predicting the belief of the plan's eventual result and the time needed to form it, as will be described in Section 4.3.2. The prediction information contains the simulated time that the prediction was made, the predicted belief of the plan's eventual result, and the predicted time needs for each of the plan's i-goals.

A plan has a rating, which is computed as a combination of several rating-factors. These factors include: a prediction about the belief of the plan's eventual result (which comes from the prediction information); the fraction of the plan already completed (indicating both the effort invested in the plan and the distance to completing the plan); the number of alternative-goals that the plan is concurrently working toward (the length of the objective:alternative-goals list); and information about the more near term qualities of the plan—how highly believed it expects the result satisfying the next i-goal to be and how highly-rated the KSIs it has to work with in the near future are. The rating calculation will be discussed more fully in Section 4.3.2.

4.3.2 Creating Plans

Given a set of alternative-goals to achieve (which were generated by the clustering mechanisms), the planner extracts the most highly-rated alternative-goals in order to generate only the most promising plans.[6] A parameter specifies the allowable range of ratings for the most highly-rated alternative-goals. Next, the planner forms groups of competing alternative-goals. Since competing alternative-goals by definition share common sub-clusters which means that they share i-goals—they are initially grouped together so that the combination can be concurrently pursued by working first on their common i-goals. A plan name and structure are then generated for each group, and the attributes of the plan are computed.

[6] An alternative-goal rating is based on the belief-range of its cluster, so highly-rated alternative-goals are based on strongly believed data.

Computing Plan Objectives

The objective:alternative-goals are simply the group of competing alternative-goals that the plan has been generated to achieve. The objective:vehicle-event-classes are initially the union of the vehicle-event-classes lists of the clusters associated with the alternative-goals.

The objective:track is a combination of the alternative-goals' time-regions. The planner builds the track with the following algorithm:

initialize the track to *nil*

for each alternative-goal's time-regions:

  for each time-region:

    if track has no entry for time

      then create a new entry with time and a list containing the region

      else modify existing entry by checking new region against list of regions for existing entry and if it matches none of them add it to the list

For example, if the time-regions of alternative-goal-1 are $((1\ R_1)(2\ R_2)(3\ R_3))$ and the time-regions for alternative-goal-2 are $((1\ R_1)(2\ R_2')(3\ R_3'))$, then their combined track is $((1\ (R_1))(2\ (R_2\ R_2'))(3\ (R_3\ R_3')))$. This combined track indicates that for each time there are one or more regions that could be developed: at sensed time 1 the vehicle passes through region R_1, at sensed time 2 the vehicle passes through R_2 *or* R_2', and so on. If the planner decides to work on data for sensed time 1 first, then it need not decide yet which of the regions for the other times it will explore—the track indicates several possibilities so that the planner need not commit itself before it has to. Later on when it needs to work at other sensed times, the planner will have to divide the plan into alternative plans to explore the different regions.

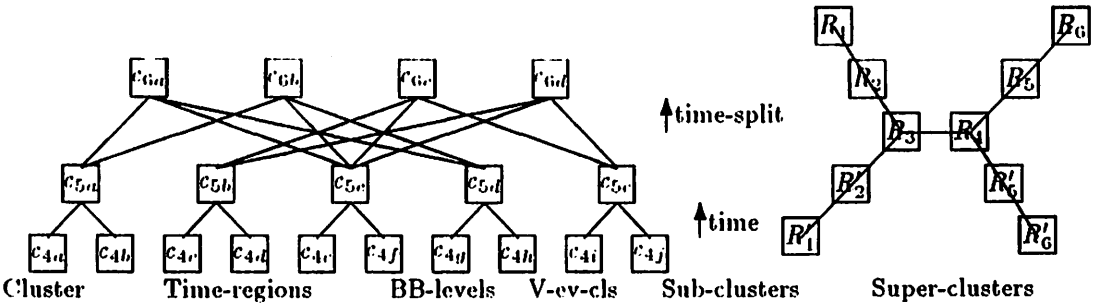
Initially, the plan will have no competing-plans because it is attempting to achieve an entire group of competing alternative-goals. However, as the plan is pursued (and modified when the alternative-goals change), the plan may be divided into different plans to achieve subsets of the original competing alternative-goals. The planner keeps track of what plans are working on competing alternative-goals, and this information is stored in a plan's objective:competing-plans.

Computing Plan Long-term Information

The most important long-term plan information is the order in which the plan will pursue the various i-goals. Since the node is attempting to process all of the appropriate data and integrate it all together into a solution, it can process and integrate the data in any order. However, by processing some data before other data, the node can substantially reduce the time it needs to form a solution: partial solutions provide predictive information that guides processing to extend those partial solutions. Therefore, although the node could pursue the i-goals of a plan in any order and eventually complete the plan (if it can be completed), finding a good ordering of the i-goals can reduce the time needed to achieve a plan or the time needed to recognize that the plan is not worth pursuing.

The order of i-goals is called the *time-order* because each i-goal is represented as a sensed time. By a time-order of (i j k ...) is indicated that the node will form a partial track from data for sensed time i, then attempt to integrate data for sensed time j into that track, then try to integrate data for sensed time k into the track composed by i and j, and so on. To generate a time-order, the planner uses several heuristics about how it is best to go about pursuing a set of long-term goals in domains where the goals may interact and possibly be unachievable. These heuristics were briefly described in Section 4.2, and guide the planner to prefer i-goals that are common to several alternative-goals (to avoid premature commitment to some goals), to prefer i-goals that it expects will require the least time to achieve, and to prefer i-goals that will most quickly generate partial solutions that indicate whether the plan is worth pursuing (compared to other potential plans). These heuristics are incorporated into the i-goal ordering function as follows:

1. First, the function finds common sub-clusters. The top-level clusters that are associated with the alternative-goals may have some common sub-clusters, where the sub-clusters represent data that is unambiguously related by time. For example, the top three levels of a sample clustering hierarchy are shown in Figure 25a. By stepping through the alternative-goals' top-level clusters, the function counts the occurrences of each of their sub-clusters. The result is a list that associates sub-clusters and their occurrence-counts (indicating which sub-clusters are most common),



Cluster	Time-regions	BB-levels	V-ev-cls	Sub-clusters	Super-clusters
c_{4a}	((1 R_1))	(sl)	(1 2)	(···)	(cl_{5a})
c_{4b}	((2 R_2))	(sl)	(1 2)	(···)	(cl_{5a})
c_{4c}	((1 R'_1))	(sl)	(1 2)	(···)	(cl_{5b})
c_{4d}	((2 R'_2))	(sl)	(1 2)	(···)	(cl_{5b})
c_{4e}	((3 R_3))	(sl)	(1 2)	(···)	(cl_{5c})
c_{4f}	((4 R_4))	(sl)	(1 2)	(···)	(cl_{5c})
c_{4g}	((5 R_5))	(sl)	(1 2)	(···)	(cl_{5d})
c_{4h}	((6 R_6))	(sl)	(1 2)	(···)	(cl_{5d})
c_{4i}	((5 R'_5))	(sl)	(1 2)	(···)	(cl_{5e})
c_{4j}	((6 R'_6))	(sl)	(1 2)	(···)	(cl_{5e})
c_{5a}	((1 R_1)(2 R_2))	(sl)	(1 2)	(cl_{4a} cl_{4b})	(cl_{6a} cl_{6b})
c_{5b}	((1 R'_1)(2 R'_2))	(sl)	(1 2)	(cl_{4c} cl_{4d})	(cl_{6c} cl_{6d})
c_{5c}	((3 R_3)(4 R_4))	(sl)	(1 2)	(cl_{4e} cl_{4f})	(cl_{6a} cl_{6b} cl_{6c} cl_{6d})
c_{5d}	((5 R_5)(6 R_6))	(sl)	(1 2)	(cl_{4g} cl_{4h})	(cl_{6a} cl_{6c})
c_{5e}	((5 R'_5)(6 R'_6))	(sl)	(1 2)	(cl_{4i} cl_{4j})	(cl_{6b} cl_{6d})
c_{6a}	((1 R_1)(2 R_2)(3 R_3) (4 R_4)(5 R_5)(6 R_6))	(sl)	(1 2)	(cl_{5a} cl_{5c} cl_{5d})	()
c_{6b}	((1 R_1)(2 R_2)(3 R_3) (4 R_4)(5 R'_5)(6 R'_6))	(sl)	(1 2)	(cl_{5a} cl_{5c} cl_{5e})	()
c_{6c}	((1 R'_1)(2 R'_2)(3 R_3) (4 R_4)(5 R_5)(6 R_6))	(sl)	(1 2)	(cl_{5b} cl_{5c} cl_{5d})	()
c_{6d}	((1 R'_1)(2 R'_2)(3 R_3) (4 R_4)(5 R'_5)(6 R'_6))	(sl)	(1 2)	(cl_{5b} cl_{5c} cl_{5e})	()

(a)

((*cl*$_{5c}$ 4)(*cl*$_{5a}$ 2)(*cl*$_{5b}$ 2)(*cl*$_{5d}$ 2)(*cl*$_{5e}$ 2))

(b)

((*cl*$_{5c}$ 4 1)(*cl*$_{5a}$ 2 2)(*cl*$_{5b}$ 2 2)(*cl*$_{5d}$ 2 2)(*cl*$_{5e}$ 2 2))
where each entry has (cluster occurrence-count cost-value)
and
Vehicle-event-class uncertainty (each cluster) = 2 (v-ev-cls = (1 2))
Spatial uncertainty (each cluster) = 1 (each region covers 1 location)
Distance to tracking-levels (each cluster) = 2 (distance from *sl* to *vl*)

(c)

Figure 25: Ordering I-goals for Sample Situation.

as shown in Figure 25b.

2. Next, the function finds a cost-value for each sub-cluster. The cost-value of a cluster represents a rough indication of the effort needed to achieve an i-goal associated with that cluster, factoring in the relative importance of that i-goal. The cost-value is therefore a function of event-class uncertainty, spatial uncertainty, the distance from the tracking-levels, and the occurrence-count. Event-class uncertainty is measured as the number of vehicle-event-classes associated with the cluster (more vehicle-event-classes generally means more processing is needed to determine exactly which type of vehicle is detected). Spatial uncertainty is measured as the average area (number of locations) covered by the regions in the cluster's time-regions (a larger area indicates that more processing is needed to identify exactly where the vehicle is). Distance from the tracking-levels is measured as the number of blackboard-levels between the highest blackboard-level of the clustered hypotheses and the lowest blackboard-level where data is integrated into tracks (a larger distance indicates that more synthesis is needed before data is integrated into tracks that satisfy i-goals). The cost-values for the example sub-clusters are shown in Figure 25c, where each cost-value is computed as the product of the three factors divided by the occurrence-count:

$$\text{cost-value} = \frac{Uncert_{v-e-c} \times Uncert_{spatial} \times Dist_{levels}}{occurrence\text{-}count}$$

Dividing by the occurrence-count distributes the cost among the relevant alternative-goals, so that more costly sub-clusters can still be preferred if they contribute toward achieving enough alternative-goals. Also note that the cost-value is not normalized since we cannot determine *a priori* a maximum possible cost-value.

3. The function orders the sub-clusters based on their occurrence-counts and cost-values. The relative importance of these two factors is controlled by a user-supplied parameter **cost-heuristic-weight* (*chw), and it is this parameter that determines the balance between the heuristic to prefer common i-goals and the heuristic to prefer less costly i-goals. The overall rating for

each cluster is computed as:

overall-cost = (*chw × cost-value) − [(1 − *chw) × occurrence-count].

The sub-clusters are ordered from lowest to highest overall-cost. Since the cost-value is computed from both the cost factors and the occurrence-count, a *cost-heuristic-weight of 1 is typically used since it balances the heuristics, but experiments with a *cost-heuristic-weight of 0 are also described in Chapter 5.

4. The function then initializes the time-order list to nil. It steps through the sub-clusters and for each identifies its sub-clusters (which do *not* cluster hypotheses by time) and finds the cost-value for those sub-clusters. It then orders them by increasing cost. Next, it forms the time-order by beginning with times corresponding to the least costly i-goals and then adding times for other i-goals, giving preference to times adjacent to those already in the time-order. This promotes extending existing partial solutions rather than generating new partial solutions, and extending partial solutions outward in both directions. The algorithm is:

while some clusters-ordered-by-cost remain, extract the first (lowest cost) cluster and any clusters with the same cost into a *cluster-group*:

for the cluster-group, find the times (i-goals) for those clusters (the times in their time-regions), and remove from these times any that are already in the time-order;

if the time-order is nil (none yet), then set the time-order to the cluster-group's times (arbitrarily ordered from lowest time to highest);

otherwise while some cluster-group times remain:

find the next time to add to the time-order as:

if exactly one cluster-group time is adjacent to a time in the time-order (one more or one less), use that time;

else if more than one cluster-group time is adjacent to a time in the time-order, then choose the one nearest the first time in the time-order (if there is still a tie, choose the one closer to the second time in the time-order);

else if no cluster-group time is adjacent to a time in the time-order, then choose the lowest cluster-group time;
add the chosen cluster-group time to the end of the time-order and delete it from the cluster-group times.

For example, given the information in Figure 25c and using a *cost-heuristic-weight of 1, the planner first adds times 3 and 4 to the time-order. Next, since the remaining clusters have equal cost-values, the planner takes the cluster-group times (1 2 5 6) and adds them to the time-order: first 2 is added (both it and 5 are adjacent to (3 4) but 2 is closer to 3), then 5 is added (both it and 1 are adjacent to (3 4 2) and equally close to 3 but 5 is closer to 4), then 1 is added (both it and 6 are adjacent to (3 4 2 5) but 1 is closer to 3), and finally 6 is added.

The function only considers cost and commonality when building the time-order. The third heuristic—to prefer i-goals that help discriminate between competing plans—is not needed yet since a single plan is created for the entire set of competing alternative-goals. Later on, when the plan is divided into several plans where each has some subset of the original plan's alternative-goals, this heuristic is employed by having the different plans use a common time-order so that they work in competing areas for the same sensed times. For example, if $plan_1$ is pursuing an i-goal to determine whether a partial track passed through region R_i at sensed time i, then it makes sense to have $plan_2$, which wants to extend the same partial track, pursue its i-goal to extend the track into R'_i so that the planner can directly compare the results made by each plan to decide which plan shows more promise. This heuristic is described more fully shortly.

The long-term tracking-levels specify the blackboard-levels where tracks are constructed: where locations are combined into short tracks, where tracks and locations are extended to form longer tracks, and where tracks are merged to form longer tracks. The choice of appropriate tracking-levels depends on several factors:

- the available KSs: the node may only have KSs for forming, extending, and merging tracks at certain levels;
- the branching of the grammar: by constructing tracks at levels where the number of event-classes is least, the number of tracks can be minimized.

- the communication blackboard-levels: building tracks on blackboard-levels below the communication blackboard-levels makes no sense because received tracks cannot be integrated into larger tracks (since the node currently has no way of decomposing or "de-synthesizing" hypotheses). Building tracks on blackboard-levels above the communication blackboard-levels is wasteful since nodes will have to communicate about individual location hypotheses and (probably) redundantly generate tracks. Therefore, it makes the most sense to track at the same blackboard-level as communication.

The planner as currently implemented does not make sophisticated decisions about the tracking-levels. Because the grammars used in the experiments to date have had the smallest number of event-classes at the vehicle blackboard-level, and because communication between nodes is at this blackboard-level as well, the planner computes the tracking-levels as the communication blackboard-levels (when the node is part of a network) or as the *vt* blackboard-level otherwise. In all of the experiments discussed in this document, the tracking-levels is the list (*vt*), although the mechanisms that use the tracking-levels can work with other blackboard-levels as well. Our future research directions include modifying the planner so that it works with the (as yet unimplemented) organizational structuring mechanisms to determine what blackboard-level is best for tracking based on the node's capabilities and the type of information it can communicate.

The long-term:time-predictions and long-term:activity-per-time are computed after the short-term information and the predictions have been computed. In essence, the time-predictions indicate for each i-goal the expected time needed to achieve that i-goal (or if the i-goal has already been achieved, how long it took). These are taken directly from the plan's prediction:runtimes slot once the predictions about the plan have been made. The sum of these times is also computed as the expected overall time needs for the plan. The time-predictions thus have the form:

$$((t_i \;\; t_{i+1} \;\; t_{i+2} \;\; \ldots \;\; t_j) \;\; \sum_{k=i}^{j} t_k)$$

where t_k is the predicted or actual time needed to achieve i-goal k (recall that i-goals are simply represented as a sensed time) and is computed by the prediction functions described below. The activity-per-time attempts to generalize the typical activities (KSs) that are needed to achieve an i-goal, and are extracted from

the short-term actions: each action has a KS-type associated with it, and the mechanisms simply step through the plan's short-term:actions when they have been computed and lists together the sequence of KS-types to be pursued. Since both the long-term:time-predictions and the long-term:activity-per-time information are used primarily for communicating with other nodes about plans, they are not discussed any further in this chapter.

Computing Plan Short-term Information

The long-term information indicates the planner's long-term approach for achieving the plan's objectives. By achieving the i-goals in order, the node will hopefully arrive at an acceptable overall solution in an efficient manner. However, before problem solving can occur, the planner must translate the i-goals into specific actions.

Although the planner could detail the actions for every i-goal when it creates a plan, there is little point in doing so: because unexpected changes might occur to the problem information that can change the goals and how they should be pursued, the planner should only expend effort planning detailed actions for the near future. By adding detailed actions for i-goals only when they are needed, the planner avoids wasting time planning future actions that it may never take. This *incremental* form of planning strikes a balance between strategic and reactive planning: by following the long-term:time-order, the planner works purposefully toward its long-term goals, but by delaying its decision about how it will achieve i-goals, the planner can respond to unexpected situations.

The short-term information is computed whenever a plan needs details for achieving an i-goal, which includes when a plan is created and when it has successfully achieved an i-goal and needs to incrementally add details for the next. The planner begins by comparing the long-term:time-order with the short-term:record of any past actions it took so that it can determine which i-goal to detail plan steps for. For this current i-goal, the planner then computes what detailed actions have to be taken to achieve it. The basic activities performed in achieving i-goals are repetitive: the node should process data to generate the most highly believed hypotheses in the area, and then should integrate these hypotheses with any compatible hypotheses it made previously to generate larger, more encompassing hypotheses. By always following these basic steps for achieving an i-goal,

the planner simplifies its task of finding an appropriate sequence of actions.

The planner therefore faces little uncertainty about which actions should be taken, but it faces considerable uncertainty about what data will be needed by the actions and what results those actions will generate. As it works through the basic sequence of phases for generating a detailed *action-list* for the current i-goal, the planner must use models of the KSs and knowledge about precedence relationships between actions (so that all of the actions to generate results needed by an action are completed before that action is attempted). The basic sequence of phases for detailed planning are:

1. Find the initial data hypotheses that should be processed to achieve the current i-goal;

2. Find any actions to increase the beliefs of those hypotheses;

3. Find any actions to synthesize those hypotheses to a tracking-level;

4. Find any actions to integrate the new hypotheses with any hypotheses generated in the past to create larger hypotheses (longer tracks);

5. If the current i-goal is the last i-goal, then find any actions to synthesize the complete track hypothesis to the solution blackboard-level;

6. Finally, in the course of finding the detailed actions, choices may have been made that committed the plan to pursuing some subset of its objective:alternative-goals or objective:vehicle-event-classes, and the planner must create new plans for any alternative-goals and vehicle-event-classes no longer covered by this plan.

Each of these planning phases is discussed in more detail after a description of the models of KSs that the planner uses.

Models of the KSs. The planner uses models of the KSs to predict the results and durations of its actions.[7] There are two types of models: a model for synthesis KSs and a model for integration (extension) KSs.

[7] These models are similar to Hearsay-II's precondition functions that would compute a *response frame* (a stylized description of the KS's results) based on a *stimulus frame* [Erman *et al.*, 1980].

A synthesis KS finds a group of hypotheses at one blackboard-level and combines them to generate hypotheses at the next higher blackboard-level. To generate output hypotheses, the KS must not only determine what hypotheses should be combined, but also *how well* they combine—indicated by the output hypotheses' beliefs. Given a set of input results from preceding actions, the model for synthesis KSs computes the expected characteristics of the output results:

blackboard-level is the next higher blackboard-level above the input results;

time-regions is the same as the input (time-regions are altered by integration KSs);

event-classes are computed by functions based on the grammar that map the event-classes of the input results to event-classes at the next higher blackboard-level;

volume-of-data equals the number of expected output event-classes (one hypothesis will be generated for each event-class);

belief-range is approximated by the method described below.

Determining the likely output event-classes is fairly inexpensive. The KS-model, like the KS, merely examines the grammar to map event-classes at one blackboard-level to event-classes at the next blackboard-level. However, whereas a KS computes how highly believed the output hypotheses should be based on more complex knowledge about how beliefs should be combined, the model for synthesis KSs uses a much simpler computation that only roughly approximates the KS calculations (the formulas below are based on the typical behavior of the KSs). To compute the belief-range for an output result, the KS-model first uses the grammar to determine how many possible event-classes at the input blackboard-level can lead to any of the output event-classes. It then examines the input results to find out how many actual event-classes it has at the input blackboard-level. It compares the number of possible input event-classes (EC_{pos}) to actual input event-classes (EC_{act}) and computes the likely output belief ($belief_{out}$) as a function of the maximum of the input result belief-ranges ($belief_{in}$):

- If $EC_{pos} = EC_{act}$ then $belief_{out} = belief_{in} + (1 - belief_{in})belief_{in}$.

- If $EC_{pos}/2 \leq EC_{act} < EC_{pos}$ then $belief_{out} \quad belief_{in}$.
- If $0 < EC_{act} < EC_{pos}/2$ then $belief_{out} = belief_{in}/2$.
- If $EC_{act} = 0$ then $belief_{out} = 0$.

Because these simple formulas only return a single predicted output belief, the belief-range for the output result has a minimum and maximum both equal to $belief_{out}$.

For integration KSs, the model must also determine both what results to combine and how highly to believe their combination. The integration KSs generate hypotheses at the same blackboard-level as the input hypotheses, but the scope (time-locations) of the output hypotheses is extended. The characteristics of the output results based on input results are estimated by the model of integration KSs as:

blackboard-level is the same as the blackboard-level of the input results;

time-regions is the combination of the input results' time-regions;

event-classes is the same as the event-classes of the input results;

volume of data is equal to the number of output event-classes (since a hypothesis must be developed for each);

belief-range is the average of the beliefs of the input results, as described below.

Integration KSs, like synthesis KSs, perform complex calculations to determine the belief of output hypotheses. The belief of an output hypothesis is a function of the beliefs of the input hypotheses, where the function considers how closely the combined track fits vehicle movement expectations (reducing belief for tracks that involve unlikely movements) and how much of a contribution each hypothesis makes to the combined track (beliefs of hypotheses that are longer have more effect on the combined hypotheses' beliefs). The model of integration KSs uses a much simpler calculation: it simply computes the output belief-range minimum (maximum) as the average of the input results' belief-range minima (maxima).

The duration of both synthesis and integration KSs is a function of how many hypotheses they produce, and is computed as $duration = ax + b$ where a and b are

constants and x is the number of output hypotheses. Because the models of KSs estimate the volume of hypotheses that they expect to produce, they compute the expected duration using the same function, where x equals the volume of data and where a and b are retrieved from the node's data about its KSs.

The models of KSs therefore estimate the duration and roughly determine the characteristics of the output of a KS based on the characteristics of the input. They determine important attributes of the data—the time-regions and event-classes—using the basic knowledge available to the KSs, but do not expend nearly so much effort on determining the belief of the output hypotheses. As a result, the models of KSs are much less costly than the KSs themselves, and provide a cheap means for developing an expectation about the duration and results of a sequence of actions. However, since the models only crudely estimate the characteristics of the results, and since the results estimated for one action are used as input to estimate the results of the next action, the potential for generating poor estimates increases proportionally with the length of a sequence of actions. By only finding detailed sequences of actions to achieve the next i-goal, the planner avoids forming long sequences that could introduce such errors.

Finding Hypotheses to Develop. Given an i-goal to achieve, the planner must first locate the data that it should use to achieve that i-goal. Once again, the planner calls on the clustering hierarchy. The planner takes the plan's alternative-goals' clusters, the plan's objective:vehicle-event-classes (which can change as problem solving proceeds), and the current i-goal, and works its way down the clustering hierarchy following a path of clusters that are compatible with the i-goal and vehicle-event-classes. Having arrived at a set of base-clusters that satisfy the required characteristics, the planner removes from these any clusters that are already associated with the i-goal in the short-term:used-clusters information. If the planner had previously attempted to achieve the i-goal, for example, and that attempt had failed, we want the planner to start with different hypotheses on its next try. By associating with each i-goal of a plan the base-clusters that have been used to achieve it so far, the planner avoids repeating its mistakes.

The planner then chooses from the remaining base-clusters the cluster with the highest belief-range (the cluster with the most highly believed hypotheses). This cluster specifies the hypotheses that the planner will start with. Note, however,

that it is possible that this base-cluster is *not* common to all of the alternative-goals: as the planner pursues successive i-goals in a plan to achieve multiple alternative-goals, it will eventually attempt an i-goal that is not shared by the alternative-goals. When this happens, the functions to find the best initial data will return a cluster that is not shared by the alternative-goals. The alternative-goals of the plan are changed to the subset that do share the chosen cluster, and the remaining alternative-goals are saved for the last stage of short-term planning, where they are used to trigger the creation of new plans.

Improving the Initial Hypotheses. Once the planner has found the initial hypotheses to work with but before it starts synthesizing them to generate hypotheses that will satisfy the i-goal, the planner should first make sure that these hypotheses cannot be improved. That is, because the beliefs of the hypotheses it generates depend on the beliefs of the hypotheses it starts with, the planner should take any actions that will improve the initial hypotheses' beliefs.

To increase the belief in a hypothesis, the planner should find actions that increase the support for that hypothesis. Support comes from hypotheses at lower blackboard-levels whose attributes (time-regions and event-classes) corroborate the hypothesis, based on the KSs. The planner should determine whether there are any pending KSIs that may synthesize hypotheses at lower blackboard-levels to increase the belief in any of the initial hypotheses. The attributes of the initial hypotheses are summarized in the base-cluster, and the function determines whether actions to improve the hypotheses are available by:

1. Finding the maximum rating (max-rating) of the cluster's past and pending KSIs (from those fields of the cluster);

2. Finding any clusters at lower blackboard-levels (which are pointed at by the cluster's *below-cluster-list*) that have pending (*not* past) synthesis KSIs with higher ratings than max-rating;

3. If such clusters exist, for each of those clusters build actions to run those highly-rated pending KSIs.

Given a KSI that should be invoked, the planner builds an action whose: KS-type is the KS associated with the KSI, KSI is the KSI, and whose duration and

result it predicts by using its models of synthesis KSs. The planner appends these actions to the action-list that it is building.

Finally, because the subsequent planning phases work from the actions and results in the action-list, this phase must generate some action. If it found KSIs to improve the initial hypotheses, then these serve. However, if there were no such KSIs, the planner creates a "dummy" action directly from the initial cluster. This action specifies that an action of KS-type "dummy", with no KSI and a duration of 0, generates a result with exactly the characteristics of the initial cluster (belief-range, time-regions, event-classes, blackboard-levels, a volume equal to the length of the data-list, and sat-hyps is the data-list).

Generating the Hypotheses at a Tracking-level. The planner next finds actions to take the (possibly improved) initial hypotheses and synthesize these up to a tracking-level specified in the plan's long-term:tracking-levels. It first checks the results of the last action taken (possibly a "dummy" action) and determines whether the results are already at a tracking-level. If so, this phase is completed, but otherwise the planner must form actions to synthesize previous results to higher blackboard-levels. These actions are appended onto the action-list. The planner then once again checks the blackboard-level of the last result and determines whether it now has a result at the tracking-level. If not, then the planner forms more actions. This cycle continues until the planner has generated all the actions needed to synthesize the data up to a tracking-level.

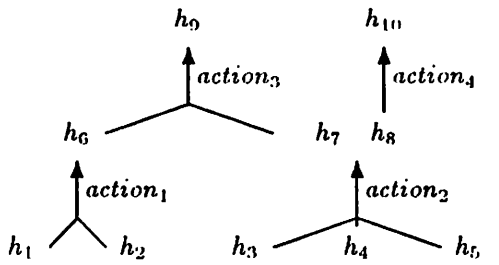
To create actions that synthesize results at one blackboard-level to the next, the planner:

1. Retrieves from the action-list the last action and its result.

2. Uses the blackboard-level of the last result as the input blackboard-level and defines the output blackboard-level as the blackboard-level directly above the input blackboard-level.

3. Steps through the available KSs (and their associated input and output blackboard-levels) and finds a list of suitable synthesis KSs.

4. Finds *all* of the relevant previous actions in the action-list. Because a KS can combine the results of several previous actions, the planner must find



$action_1 = (h_1 + h_2 \rightarrow h_6)$

$action_2 = (h_3 + h_4 + h_5 \rightarrow h_7 + h_8)$

$action_3 = (h_6 + h_7 \rightarrow h_9)$

$action_4 = (h_8 \rightarrow h_{10})$

Hypothesis	BB-level	Event-class	Hypothesis	BB-level	Event-class
h_1	sl	2	h_6	gl	2
h_2	sl	6	h_7	gl	6
h_3	sl	10	h_8	gl	10
h_4	sl	14	h_9	vl	1
h_5	sl	18	h_{10}	vl	2

Figure 26: Actions for Synthesizing VL Hypotheses.

all of these actions. For example, the actions for generating hypotheses at the *vl* blackboard-level based on the grammar of Figure 6 are shown in Figure 26. When planning the action to synthesize a hypothesis with event-class 1 at *vl* ($action_3$), the planner should use the results not only of the last action ($action_2$) but also of the other action that formed hypotheses at *gl* ($action_1$).

5. Uses models of the synthesis KSs to estimate, given some set of input results (the results of the relevant previous actions), the results of the action as well as its predicted duration. Also, if the relevant previous actions have already been taken (for example, "dummy" actions), then the planner examines their sat-hyps to find any KSIs that were formed to work on those hypotheses. If it finds a KSI that may achieve the action's result, then that KSI is used as the action's KSI, otherwise the action's KSI is initially nil.

6. Builds an action and adds it to the action-list.

As it develops these synthesis actions, the planner also keeps track of the vehicle-event-classes being planned for. As described above, the models of the KSs can hypothesize that several actions may lead to the same result and that one action can lead to more than one result. For example, in Figure 26, $action_2$

generates two hypotheses: one with vehicle-event-class 1 and the other with vehicle-event-class 2. Instead of generating for this action a result with both event-classes, the planner uses models of the KSs to identify that, in fact, the result for vehicle-event-class 2 will be substantially inferior to the result for vehicle-event-class 1. The planner can therefore decide that, rather than pursuing both vehicle-event-classes simultaneously, it should concentrate on the better choice.[8] The planner modifies the plan for the better subset of vehicle-event-classes (in this case, vehicle-event-class 1), and subsequently only develops actions that work toward that vehicle-event-class (it only includes actions 1–3 in the detailed plan). The planner will later on create a plan for the cast aside vehicle-event-classes, as is described shortly.

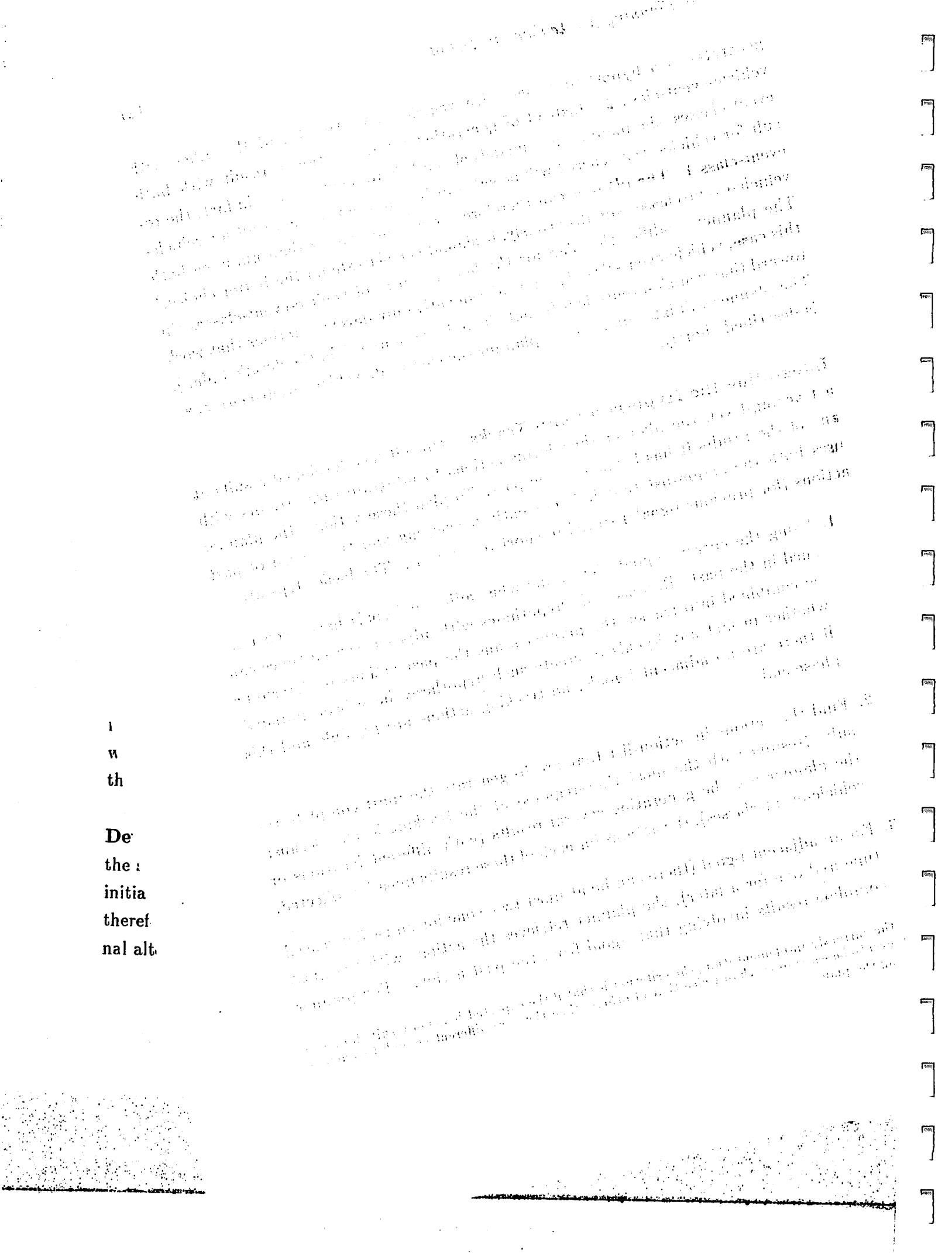
Integrating the Hypotheses into Tracks. Once it has developed results at a tracking-level, the planner then forms actions to integrate these results with any of the results it has formed in the past. To plan these actions, the planner uses both the action-list that it is currently developing and the record of past actions (for previous i-goals) stored in short-term:record. The basic steps are:

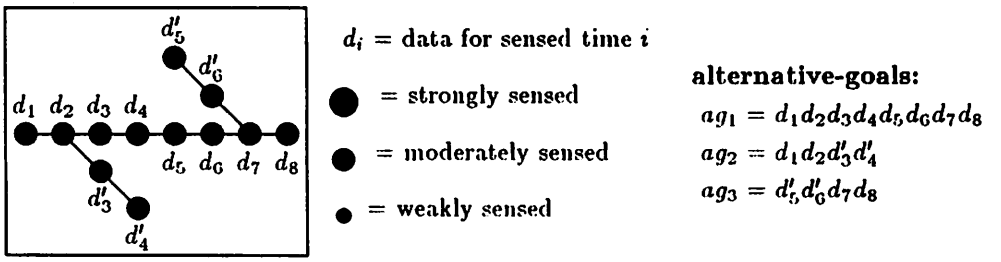
1. Using the current i-goal, determine what adjacent i-goals have been pursued in the past. Because only hypotheses with adjacent sensed times can be combined into tracks, the planner scans the past actions to determine whether in fact any i-goals to create such hypotheses have been pursued. If there are no adjacent i-goals, no tracking actions are possible and this phase ends.

2. Find the actions in action-list that are to generate the most complete results (results with the most time-regions) at the tracking-level. Because the planner may be generating several results (with different locations or vehicle-event-classes), the actions for each of these results must be collected.

3. For an adjacent i-goal (there can be at most two - one for an earlier sensed time and one for a later), the planner retrieves the actions with the most complete results involving that i-goal from the past actions. The planner

[8] In the currently implementation, the criterion is that if the expected belief of results for some vehicle-event-classes is more than twice that of others, then they are different enough to warrant modifying the plan.





Three alternative-goals are generated for the sensed data. Since both ag_2 and ag_3 share data with ag_1, the three alternative-goals are initial grouped together into a competing set. If ag_1 is removed from the set, the remaining alternative-goals no longer belong in the same set since they do not compete with each other.

Figure 27: Competing Alternative-Goals Example

toward a tracking-level, the planner may have decided that some vehicle-event-classes appear more promising (have substantially more support) than others, and therefore may have changed the plan's objective:vehicle-event-classes to this promising subset.

After it has planned the detailed actions for the plan, therefore, the planner must check to see if there are now any alternative-goals or vehicle-event-classes or both that are no longer planned for. The alternative-goals that were dropped from the current plan are grouped together into subsets based on whether they compete with each other. Recall that the original plan was created for set of competing alternative-goals. However, the set was the union of all alternative-goals that compete with at least one other alternative-goal in the set, so they need not all still compete when some are removed from that group. For example, in the situation of Figure 27, alternative-goals ag_1, ag_2, and ag_3 are initially grouped together since ag_1 and ag_2 compete and ag_1 and ag_3 compete, even though ag_2 and ag_3 do not compete. If the plan is changed to pursue ag_1 only, then the other two alternative-goals will not be grouped together into the same new plan.

A new plan is created for each new group of alternative-goals. These plans inherit some of the information from the original plan: the relevant actions in the short-term:record (that worked toward i-goals that these plans did have in common), the short-term:used-clusters for common i-goals pursued in the past, and, most importantly, aspects of the long-term:time-order. The new plans can-

not directly inherit the long-term:time-order since there is no guarantee that the alternative-goals have time-regions for the same sensed times. For example, in Figure 27, ag_1 and ag_2 have different length tracks. If the initial plan to achieve both concurrently had a long-term:time-order of (1 2 3 4 5 6 7 8), then when the plan is attempting to achieve i-goal 3 the plan will divide since the two alternative-goals do not share a common region for sensed time 3. If a new plan is created for ag_2, then it cannot inherent the original plan's time-order since it does not have i-goals for times 5–8.

However, in the discussion about ordering i-goals, the third heuristic is to prefer working on discriminating i-goals. When the planner divides an initial plan into separate plans for ag_1 and ag_2, it should give preference to i-goal 3 for each: since by working on i-goal 3 for each it develops two possible extensions of the track for sensed times 1 and 2, it can compare these two extensions when deciding which plan is better to pursue. The planner can directly compare how well each can extend the short track and can choose to continue with the plan that formed the more highly believed result. Therefore, the third heuristic is implemented not when choosing an initial ordering of i-goals but when dividing a plan into competing plans: the planner forces the competing plans to work on i-goals for their common sensed times in the same order so that the plans are most likely to generate results that can be compared to discriminate between them.

A new plan is also created for the unused vehicle-event-classes. In this case, the plan also inherits the same alternative-goals as the plan that it is being created from. As with dividing a plan for separate alternative-goals, when a plan is divided for different vehicle-event-classes, the planner forces the plans to develop partial solutions that can be better compared.

Short-Term Planning Details: Loose ends. The planner as implemented need not stop after generating the short-term:actions for only the next i-goal. To determine when to stop, the planner consults a user-supplied parameter called the *detailed-planning-window that is a number corresponding to a certain amount of a node's time. If, after it has planned for the next i-goal, the planner totals up the expected durations of all the actions that it has just found (and stored in the short-term:actions) and this total is less than the *detailed-planning-window, then the planner will detail plans for subsequent i-goals until the total predicted duration

of all the actions it finds is equal to or greater than (since it finds actions for complete i-goals) the *detailed-planning window. If set high enough, the planner details every action for the life of the plan: the planner swings toward being a strategic planner (although with the mechanisms to update the plan when new data arrives, the planner will still react to changes by just throwing away some of these details). If set to 0, this parameter turns the planner into a purely reactive planner: the planner does not detail any actions and instead simply reverts to running the most highly-rated KSI (as in the basic control structure outlined in Chapter 2). For the experiments described in this document, the parameter is set at 1, so that the planner finds detailed actions for only the next i-goal.

This description of how the planner forms detailed plans also left out a great many implementation details such as how the various structures are searched and maintained and how the planner keeps track of what results it is or should be working toward and what results it has generated already or should avoid generating. However, the level of detail presented in this section sufficiently describes the major steps in forming the detailed plans. It might also be noted that, when viewed at this level of detail, the mechanisms look to be especially suited to a vehicle monitoring node, or perhaps more generally to interpretation problem solvers, but not particularly general to other types of problem solvers. However, the concepts behind the implementation are general and the mechanisms may be surprisingly useful in a variety of domains. A thorough discussion of the generality of these planning mechanisms is presented in Section 4.4.

Computing Plan Predictions

The planner uses information about the costs and results of the plan's past and current activities, and extrapolates over the future activities to predict the overall costs and results of the plan. By assuming that i-goals that process similar data will be pursued in similar ways, the planner bases predictions for future activities on the most similar past activities. However, the costs of achieving a particular i-goal and the quality of the result depend on the attributes of the data that must be processed (such as how strongly it is sensed, its frequency distribution, and its spatial distribution), and these attributes can vary from one i-goal to another [Pavlin, 1983].

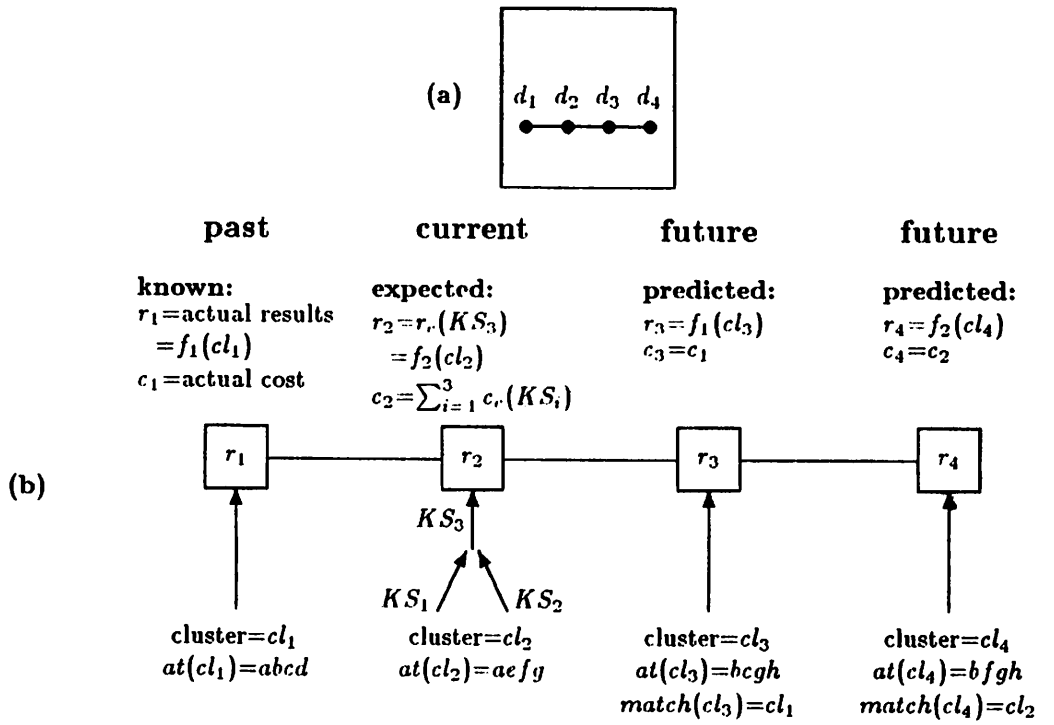
The method of forming predictions for a simple plan to process the data shown

in Figure 28a is illustrated in Figure 28b. The plan will meet its long-term goal of forming a track covering sensed times 1–4 by consecutively achieving its i-goals (generating partial results for each time). The base-cluster for an i-goal summarizes the attributes of its data, so to predict the costs and results for a future i-goal, the planner matches its cluster against the clusters for the current i-goal and any past i-goals, and then extrapolates based on the closest match. Since a plan always has a current i-goal, the mechanisms always have some basis for prediction.

Figure 28 shows the plan in a partially completed state: the i-goal for time 1 was achieved in the past, the i-goal for time 2 is currently being worked on, and the other i-goals are still pending. For a past i-goal (time 1), the results and the cost of forming those results are known. The planner finds the relationship between the attributes of that i-goal's cluster and the results achieved, representing the result as some function of the cluster's attributes. For the current i-goal (time 2), the planner has used models of the KSs to estimate the attributes of the results of a KS action (based on the attributes of its input data) and the costs of the action. The overall costs of the i-goal is the sum of the estimated KS costs, and its result is simply the estimated result of the last KS action. Once again, the relationship between this result and the i-goal's cluster is found.

To predict the results and costs for a future i-goal, the planner first must determine whether that i-goal is shared by any competing plans and, if so, if any of those plans have already achieved the i-goal. If another plan has in fact achieved the i-goal, the planner treats the i-goal as if it is a past i-goal, using the results and costs found by the other plan. If not, then the planner must predict the results and costs for the future i-goal based on past i-goals (of this plan or some other plan) and this plan's current i-goal.

The first step in predicting the results and costs for a future i-goal is to match its cluster against the past and current i-goals' clusters to find the closest match. To simplify Figure 28, we represent the attributes as letters, and the closest match is the past or current cluster with the most letters in common: cl_3 (with attributes bcgh) is closer to cl_1 (abcd) than cl_2 (aefg) while cl_4 (bfgh) is more like cl_2 (aefg). In our implementation, the attributes currently considered are the clusters' blackboard-levels, the volume of data hypotheses they represent, and the time that the data was processed. Given a future i-goal's cluster, the algorithm



Overall predicted result (currently) = $average(r_1, r_2, r_3, r_4)$
Overall predicted cost (currently) = $c_1 + c_2 + c_3 + c_4 = 2c_1 + 2c_2$

Definitions:

r_i	result satisfying intermediate goal i
cl_i	initial clustered hypotheses for intermediate goal i
$at(cl_i)$	attributes of cl_i, here simply represented as letters
$match(cl_i)$	closest matching past or current i-goal cluster for cl_i
c_i	cost (in time) of forming r_i from cl_i
f_i	function relating cl_i with r_i for past and current i-goals
KS_j	KS activity j for achieving current i-goal
$r_e(KS_j)$	expected result generated by KS_j based on KS models
$c_e(KS_j)$	expected cost of performing KS_j based on KS models

The plan data is shown in (a). A partially completed plan to form a track connecting the data is shown in (b): intermediate-goal 1 (i-goal to form r_1) has been achieved, i-goal 2 is being worked on, and i-goals 3 and 4 will be worked on in the future. The results and costs for i-goal 1 are known and the expected results and costs for i-goal 2 are derived using models of KSs. The results and costs for 3 and 4 are found by matching their clusters against those of 1 and 2, and applying knowledge about the closest match. When the plan started, only the expected results for 1 (the current i-goal at that time) were known and predictions for 2–4 were based only on these expectations. The predictions have since improved: for example, better predictions for 4 are made when the more closely matching i-goal 2 becomes current. Finally, KS_1 and KS_2 independently process different subsets of the data in cl_2 to generate supporting hypotheses for KS_3 (hence the representation); the belief in KS_3's result is decreased if only one of the supporting KSs is executed.

Figure 28: Simple Example of Making Predictions.

scans the past and current clusters for those with the closest blackboard-levels (because data at close blackboard-levels undergo similar processing). If only one is found it is returned, but if two or more are equally close then of these the ones with the closest volume of data are found (since more hypotheses may mean substantially more processing is needed). Again, if only one is found then it is returned, but if several are equally close then the one of these whose i-goal was most recently worked on is returned (because more recent activity probably reflects future activity better).

When the closest match is found, the future i-goal's cost is predicted to be the same as the cost for the matching cluster's i-goal— the processing time needed is expected to be the same (Figure 28). To predict the result quality, the planner uses the relationship between the matching cluster and its result (relating the average belief of the cluster's hypotheses with the result's belief) and predicts that the same relationship will hold between the future i-goal's cluster and result (Figure 28). For example, say the matching i-goal's result has a belief twice that of the average of its cluster's hypotheses (this happens, for instance, when the distribution of clustered hypotheses fits the grammar well). Then the belief of the future i-goal's result is predicted to be twice the average belief of its cluster's hypotheses.

The predicted overall cost of a plan is the sum of the i-goals' costs. The predicted overall result of the plan is also a combination of the i-goal's results. When KSs combine data for individual time-locations into a track, the belief of the track hypothesis is a combination of the individual time-location beliefs—usually the average of these beliefs decreased by some penalties for any unlikely vehicle movements (which can only be recognized over a sequence of time-locations). Because the plan's high-level view is too imprecise to recognize unlikely vehicle movements, the prediction mechanisms estimate the overall result's belief simply as the average of the beliefs of the i-goals' actual or predicted results. The predictions thus tend to overestimate rather than underestimate the actual result.

As a plan is pursued, predictions about that plan will generally improve, both because actual costs and results replace those predicted so that overall predictions improve, and because more past experience increases the chances of finding more relevant past i-goals for making predictions about future i-goals. The plan in Figure 28, when it was just starting, could only base predictions on the expected

result and cost of the i-goal for time 1 (which was its current i-goal at that time). Since it now can also base predictions on the i-goal for time 2, it can make better predictions about time 4's i-goal (which matches 2 more closely than 1), and when time 3 becomes the current i-goal, the predictions for 4 will be even better (since it most closely matches 3). By interleaving planning and execution, the predictions about future activities tend to improve as experience with the plan is gained. In fact, by maintaining an extensive database of all its past experience, the planner could gain enough knowledge to make very good predictions for any contingency. However, techniques for efficiently acquiring, saving, and perhaps generalizing this knowledge are machine learning tasks that this research does not address.

Computing Plan Ratings

A plan rating is based on several factors. One factor is the predicted belief of the plan's result (as determined by the prediction mechanisms described above). Because the node is attempting to generate acceptable results as quickly as possible, and because highly believed results are more acceptable than lowly believed results, the planner should rate plans to pursue highly believed results higher.

Another factor that the planner must consider when rating plans is the fraction of the plan that has already been completed: the nearer a plan is to completion, the more emphasis there should be on completing it. By considering the fraction-completed, the planner not only makes problem solving more purposeful (by preferring to continue along a plan that it has been developing), but also can avoid being distracted by a plan that may generate a slightly better result. That is, the node is attempting to generate acceptable solutions as quickly as possible, so by considering both predicted result belief *and* the fraction-completed, the planner can strike a balance between forming the best solution and forming a good solution quickly. The relative importance of these factors is determined parametrically, as discussed below.

Another factor that affects how the planner rates plans is the number of alternative-goals the plans are pursuing. A plan that pursues several alternative-goals is attractive because by following that plan the node works concurrently on several potential solutions and planner can delay committing to a particular potential solution.

Finally, the planner should consider the short-term advantages of a particular plan. The predicted result belief for the plan estimates the plan's long-term utility but the prediction may be errorful; the predicted results of the next i-goal are more likely to be correct, and the ratings of the KSIs to achieve that i-goal also indicate how important it is to pursue the plan, at least for the short-term. Short-term information can be very useful, for example, where choosing between plans that indicate alternative ways of extending the same track. Since goal processing uses knowledge about vehicle movements to form goals indicating where the track can most likely be extended, and since the subgoals of these goals may cause certain KSIs to have their ratings increased, it is useful for the modified ratings of these KSIs to influence the plan rating. In short, the planner should temper its less certain information about the long-term promise of a plan with the more certain information about the plan's near future—both the expected near future results and the ratings of the near future activities.

These separate factors are combined into a single plan rating using the formula:

$$rating = (w_{pr} \times pr) + (w_{fc} \times fc) + (w_{nr} \times (mk \times nr/mb)) + (w_{ag} \times ag).$$

The weights w_{pr}, w_{fc}, w_{nr}, and w_{ag} are normalized (they sum to 1). The predicted result belief, pr, has a value within the DVMT belief/rating range (0 to 10000). The fraction-completed, fc, is also expressed in terms of this range: if the plan is half done, its fraction-complete is 5000. The maximum-ksi rating, mk, falls within this range as does the next i-goal result's expected belief nr. When these are multiplied together, their product is divided by the maximum belief mb of any result (10000) so that their combination has a value within the DVMT belief/rating range. The last factor is the number of alternative-goals, ag, and has a value well under 10000 for the problem situations we examine. Since the weights are normalized, the weighted sum of these factors also falls within the DVMT belief/rating range: a plan can have a maximum rating of 10000. Typical ratings and beliefs in the system are between 1000 and 9000 while a plan usually pursues less than a dozen alternative-goals, so if the factors are weighted equally (at .25) the plan rating is most influenced by the first three factors (predicted result belief, fraction-completed, and expected next result belief). In this case, the number of alternative-goals essentially acts only to break ties between otherwise equally rated plans. In the next chapter (Section 5.1.2), several experiments

are described where the relative weights are changed to examine how they affect problem solving.

Finally, it is important to notice how domain independent the rating factors are. They involve information that is part of most if not all plans: the importance of the plan's result, the investment made in the plan and the additional work needed to complete it, the number of desired results that it is pursuing concurrently, and the attractiveness of the next plan steps. Determining how much influence each of these factors has on the overall plan rating depends on the domain and the desired behavior of the planner.

4.3.3 Plan Execution, Monitoring, and Repair

Because the planner only details short-term actions to achieve one i-goal at a time, planning must be interleaved with execution: after the node has executed the short-term actions for one i-goal, the planner must find a sequence of actions for the next i-goal. Thus, planning must occur at least as frequently as i-goals are satisfied.

In fact, planning should occur more often than that. The planner should monitor plans as each short-term action is performed because those actions may not always achieve their expected results (the KSs might behave differently from their modeled behavior). If a plan fails to generate the results that were expected, the planner should attempt to repair the plan by inserting new actions that may lead to the desired results. In addition, when the short-term actions were detailed, the KSIs for each action may not have been available. If some earlier actions generate hypotheses used by later actions, for example, then the KSIs associated with the later actions will not be available until after the earlier actions are carried out. Finally, new data may have arrived from the node's sensor or from another node, and that data could alter the plan's alternative-goals, its relationships with other plans, and how it can best order its i-goals to achieve its goals. This section describes how the planner monitors, repairs, and updates the short-term actions of the plans as problem solving proceeds. Section 4.3.4 discusses how the objectives, long-term actions, and short-term actions of plans are modified when new data arrives at the node.

Monitoring Plans

Each time a short-term action is carried out by invoking the KSI associated with that action, the planner monitors the action to determine whether the hypotheses generated by the KSI meet predictions for that action (which were computed using the KS-models). The monitor retrieves the KSI's created hypotheses (the invoked KSI's data structure contains a list of the hypotheses that it created), and checks each against their expected attributes:

if the hypothesis' blackboard-level matches the expected result's blackboard-level;

and if the hypothesis' event-class is included in the expected result's event-classes;

and if the hypothesis' time-locations at least cover the expected result's time-regions (described below);

and if the hypothesis' belief is tolerably close to the expected result's belief-range (also described below);

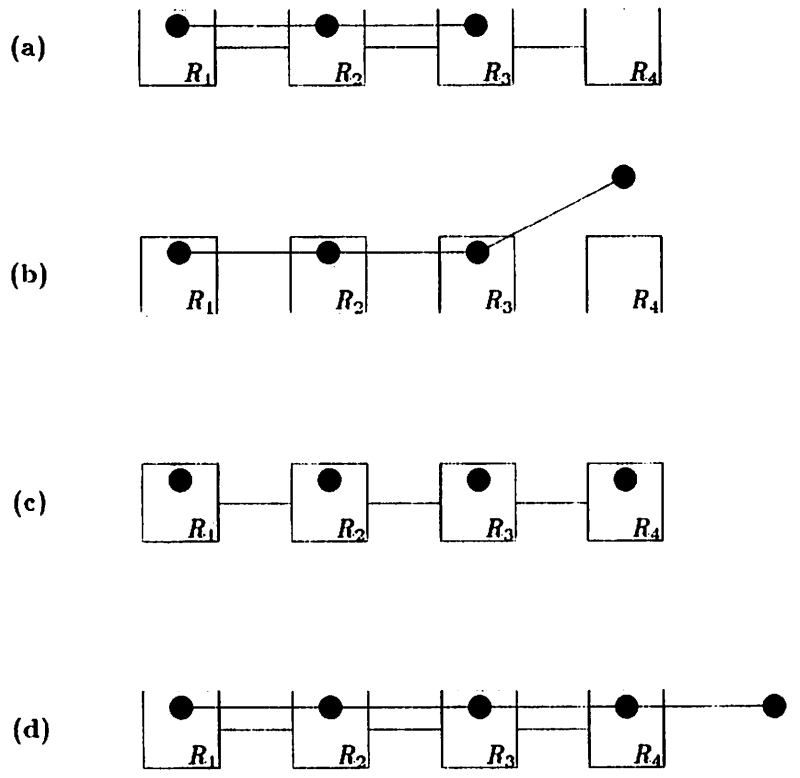
then the hypothesis satisfies the expected results of the action.

The first two criteria are straightforward. The third checks the hypothesis's track against the expected time-regions. The track should have a time-locations that at least cover the result's time-regions: for each of the time-regions, the track should have a time-location with the same time and whose location falls within the desired region. Cases of where the track matches or fails to match the expected result's time-regions are shown in Figure 29. If the track misses a time-region by having no time-location for the same time then the track does not match the time-regions (Figure 29a). Also, if the track misses a time-region by having a time-location whose location does not fall within the desired region, then the track does not satisfy the result (Figure 29b). If each of the tracks time-locations matches one of the expected result's time-regions, then the hypothesis' track satisfies the result's time-regions (Figure 29c). The final case is when the hypothesis' time-locations cover the result's time-regions, but also include time-locations for times not covered by the time-regions (Figure 29d). This can happen when the KSI finds on the blackboard a hypothesis with a longer track (formed

by another plan or received from another node), and combines this hypothesis with the partial result generated by the plan's past actions. The overall track may or may not be acceptable: if the track's additional time-locations do *not* fall within the expected time-regions of *all* of the plan's alternative-goals, then the hypothesis is unacceptable because it does not represent a result that contributes to all of the plan's objectives. If the track's time-locations are consistent with the plan's objectives, however, then the planner adopts the more complete track as satisfying the action. The more complete track can make achieving some future i-goals (for extending the track into times already covered by the more complete track) trivial to achieve and therefore allows the plan to be finished sooner. The planner thus takes advantage of having more complete results being formed unexpectedly, so long as those results are consistent with all of the plan's objectives.

Because the results of one action are used as input to future actions, it is important that the characteristics that affect a result's utility match: if there are no hypotheses with correct blackboard-level, event-class, or time-region attributes, then future actions either cannot be performed or cannot achieve their results. The belief of a hypothesis primarily affects the beliefs of future hypotheses that it contributes to, *not* whether those hypotheses can be formed. Therefore, because the belief has less effect on whether future actions can be taken than the other attributes, and because the models of KSs use very crude approximations when predicting the beliefs of actions, the planner tolerates cases where the beliefs of the actual hypotheses generated by an action deviate from their expected beliefs. A hypothesis' belief matches the expected result's belief-range if its belief is at least some user-specified fraction of the expected minimum belief indicated by the belief-range. In the current implementation, this fraction is one-half: if the belief is at least half of the minimum expected, it is acceptable.

The planner compares all of the hypotheses created by the invoked KSI with the expected results, and builds a list of the hypotheses satisfying the expectations (the sat-hyps). This list replaces the previous value (an empty list) in the expected result's data structure, to record the actual results of the action.



Four cases of attempting to match a hypothesis' time-locations (connected dots) with the expected result's time-regions $R_1R_2R_3R_4$ (connected squares) are shown. In (a), the time-locations do not match because there is no time-location in R_4. In (b), the time-locations do not match because the location for time 4 does not fall within R_4. In (c), the time-locations match the time-regions. In (d), the time-locations extend beyond the time-regions, and whether they match or not depends on whether the additional time-location is consistent with the plan's objectives (its alternative-goals' tracks).

Figure 29: Matching Hypothesis Tracks to Expected Time-Regions.

Repairing Plans

As it monitors a plan's actions, the planner watches for actions that have no satisfying hypotheses. When such an action occurs, it indicates that the plan's short-term actions failed to generate an expected result—that the plan deviated from expectations. The planner attempts to repair the plan by inserting new short-term actions that it expects will generate the missed result. Because plan repair is a complex process that is not the focus of this research, the planner as currently implemented is equipped to only repair plans that fail in certain ways to illustrate how repair can be effected in this framework.

The planner is capable of repairing plans whose actions fail to generate hypotheses with the right time-location characteristics. For example, a KSI might find hypotheses on the blackboard that the plan did not anticipate (hypotheses formed by another plan or received from another node) and thus might generate hypotheses whose tracks are too long or too short or that deviate from the expected time-regions. When this happens, the planner is equipped to generate new actions that may lead to suitable hypotheses whenever possible. The planner begins by finding hypotheses that it may be able to combine into the desired results.[9] It examines the hypotheses produced by the last action (the action that failed), and finds the hypothesis closest to the desired result: a hypothesis that correctly covers as many of the expected time-regions as possible. The planner tries to find the hypothesis with the longest track that is consistent with the desired time-regions but falls short of covering them all. If all of the hypotheses produced extend beyond the desired time-regions, then the planner finds the hypothesis that is consistent with all of the desired time-regions and has the fewest additional time-locations beyond those time-regions.

If the planner cannot find any hypotheses that meet these criteria, then it fails to repair the plan. Having found a hypothesis that extends beyond the desired time-regions, the planner first scans through is past actions to find the hypotheses

[9]Since the blackboard stores every hypothesis generated in the course of problem solving, the planner can retrieve any hypotheses that were formed at any time, even if these hypotheses have been used to generate other hypotheses. When trying to repair a plan, therefore, the planner can plan actions that use some or all of the hypotheses used by earlier plan steps. In other domains, such as assembly tasks, an action that combines two pieces into a subassembly causes those individual pieces to be no longer available: if plan to form a subassembly fails, the planner cannot simply develop a new sequence of assembly steps but must also *disassemble* the faulty subassembly.

that it generated that led to this hypothesis, and returns the hypothesis with the longest track that is consistent with the desired time-regions but falls short of covering them all. With a hypothesis that is shorter than the desired results, the planner then triggers the goal processing to generate a goal to extend the hypothesis into the missing time-regions. In turn, this goal triggers the creation of KSIs to achieve it. If any KSIs are generated, then the planner takes the most highly-rated of these and creates a new detailed action based on the KSI. It inserts this action as the next short-term action, and the repair is completed. Of course, if this action also fails to produce the desired result, the planner once again attempts to repair the plan in the same way. Eventually, either the desired results will be formed or else the planner will run out of KSIs (ways of combining relevant hypotheses) to try, and no new actions are inserted. Whenever a plan's action fails to achieve its result and no new actions can be inserted to repair the plan, the plan is aborted (see Section 4.3.6).

Updating Plans

After the planner monitors the plan (and repairs it if necessary), it must find the next action and KSI. It starts by taking the first action from the list of pending short-term actions and checks whether the KSI for that action is already specified (the KSI was associated with the action by the short-term planning functions or by the repairing functions). If so, then that KSI is triggered as the next local activity and the action is pushed onto the short-term:record list. Otherwise, a KSI for the action was unavailable at the time the short-term:actions were detailed because the results of earlier short-term actions are expected to trigger the formation of this KSI. The planner therefore must cause the goal processing mechanisms to generate the appropriate KSI.

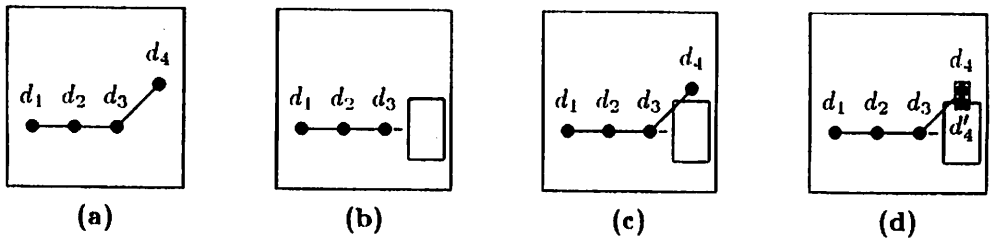
As described in Chapter 2, goals are formed as a result of new hypotheses being posted on the data blackboard. The goal processor generates three goals for each hypothesis: to synthesize it to the next higher blackboard-level, to extend it forward in time, and to extend it backward in time. The planner intervenes in this process by not allowing the goal processor to generate goals for hypotheses unless the planner specifically asks for them. By referring to the future actions that will use a result, the planner can invoke goal processing on a selected hypothesis and can demand only a specific goal be made (synthesis, forward-extension, or

backward-extension) rather than having the goal processor generate all possible goals for the hypothesis. In addition, if the planner is pursuing a plan to extend a track, and that plan is working toward competing alternative-goals that indicate different extensions, then the planner can invoke subgoaling on the extension goal so that it can better differentiate between KSI's that contribute to the different extensions. By being more selective about what goals are formed and when they are subgoaled, the planner reduces goal processing overhead.

Given an action without an associated KSI, the planner first scans the short-term:record to find the actions that lead up to this action: the results that this action is expected to use. The planner then compares the relevant past results to the current action to determine what type of activity is needed: synthesizing the past results, extending them forward, or extending them backward. By passing the satisfying hypotheses (sat-hyps) of the relevant past results and the type of goals desired to the goal processor, the planner triggers new goals to combine the past results in the appropriate way. The new goals in turn trigger the instantiation of new KSIs intended to achieve the goals. The planner then finds the most promising KSI (the most highly-rated KSI whose goals are consistent with the expected result of the next action) and inserts that KSI into the action's KSI slot. Finally, the KSI is triggered as the next KSI (moved to the top of the KSI-queue) and the action is pushed onto the short-term:record list.

When the planner invokes the goal processor to create goals that extend previous results, the goal processor might determine that such goals already exist—they were formed by another plan working on the same data. The planner then must check to see if these goals are already satisfied (hypotheses were formed that satisfy them) and, if so, whether the satisfying hypotheses already achieve the desired results of the next action. When this occurs, the action is trivially achieved and its attributes are updated (its result sat-hyps are set to the satisfying hypotheses, its KSI is set to the KSI that created those hypotheses). It is then pushed onto the short-term:record and the next action in the short-term:actions list is retrieved and pursued.

Finally, it is possible that no KSI can be found for an action because the goal processor may have been unable to instantiate a suitable KSI. This typically occurs when the plan is attempting to generate a result that conflicts with the KSs' knowledge. For example, a plan may be attempting to extend a track into



The plan data is shown in (a), and a partial track for $d_1d_2d_3$ is shown in (b) along with the acceptable range of extensions for that track. The possible range of extensions does not include the data for d_4, shown in (c), and no KSI can currently be formed to extend the partial track. If new data d'_4 arrives, the planner clusters it with d_4 and modifies the plan's expected track. The new data does overlap with the range of extensions (d), so a KSI can be formed and the plan can be pursued.

Figure 30: Example of Suspended Plan and its Continuation.

a new time-region (Figure 30a). Based on the clustering mechanisms' knowledge, this track should be possible because a vehicle in R_3 at time 3 could be in R_4 at time 4. The plan therefore details an action to join the track for times 1–3 with data at 4 into a track for times 1–4. The KSs, on the other hand, have more extensive knowledge about vehicle movements. In particular, given the track for times 1–3 and knowledge about a vehicles maximum turning rate (acceleration), the KSs identify that only data within a specific region can be combined with track 1–3 (Figure 30b). Because the data falls outside of this region, no KSI can be formed to generate the plan's desired result (Figure 30c). The plan is suspended; it is not aborted because new data may possibly arrive that can be used by the KSs to continue the plan. For example, new data that is located close enough to the old data to be clustered together might also overlap with the range of acceptable extensions and therefore allow the plan to continue (Figure 30d).

4.3.4 Modifying Plans

Not only do plans change as they are pursued (as described above), but they also change when relevant new information arrives at the node (from another node or from the sensors) that affects the plans' goals. In Section 3.3.4 the mechanisms for updating the clustering hierarchy based on new information were described.[10]

[10]The clustering mechanisms could also be triggered when a local KS generates completely unexpected hypotheses which would not fit into any existing plans. Extending the mechanisms

When the clustering hierarchy changes, the top-level clusters may change: some existing clusters may be extended or joined together, and new top-level clusters may be added for new information. Because each top-level cluster corresponds to an alternative-goal, the planner must modify the set of alternative-goals as well, changing any existing alternative-goals whose clusters have changed and creating new alternative-goals for new clusters.

The planner then plans for any new or modified alternative-goals. It divides the new alternative-goals into groups of competing goals and creates a new plan for each group (see Section 4.3.2), recording the names of these new-plans. It then steps through the modified alternative-goals and, for each, determines what existing plans were pursuing that alternative-goal. The names of these modified plans are also recorded.

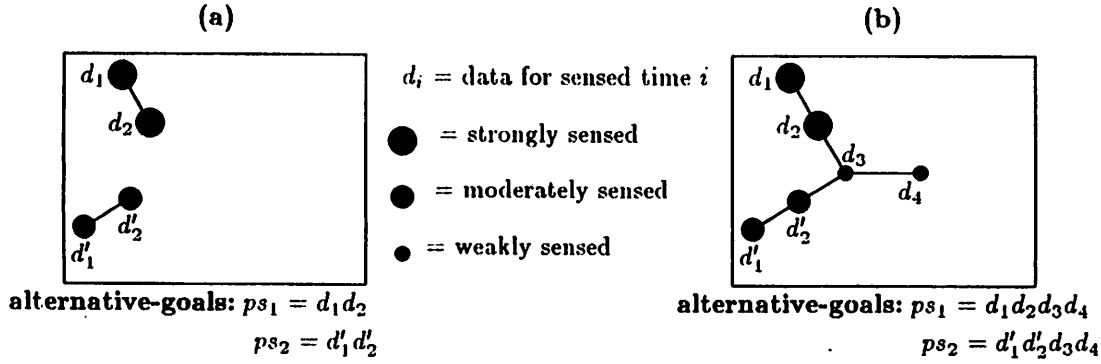
The planner must then determine whether any of the new and modified plans should be merged: separate plans that were generated for previously non-competing alternative-goals may need to be merged if the new data has caused their alternative-goals to now become competing. For example, given the initial situation in Figure 31a, the planner identifies two alternative-goals: to form the track d_1d_2 and the track $d'_1d'_2$. Because these alternative-goals have no common data, they are non-competing and a separate plan is made for each. Some time later, data arrives for sensed times 3 and 4 -- d_3 and d_4 --that is compatible with both of the initial tracks (Figure 31b). The alternative-goals are modified to $d_1d_2d_3d_4$ and $d'_1d'_2d_3d_4$, and both of the initial plans therefore have modified alternative-goals. Whereas the plans were initially non-competing, now their alternative-goals overlap in d_3d_4. The planner could pursue the now competing plans separately, but if it merges them into a single plan, it can reason about how both alternative-goals can be pursued by working on their common data.[11]

When deciding whether to merge two plans, the planner considers their ob-

---

to cluster such hypotheses would be straightforward, but is currently unnecessary since the KSs currently used in the DVMT cannot generate such hypotheses.

[11] Besides giving plans formed by dividing a single plan similar long-term:time-orders, the planner does not reason about how pursuing one plan will affect other local plans. The planner is implemented to pursue the most highly-rated plan without regard to other plans, otherwise the complexity of the planning task (number of different combinations of plans and ways of achieving them) would increase, along with the planning overhead. To have the planner reason about achieving several alternative-goals simultaneously, the alternative-goals must be part of the same plan, so related plans must be merged.



Initially, the planner has for data points: d_1, d'_1, d_2, and d'_2, shown in (a). After some time, more data (d_3 and d_4) arrives and can be combined with the previous data into modified alternative-goals, shown in (b). The separate plans based on the initial (non-competing) alternative-goals in (a) should be merged to pursue the competing alternative-goals in (b) by working on their common data.

Figure 31: An Example of Modified Alternative-Goals.

jectives and past activities. They must have competing alternative-goals, the same objective:vehicle-event-classes, and any past actions taken by either plan should not interfere with their combined actions. This last requirement needs some explanation. When two plans are merged, their short-term:record information is merged—the combined plan should maintain a record of the past actions and results of the plans. However, if either of the plans has already generated a combinable result in an area that is not shared by the alternative-goals but is adjacent to a shared area, then the plans should not be combined because this result will interfere with the results generated in the common area. For example, if the plan to form track $d_1 d_2$ in Figure 31a has formed that track before the new data arrives, then consider what would happen if the plans were merged because of the common data in $d_3 d_4$. After the merged plan raises the data in d_3 to a tracking-level, the planner recognizes that, in the past, a result was formed for sensed time 2, so d_3 should be joined to that result, forming $d_1 d_2 d_3$. This result is *not* appropriate because it is not shared by all of the objective alternative-goals. As a result, only plans that have not yet formed results adjacent to shared areas can be merged.

When two plans are merged, their combined attributes are assigned to one of them and the other is deleted from the set of plans. Their alternative-goals

are combined, and so are their short-term:records (their lists of past actions are combined into a single list) and short-term:used-clusters. When the merged plan is later divided up again for the different alternative-goals, the planner extracts out and gives the divided plans only the short-term information appropriate—it gives each only the short-term information corresponding its i-goals. The short-term:actions for the merged plan is reset to nil, since the change in alternative-goals (and the addition of common areas) may cause the ordering of i-goals to change.

The final step in modifying the plans is to update the attributes of any merged plans and modified plans, since in both cases their alternative-goals have changed. The objective:track is recomputed for the new combination of alternative-goals, as is the long-term:time-order. New short-term actions are found for the plan, and new predictions are made. Finally, the rating factors and the rating are recalculated.

4.3.5 Deadlines and Termination

A problem solver is seldom given exactly as much time as it needs to solve a problem, especially when the costs of problem solving are initially uncertain. When given extra time, the problem solver should make intelligent decisions about when to terminate problem solving: it should sufficiently explore the possible solutions to be reasonably confident that it has found the best one, but should avoid wasting time generating solutions that could not possibly be of use [Hayes-Roth and Lesser, 1977; Woods, 1977]. When given less time than it needs, it should revise its plans to generate some inferior but still acceptable solution if it can.

When a node has found a solution with time to spare and needs to decide whether to terminate problem solving, the planner compares the solution with the predicted results of competing plans. Since the node is attempting to form the best solution, any plans that predict better results should be pursued. Any plans that predict much worse results should not. Plans that predict somewhat worse results might still be worth pursuing, since the predictions may underestimate the actual results, especially when the plan's i-goals work with very dissimilar data. Our mechanisms allow the user to specify how close a competing plan's predicted results must be to the best solution already found for the plan to warrant further

work. The planner can be conservative if the window of acceptable plans is so large that it is unlikely to miss any good solutions, or if the window is very small the node can more quickly propose a solution at the risk of missing a better solution had it kept looking. Unlike our earlier termination technique of using statically defined criteria such as generating a hypothesis with certain predefined characteristics, the new mechanisms allow the node to dynamically compare the solutions it has developed with those it predicts it could develop and decide when it has found the best solutions.

Instead of having extra time and needing to decide whether it is worthwhile exploring alternative solutions, the node might have too little time to generate even a single solution. The node might face tight deadlines. Without the ability to predict how long a plan will take, the node would simply pursue a plan and hope to finish in time. Our new mechanisms, however, allow the node to roughly predict how long a plan will take. Before it gets far with a plan, the node can recognize that the predicted time needed by the plan will probably exceed the time available, and can do something about it. The planner can respond to this situation in any of a number of ways [Lesser and Pavlin, 1987]: it can reduce the needs of the plan by making the plan's long-term goal less constrained (for example, instead of forming the entire track it may focus on only forming a shorter track for the most recently sensed data); it may replace costly activities with less costly activities which may produce inferior but acceptable results; or it may choose another, cheaper plan that can be finished.

Our current implementation has two simple mechanisms for revising a plan to meet deadlines. The first mechanism is to reduce the scope of the plan's long-term goals by ignoring data for some of the sensed times. Because a vehicle's more recent movements are usually most important (for recognizing pending collisions with other vehicles, for example), our mechanism simply drops i-goals for earlier sensed data until it predicts that the plan to process the remaining data will meet the deadlines. The other mechanism is to drop plan steps that corroborate hypotheses and increase the belief in the solution but do not affect the scope of the solution. For the current i-goal in Figure 28, for example, KS_1 and KS_2 both supply supporting hypotheses for KS_3. The results formed by KS_3 will be better (more highly believed) if both supporting KSs are executed, but an inferior result can still be made if one of the two is dropped. In a given situation where a plan

will exceed deadlines, preferences about which of these mechanisms to try first depends on whether the agent needing the solution (currently by the user) tells the node that belief is more important than scope or *vice versa*.[12]

4.3.6 Putting It All Together: Planning for Problem Solving

The planner is invoked as part of a node's control activities after each KSI execution:

1. it forms or updates the clustering hierarchy with any new data, and creates and modifies plans given a new/updated clustering hierarchy;
2. it monitors the plan that was last pursued, determining whether the KSI invoked achieved the desired result, and if not, it repairs the plan;
3. it determines whether the plan last pursued has exhausted its set of detailed short-term:actions and needs detailed planning for its next i-goal;
4. it finds the best active plan to pursue next;
5. and it finds the appropriate KSI for the plan being pursued and puts that KSI at the top of the KSI-queue.

If new data has arrived at the node (received from its sensors or from another node), then the planner updates the clustering hierarchy if one already exists, and otherwise creates the clustering hierarchy. The planner then uses any new and modified alternative-goals to create new plans and to modify existing plans (if any). These new and modified plans are inserted into the planner's queue of plans, where the queue is prioritized based on the ratings of the plans.

The planner determines if the previous KSI invoked by the node was triggered by the planner, as opposed to being triggered by the arrival of new sensor data. If so, then the planner must compare the results of that KSI with the expected results. In this way, the planner monitors the plan, and repairs it if necessary (and

[12]Since the KSs compute belief as essentially the average of the individual pieces of a track, belief does not necessarily increase with increasing scope—belief is more a function of the amount of processing done on each piece of a solution than of how many pieces have been processed. Thus, our two mechanisms can treat scope and belief as being relatively independent attributes of a solution.

possible). Furthermore, the planner checks to see if the plan's short-term:actions have all been achieved. If the previous KSI invoked by the node corresponds to the last action of a plan's short-term:actions, then the planner must plan the short-term actions that will achieve the plan's next i-goal.

The planner next must determine what plan should now be pursued. At any given time, a plan can be in one of four possible states:

active: the last action of the plan was successful and the next action has a KSI associated with it (so it can be pursued);

complete: the last action of the plan was successful and the planner does not have any more actions (it has achieved all of the i-goals);

suspended: the last action of the plan was successful but the planner cannot currently find a KSI for the next action;

aborted: the last action of the plan was unsuccessful and the planner cannot find another action to repair the plan.

To find the plan that it should pursue, the planner examines each plan in the plan queue until it finds an active plan. Since the plan queue is ordered so that more highly-rated plans are earlier in the queue, the first active plan is also the most highly-rated active plan. The planner retrieves the KSI associated with the best plan's next action and inserts that KSI into the top of the KSI-queue.

If for some reason the planner fails to find a KSI (there are no active plans), then the node reverts back to the simpler control scheme of executing the most highly-rated KSIs. Since the planner might have suppressed the goal processor from forming goals and subgoals that were not needed by the plans, and since some of these goals and subgoals might be needed by the simpler control scheme to form and rate useful KSIs, a node without active plans might retrigger goal processing on some of its hypotheses. Even when no plans are active, however, the planner is still invoked between each KSI execution so that it can modify its plans based on any new sensor data or data received from another node. If it develops active plans, the planner once again controls node problem solving.

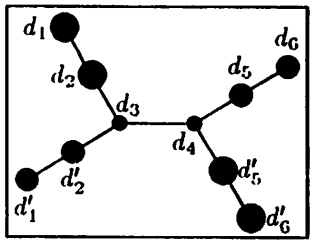
4.3.7 An Example of Local Planning

As an example of how the planning mechanisms work, we describe in more detail the example used in the overview (Section 4.2) and in the more complete description of the mechanisms above: the problem data is shown in Figure 32. Assume that the data for the six sensed times arrives over the first six simulated time intervals. Problem solving thus begins at time 7.

The planner begins by forming the clustering hierarchy. The top three levels of this particular clustering hierarchy were discussed in the previous chapter (Figure 25). When it has completed the clustering hierarchy, the planner then extracts the four top-level clusters and generates alternative-goals. All four alternative-goals are competing since they share common data in their clusters. The plan generator then groups the alternative-goals into groups of competing alternative-goals– in this case, into a single group.

The plan generator creates a plan for this group of alternative-goals. It computes the plan's objectives by combining the attributes of the alternative-goals. It then computes the long-term information for the plan. The long-term tracking-levels is set to (vt) because that is the lowest blackboard-level at which the node has KSs that can combine hypotheses into tracks. To order the plan's i-goals, the planner computes the overall-cost for the various sub-clusters (Figure 25) and identifies that the most cost effective area of the plan is the common area d_3d_4 (the *cost-heuristic-weight is 1 in these calculations). The planner starts the time-order with these i-goals: having no reason to prefer one to the other, it arbitrarily orders these i-goals as (3 4)—to work in d_3 and then d_4. By comparing the costs of the remaining clusters, it groups them together and orders the i-goals as (3 4 2 5 1 6).

Before it can determine the other long-term attributes of the plan, the planner must plan short-term activities and make predictions about the plan. The short-term actions are found for the first i-goal (to process data in d_3). Initially, the planner attempts to find actions that will develop hypotheses at the tracking-level at both vehicle-event-classes (1 and 2). It searches down the clustering hierarchy for an appropriate base-clusters and forms actions to synthesize their data to the proper blackboard-level. Based on its models of KSs, however, the planner identifies that the hypotheses formed for vehicle-event-class 1 are likely to have a much higher belief. Therefore, the objectives of this plan are changed



d_i = data for sensed time i

= strongly sensed

= moderately sensed

= weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5 d_6$

$ag_2 = d_1 d_2 d_3 d_4 d'_5 d'_6$

$ag_3 = d'_1 d'_2 d_3 d_4 d_5 d_6$

$ag_4 = d'_1 d'_2 d_3 d_4 d'_5 d'_6$

acceptable solutions: ag_3

Time	Plan	Rating	A-goals	V-e-cs	Current Result	Pend I-goals
7	$plan_1$	2719	ag_1, ag_2, ag_3, ag_4	(1)	none	(3 4 2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
14	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4213	ag_3, ag_4	(1)	$d_3 d_4$	(2 5 1 6)
18	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4316	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	4344	ag_3	(1)	$d'_2 d_3 d_4$	(5 1 6)
22	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4316	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	4240	ag_3	(1)	$d'_2 d_3 d_4 d_5$	(1 6)
26	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	0	ag_4	(1)	$d_4 d'_5$	(1 6)
	$plan_4$	4240	ag_3	(1)	$d'_2 d_3 d_4 d_5$	(1 6)
34	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	0	ag_4	(1)	$d_4 d'_5$	(1 6)
	$plan_4$	5906	ag_3	(1)	$d'_1 d'_2 d_3 d_4 d_5 d_6$	()

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node. The plans at various times are shown to indicate the evolution of the plans as problem solving progresses. For each plan is given its rating (out of 10000), the alternative-goals that it is expected to work toward, the Vehicle-event-classes of the vehicles it is attempting to track, the result it has generated to this point, and the pending i-goals in the order that they will be attempted.

Figure 32: A Plan Summary for an Example Environment.

to pursue only that type of vehicle, the short-term actions are developed for that type of vehicle (and the short-term used-clusters updated), and the remaining vehicle-event-class is recorded so that once this plan has been completely formed, a second plan for the other vehicle-event-class can be produced.

The planner then forms predictions for the plan. Until it has pursued its plans further, it bases these predictions on the expected results of the short-term actions (which were predicted by the models of KSs). It compares the expected results of i-goal 3 with that i-goal's base-clusters, and predicts the results of the future i-goals by identifying their base-clusters and expecting the same relationship to hold between their results and base-clusters as between i-goal 3's results and its base-clusters. The planner also predicts that the time costs for future i-goals will be comparable to the expected time needs (again predicted by the KS models) of i-goal 3.

The remaining long-term information is determined: the time-predictions are based on the predictions just found, and the activity-per-time is found by scanning through the short-term actions and listing those actions' KS-types. Finally, the plan rating factors are found: predicted result belief from the predictions, predicted belief and KSI ratings for the next i-goal are found from the short-term actions, fraction-completed is found from the (initially empty) short-term record, and the number of alternative-goals is found from the plan's objectives.

The planner inserts this plan on the plan queue, where it also puts the more lowly-rated plan that it creates for the other vehicle-event-class. The plan dispatcher chooses the most highly-rated plan as the current plan, which gets passed to the plan executor. The plan executor first determines whether there are any short-term actions, and if not it sends the plan to the plan generator to form the detailed short-term actions for the next i-goal. If the next i-goal is not common to all of the plan's alternative-goals, then the plan generator will divide the plan: it generates actions that only pursue a subset of these goals, and builds new plans to achieve any remaining alternative-goals. These are placed in the plan queue and the plan dispatcher once again sends the most highly-rated plan to the plan executor.

If the plan has short-term actions, the plan executor then determines whether the predicted time needs of the plan exceed any deadlines, and if so it modifies the plan appropriately (reducing its goals, removing some of its short-term actions).

If it changes the plan, the plan executor rerates the plan, reinserts it into the plan queue, and triggers the plan dispatcher to find the current plan once again. When it has a plan that needs no modification, the plan executor takes the next short-term action and determines whether it has a KSI associated with it. If not, the plan executor examines the plan's short-term actions to find relevant hypotheses formed in the past and triggers the goal processor to build appropriate goals from these hypotheses and KSIs to satisfy these goals. The plan executor selects the appropriate KSI for the action from the KSI queue. If no acceptable KSI is available, the plan executor gives the plan a rating of 0 and reinserts it into the plan queue: the plan is suspended and may be retriggered (rerated) if new data arrives at the node. When the plan executor can associate a KSI with the action, the planner then sets the node's top (current) KSI to this KSI, and the node invokes the KS.

When the KS has completed, the plan executor retrieves the new hypotheses from the blackboard and checks them against the predicted results of the action. It thus monitors the actions and detects when the actual hypotheses produced deviate from the expected results. When this occurs, it invokes the repairing functions, which in turn might trigger more goal processing to generate new KSIs. The plan repairing functions may be unable to generate alternative actions to achieve the desired result, and if that occurs then the plan is aborted: it is rated to 0 and reinserted on (the bottom of) the plan queue. If it can find a repairing action, the planner inserts this as the next short-term action and reinserts the plan into the plan queue.

After the initial plans are formed, therefore, the planning activity cycles through choosing a current plan, finding the appropriate KSI for that plan, invoking the KSI, monitoring (and repairing) the results, and then once again choosing a current plan. This cycle continues until the termination mechanisms end problem solving: either a hypothesis has been formed that meets the solution criteria, or there are no plans for which KSIs can be found, or none of the remaining plans are likely to generate better solutions than those already found. At any time, plan generator can modify plans on the plan queue when new data arrives that changes the clustering hierarchy. When that happens, the plan generator modifies any existing plans on the plan queue and generates any needed new plans.

To illustrate how plans change over time, the plans on the plan queue at

selected times are shown in Figure 32. The plans are created at time 7, and $plan_1$ is pursued until time 14, when the common i-goals have been achieved so the plan is divided into two plans: $plan_1$ now only extends d_3d_4 into d_2 and a new plan $plan_3$ is generated to extend d_3d_4 into d_2' (Figure 32, time 14). The track d_3d_4 is used by the goal processing mechanisms to form a goal indicating where good data for sensed time 2 is most likely to lie, as is shown in Figure 33.

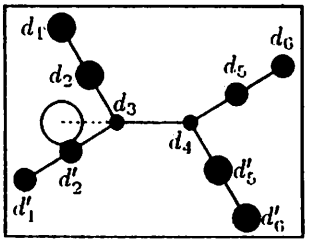


Figure 33: Expected Extension of Track d_3d_4.

This goal overlaps only with the data in d_2', so the KSIs to work on this data have their ratings increased while the ratings for KSIs in d_2 are unchanged. Although originally the KSIs for d_2 were more highly rated (since they work with more strongly sensed data), the KSIs to work in d_2' are now more highly rated. The plan to work on d_2' ($plan_3$) is therefore pursued because its more highly-rated KSIs cause its rating to be highest. Similarly, when this plan is divided into separate plans to extend either to d_5 or to d_5' (Figure 34, time 18), goal processing affects KSI ratings so that $plan_4$ (to work on d_5) is preferred. After generating the track $d_2'd_3d_4d_5$, however, the plan to continue developing this correct track is no longer the most highly-rated plan: because the track spanning times 2–5 has a lower belief than was predicted (the actual belief was reduced because of vehicle turns which the prediction mechanisms cannot take into account), $plan_3$ to extend the track $d_2'd_3d_4$ to d_5' becomes more highly rated (its predicted result's belief is higher and its fraction-complete is sufficiently high, so its overall rating is higher).[13] This plan is pursued (Figure 34, time 22) until the node is unable to join d_5' to $d_2'd_3d_4$ because of vehicle turning constraints. This plan $plan_3$ is aborted (its rating is set to 0), and $plan_4$ is resumed (Figure 34, time 26) and pursued until the solution is generated (Figure 34, time 34).

[13]Experiments with different weights for the various plan-rating factors are described in Section 5.1.2.

4.4 The Planning Mechanisms in General

The planner must concurrently reason at three different levels: it must reason about entire plans—rating alternative plans and following the best one at any time; it must reason about the intermediate goals for achieving plans—finding an ordering of these goals that allows it to efficiently form solutions or detect that solutions are not worth forming; and it must reason about detailed actions—identifying specific actions (KSIs) that may lead to desired results. Because plan ratings depend on the order of i-goals and the detailed actions, the i-goal ordering depends on the overall goals of the plan, and the detailed actions depend on previous i-goals, the planner must reason about all three levels interdependently.

In essence, the planner groups primitive actions (KSIs) and explicitly relates these actions to the larger goal (the intermediate-goal) to which they contribute. In turn, it relates intermediate-goals to larger plans for generating overall solutions. The planner improves control decisions by reasoning in a more top-down manner: instead of attempting to find the best KSI directly from a queue of many KSIs, the planner first chooses the most highly-rated plan, then the next i-goal of that plan (which it determined was the best i-goal to pursue next for that plan), and then finally the next KSI for that i-goal (which it determined was the best KSI to invoke next to achieve the i-goal). By explicitly representing higher-level goals and activities, the planner reasons more effectively about achieving long-term goals.

The top-down planning approach has been used in many other systems, but several aspects of the particular domain in this research affect how the planning is actually performed. First of all, the planner must formulate its own goals. In typical planning systems, the planner begins not only with a current "state" but also with a set of prioritized goal "states" to reach. The planner described in this chapter must use its current state to identify possible goals to achieve (this process was described in Chapter 3). A related aspect of the problem that affects the planner is the uncertainty of achieving goals. Even after it has identified possible long-term goals, the planner must develop plans that allow it to efficiently resolve uncertainty about which of these goals are actually worth pursuing. To resolve this uncertainty, the planner must invoke actions to construct partial solutions that it then attempts to use to differentiate between alternative long-term goals.

In differential diagnosis tasks [Clancey and Letsinger, 1981], each solution has a well-defined set of "confirming" and "refuting" hypotheses, and diagnosis consists of finding a hypothesis that confirms some solutions and refutes others to trim the set of possible solutions. In contrast to this, the planner outlined in this chapter has no such well-defined criteria for distinguishing between possible solutions, and instead must plan actions that incrementally construct partial solutions in such a way that, over time, certain partial solutions make some overall solutions look more plausible than others.

Another important aspect of the task is that a problem solver can maintain a record of all partial solutions it has constructed, so that its reasoning is monotonic in the sense that any actions it could have taken in the past are still viable in the present. Unlike tasks where planned actions cause changes to the physical world, a problem solver's planner need not worry about "undoing" previous actions to recover an earlier state.[14] The assumption that actions never have to be undone simplifies the planning mechanisms described in this chapter, but if the functions for detailing short-term actions are given additional information about actions to undo past actions (and models of those actions), the planner could handle domains where an action not only generates an output result but also changes its inputs.

The task also allows flexibility in how results can be achieved, and the planner exploits this flexibility. Because it works in a dynamic environment, the planner does not plan too far into the future because chances are that unexpected events will affect that future. Thus, the planner plans incrementally. The drawback with planning incrementally is that specific interactions between current actions and future actions may not be identified until the future: the planner might take actions that preclude important future actions because it did not look far enough ahead. The assumption behind the planner is that there is sufficient flexibility in how goals can be achieved, so that the planner can recover from unanticipated interactions between actions and from unexpected changes to the problem situation by pursuing an alternative set of actions that will achieve the desired result.

[14]By maintaining such a record, the problem solver can incur hefty storage costs. If it can undo actions to recover earlier states, the problem solver may decide not to save all the past information.

Finally, the planner assumes predictability in the task: that it can predict the relative costs of various i-goals and detailed actions and their likely result qualities. Because the planner uses these predictions when choosing what plan to pursue, when ordering i-goals for plans, when forming detailed sequences of actions for i-goals, and when monitoring these actions, prediction is of great importance to the planner's functions. In tasks involving fairly repetitive sequences of actions and where models of individual actions can be developed, predictions can be based on modeled activity and on past sequences of actions. In tasks without these characteristics, the planner would need an alternative source of such information, such as a database that associates activities with predicted outcomes and costs.

The planner described in this chapter was built for a problem solver that exhibits all of these aspects. It is therefore important to determine how generally applicable the planning mechanisms are: what information do they need, what simplifications based on the domain do they exploit, and how might they be extended to other planning tasks. Each of the major components of the planner are next discussed with these questions in mind. Afterward, the application of these planning mechanisms to other blackboard-based problem solvers, and to other more general planning tasks are discussed. Finally, some concluding remarks about the planner are given in the final section of this chapter.

4.4.1 Generalizing the Planner's Components

The planner has several components, such as the long-term planning mechanisms, the detailed planning mechanisms, the prediction mechanisms, and so on. Each of these uses certain information about the planning task when developing its contribution to the overall plan. Most of these components can generate their parts of the plan relatively independently. For example, the short-term planning mechanisms need to know what i-goal to find detailed actions for, but exactly how the long-term planning mechanisms decided how the i-goals should be ordered is immaterial to the short-term planning mechanisms. In the subsequent sections, therefore, each of the major components of the planner is examined to determine how the implementation described previously could be generalized for other domains, and how a completely different implementation would affect the remainder of the planning components.

Generalizing the Plan Data Structures

The principal divisions of the plan data structure are intended to be domain-independent: plans have objectives (the purposes of the plans); plans have long-term (strategic) activities and short-term (tactical or reactive) actions; plans have predictions about what is expected to occur during the plan (action preconditions, postconditions, and expected durations); and plans have some rating information that helps prioritize them. Although in this domain only two levels of planning are needed, long-term and short-term, the plan structures could be extended to include more levels as well. Within these major divisions are the more domain dependent plan attributes that allow plans for specific tasks to be represented.

The plan's objective attributes not only correspond to pointers to the plan's goals (which it either identifies for itself---as in this planner—or has provided to it by the user), but also give specifications for what the plan's intended solution will look like. In the vehicle monitoring domain, these specifications indicate probable vehicle types and movements. When different plans work toward competing solutions (solutions that could not possibly both be true simultaneously), the plan data structure can also keep track of this information (to reduce the time spent finding this information whenever it is needed).

The plan's long-term attributes provide an ordering of the major subgoals of the plan's overall goal, information about how the results of these subgoals should be combined, predictions about the costs of achieving each subgoal and the overall plan, and a summary of the activities that go into achieving the subgoals. In this domain, the ordering of subgoals is represented as a sequence of sensed times, the information about combining subgoal results is simply the blackboard-level for integrating hypotheses for different sensed times, the predictions come from the prediction mechanisms, and the activity summary is derived from the short-term detailed information. In other domains, the representation of this information may differ but its purpose and content would be similar.

The plan's short-term information links specific actions to the larger subgoals. The representation of these actions is the most domain specific aspect of the plans, which is not surprising since this is the level where the planner generates commands for the problem solver. The contents of these actions-- KS-types, KSIs, specific expected result attributes (which correspond to attributes of the hypotheses generated by the actions) - are based on the actions and results in the

task domain. Other domains could make use of the more general representation of the past and pending detailed plans as "stacks" of actions, where the next action is popped off of the pending stack, executed, and pushed onto the past stack. Also, keeping a record of data used by past actions could be more generally useful in domains where uncertainty about how to form a desired result means that several attempts may be needed so the planner should keep track of what has already been tried.

The prediction information indicates the expected costs and results of the major subgoals of the plan. In other domains, additional information might be useful as well, such as resource requirements other than computation time. Finally, the rating information outlines basic information that affects how important the planner should consider the plan, including the expected quality of the overall results, the amount of effort expended on the plan, how highly rated the next actions for the plan are, and the number of goals the plan concurrently pursues. These characteristics of plans are domain-independent, and other information might be useful in other domains, such as the resource costs for achieving the plan other than the computation time (which is considered in the plan's fraction-completed factor).

Generalizing Long-term Planning

The long-term planning functions order the major subgoals for achieving the plan's overall goals. The functions, as implemented, first identify the subgoals (i-goals) by scanning down the clustering hierarchy, and then use the relative cost and commonality of these subgoals to find an ordering. The decision about how to combine the subgoal's results is simplified by assuming that hypotheses are only integrated at one blackboard-level, and in the environments of interest in this research, this is the vehicle blackboard-level.

More generally, the planner need not have a clustering hierarchy to identify subgoals. It may have knowledge in a different form about how to decompose goals into subgoals. Furthermore, the ordering decisions need not be based on cost and commonality. Unlike this domain where the subgoals can be pursued in any order (although the costs of pursuing them and their results can be affected by the order), in other domains there may be precedence constraints between the subgoals. The mechanisms detailed in this chapter could be extended to in-

clude such factors: by including the relevant factors in the computation of overall subgoal costs, an ordering that takes them all into account can be generated. Finally, the same knowledge the planner uses to decompose a goal into subgoals often indicates how the subgoals' results should be combined into an overall result. For example, if the planner knows how to decompose an assembly task into subassembly tasks (which in essence means that it disassembles the assembly into subtasks), then it knows how to reassemble the overall assembly.

Generalizing Short-term Planning

The purpose of short-term planning is to find domain-level actions that achieve the desired subgoals. From the perspective of the other planning components, how these actions are found is irrelevant. The mechanisms detailed in this chapter take one approach: they assume that basic sequence of activities will achieve the desired results, and expend most of their effort not on choosing a sequence of actions but on modeling those actions to predict what kind of results they will generate. In other domains, the detailed planning could be done quite differently. When a larger variety of actions is available and subgoals are more difficult to achieve, planning techniques involving propagating constraints about actions to find sequences that achieve the subgoals would be needed [Stefik, 1981].

The detailed planning mechanisms in this chapter are therefore geared to this particular type of domain. Because they exploit the fact that there are several ways that the problem solver can achieve subgoals, they need not perform sophisticated search that is so important in other domains. The contributions they make to planning research are therefore not in how they address uncertainty about what actions to perform but in how they address uncertainty about what the results of those actions will be. Although models of actions have been used by most planners to simulate what a sequence of actions will do, the models and their use in this planner are more complex. When these models use domain knowledge to simulate the results of acting on some input "state" (all past results formed), they must track down relevant past results and identify what new results will be formed—both desired and incidental. Because KSs can generate unexpected results when they satisfy goals, the models must attempt to predict these results as well and may change the plan, such as dividing it into separate plans to generate different results (vehicle types).

Generalizing the Prediction Mechanisms

As previously described, the prediction mechanisms implemented in this planner exploit the repetitive nature of constructing solutions. The basic assumption that the prediction mechanisms make is that similar subgoals will be achieved in similar ways. They therefore compare the initial characteristics of a subgoal—the data that will be used to achieve it—with the characteristics of previously achieved subgoals, and expect the cost and result quality of the subgoal to be related to the initial data in the same way as the past subgoals' results and initial data are related. In this domain and with the short-term planner described above that uses the same basic sequence of actions for each i-goal, the assumed similarities between i-goals are valid. More generally, if subgoals with similar characteristics can be identified (subgoals with the same basic differences between the initial and goal states), then predictions for future activities can be based on past experience.

In other domains, achieving an overall goal may not involve pursuing a sequence of similar subgoals. Or even if the subgoals are similar, the actions taken to achieve them may vary. In these domains, prediction information must come from somewhere else, if it can be derived at all. For example, other mechanisms and knowledge may be added to the system to generate predictions, or a database that matches situations and subgoals with costs and results may be provided to the system.

Without the ability to predict costs and results for i-goals, the local planner can still function but its capabilities and sophistication are reduced. If it cannot predict result quality, it loses an important factor in rating plans: plan ratings will be based on less information and will therefore more likely lead to incorrect planning decisions. It also cannot make sophisticated termination decisions since it cannot recognize whether a plan may generate a better solution than one already developed. If it cannot predict time needs for plans, the planner cannot recognize when deadlines will be exceeded and so cannot alter plans to meet deadlines. Thus, prediction is important for many aspects of local planning, but the planner can still perform some of its functions without predictions. Prediction becomes much more important when the planner is attempting to coordinate several problem solvers during cooperative problem solving: without the ability to predict what results will be formed and when, the planner cannot anticipate when and how problem solvers should interact to work together effectively.

Generalizing Plan Ratings

The generality of the factors considered when rating plans was discussed above: these factors (expected result quality, fraction-complete, rating of next actions, number of goals concurrently pursued) would generally be of interest when rating plans in any domain. The techniques for measuring these factors depend on information found by other mechanisms: the expected quality from the prediction mechanisms, the fraction complete and rating of next actions from the short-term information, the number of goals from the plan's objectives. Assuming that similar mechanisms (or at least mechanisms that generated the same information in some way) are part of a planner, these factors can be computed in other domains as well.

This is not to say that other factors are not important. For example, in domains where certain resources are scarce, the rating factors should include more information about expected resource use of a plan. Since only expected computation time is considered in the current domain, and the time needs are of a plan are part of the fraction-complete rating factor, additional factors were not needed in this planner. The choice of factors in this domain was based on experience concerning what seemed to be important when choosing a plan; in other domains, similar experience will be needed to decide what factors to use.

Experience is also necessary when deciding how much each factor should contribute to the overall plan rating: only by experimenting with different weights can a set of weights that is generally appropriate be determined. To improve plan selection decisions, the decisionmaking might be based not simply on numeric factors but on more sophisticated methods for reasoning about the different influences that affect choice of plans. This remains an open research issue, but as long as the plans can be ranked in some way (prioritized on the plan queue), it is all the same to the rest of the planning mechanisms.

Generalizing Plan Monitoring and Repair

Since the short-term planning mechanisms generate expected results for each action, plan monitoring involves simply comparing actual results with those expected and recognizing when they do not agree. More generally, plan monitoring assumes that the planner has some model of expected behavior against which

it can compare actual behavior to recognize anomalies. The mechanisms described in this chapter use the specifications of expected results from actions and determines whether the actual results meet these specifications, allowing some tolerance for error (particularly in the predicted belief) since the predictions are only estimates.

The repair mechanisms that are part of the planner's current implementation are rudimentary. Although the monitor can detect when actual results fail to meet any of the expectations, the repairer is only capable of developing repair actions for certain types of errors. In short, when the action that failed was expected to generate a track meeting certain time-region specifications, the repair mechanisms finds the longest track consistent with those specifications but falls short of meeting all of them, and develops actions to extend this track to meet all the specifications. In more general terms, when the plan was expected to combine results from several subgoals into a single result and fails, the repairer finds the largest combined result made so far and develops actions that combine it with the results of the remaining subgoals.

This approach to repair assumes that no actions need to be undone before the recovery actions can be attempted. In domains such as assembly tasks, before an alternative assembly action can be attempted, some disassembly might be needed to get back the subassemblies. More sophisticated reasoning is needed in the repairing mechanisms to handle such situations. Therefore, the repair mechanisms currently implemented represent a framework for repair in this type of domain and provide an example of how repair can be accomplished—but much work needs to be done before the mechanisms more generally allow sophisticated recovery of failed plans. Because errors in integrating partial solutions (tracks) into larger partial solutions (tracks) are both the most common type of error in the experiments studied and the most disastrous to successfully solving the problem, the currently implemented repairing mechanisms are acceptably adept at recovery for current needs.

Generalizing Plan Modification

The plan modification mechanisms allow new problem data to affect plans. The clustering mechanisms that allow the clustering hierarchy to be updated enable the planner to identify when long-term goals have changed. Given new long-term

goals, the planner modifies its set of plans to achieve those goals: it modifies existing plans whose goals have changed and adds new plans for new goals. It also recognizes when previously unrelated plans become related because their long-term goals now interact. It may modify the set of plans by merging now related plans into a single plan so that it can more effectively reason about how to achieve the related goals.

In more general terms, the modification mechanisms allow the planner to react to a changing environment by altering goals and plans when circumstances change. When a plan's goals change, the planner usually recomputes most of the plan's attributes, taking into consideration the actions already performed on the plan and the i-goals already achieved. Therefore, applying the modification mechanisms to other domains essentially reverts to applying the basic mechanisms for generating plans. The main addition that the modification mechanisms involve is the functions for merging plans: if two previously unrelated plans are now related, if their joint goals can be pursued by one plan, and if their past work will not lead to conflicts, then the plans and their past actions can be combined.

Generalizing Planning for Deadlines

The mechanisms described above for meeting deadlines simply drop i-goals or actions from plans to generate less complete or less highly believed solutions. Decisions about how to reduce the scope of goals—about what i-goals to drop—are domain dependent: in the vehicle monitoring domain, the reasoning is that the most recent movements of vehicles are most important since they are most indicative of future movements. Similarly, decisions about what actions to drop depend on the particular domain and the problem solving knowledge that goes into generating results.

Our current mechanisms for meeting deadlines are therefore tuned to the particular domain. However, the basic reasoning behind them is more general. To meet deadlines, the planner must simplify the task: it must specify a less difficult goal to achieve. This might entail reducing the size of the goal (dropping subgoals) or relaxing the constraints on an acceptable solution (allowing less highly believed solutions satisfy the overall goal). Modifying a plan to meet deadlines may thus involve altering the long-term plans (by dropping i-goals to achieve) or the short-term plans (by replacing some detailed actions with less expensive actions or

no actions at all so that an inferior but still acceptable result is formed). The implemented mechanisms therefore represent a framework in which more complex reasoning about real-time constraints can be developed [Lesser and Pavlin, 1987].

The termination mechanisms are very general. Given plans for tracking vehicles or for any other task, these mechanisms compare the predicted result quality—the expected payoff of the plan—with the results formed by completed plans to determine whether a plan is worth pursuing. The most important open research issue with these mechanisms is whether more sophisticated decisionmaking techniques can be developed that reason more fully about the the costs and potential benefits of pursuing plans, instead of simply comparing predicted result beliefs with results that have already been developed.

4.4.2 Other Applications of the Planner

The planner described in this chapter has been implemented in the DVMT, and its success in improving performance in this domain (which is further discussed in the next chapter) indicates that it could be useful in other systems as well. In particular, the planner is well suited to blackboard-based interpretation systems, because these systems are usually data-driven and construct solutions through the repeated application of sequences of KSs.

In the previous chapter, the techniques for clustering were generalized to show how data-driven problem solvers, and in particular blackboard-based systems, have inherent in their problem solving knowledge the relationships that the clustering mechanisms need to generate the clustering hierarchy and identify long-term goals. The planning mechanisms can be generalized similarly. Given a clustering hierarchy, the planner can identify the important subgoals (i-goals) for the overall goal and can use heuristics concerning the costs, commonality, and other attributes of these subgoals (such as precedence relationships) to find a suitable ordering for subgoals. After it identifies the proper blackboard-levels for integrating results (based on the available KSs), the planner can detail the actions for achieving the next i-goal. It needs models of KSs that map stimulus-frames to response-frames to roughly predict the costs and results of sequences of KSs, it needs knowledge about the domain that permits it to make predictions about future i-goals (by extrapolating from past i-goals or by referencing a database linking data and i-goals to costs and results), and it needs knowledge about what

the KSs can do so that it can update and repair plans during execution.

Many types of problems demand that solutions be *constructed*—problems ranging from design to diagnosis—and the planner is useful in such task domains. For example, a chemical-synthesis problem solver may be given a list of available chemicals (data), knowledge about possible synthetic reactions (KSs), and a rough goal of forming any compound so long as it has certain characteristics (insoluble in water, non-acidic, polar, and a melting point above 100°C, for example). Given the initial set of chemicals and rough knowledge about what types of chemicals can be combined, it can group these together to recognize more specific types of compounds that may fit the bill (for example, polystyrenes, polyesters, and polyphenylenes). Next, since some of these compounds may have common precursors, the planner can group them together so that it can be working on several at once by generating a stock of intermediate compounds. By considering both the expected cost of forming these intermediates and the likelihood that they will be of use in some eventual product, the planner can sketch out a high-level sequence of intermediate compounds that should be developed on the way to forming potentially desirable complete compounds. For the next intermediate compound, the planner could then detail the actions needed to synthesize that compound: what reagents to mix, how to mix and heat them, and so on. Because reactions all too often fail to produce the desired product, the planner should not plan subsequent steps in detail until the next intermediate compound is formed. By analyzing the products of a reaction, the planner can determine whether the desired intermediate compound has been made, and if not, can propose modifications to the reaction or an alternative reaction that might produce the desired product. Moreover, if new chemicals arrive at the laboratory, the planner should check to see whether different goal compounds can be proposed and whether these affect how the intermediate compounds currently under development should be synthesized. The chemical synthesis domain therefore needs a planner with essentially the same features as the planner developed for the DVMT, and many of the mechanisms developed for the DVMT could be reimplemented for the chemical domain. Our directions for future research (Chapter 9) include revising the planning mechanisms to better separate the domain-independent from the domain-dependent aspects to simplify implementing our mechanisms in other domains.

The planner could also be useful in a job-shop or other task scheduling system where the tasks are interdependent and where there is uncertainty about their time (resource) needs and about whether they will successfully generate useful results. For example, a task scheduler may be given a set of large tasks where each of these tasks can be broken down into subtasks. Associated with each of the large tasks is a deadline for completion. To decide what subtasks to execute and when is not simply a scheduling problem because of the uncertainty involved: the task scheduler cannot find an optimal (or even satisfactory) schedule of tasks at the outset because it is uncertain about how long they will take and whether they will succeed. The scheduler thus needs a planning component to roughly map out how each larger task should be pursued to best resolve uncertainty about whether it can be successfully completed in time to meet its deadline. The planner can also take advantage of cases where the large tasks share subtasks. For example, several of the larger tasks might need to access a database, and so they all include a *get-user-authorization* subtask. By first identifying that they share this subtask and then by planning to pursue it early on, the planner can more efficiently pursue several tasks (if the subtask succeeds) or identify tasks that it cannot complete (if the subtask fails). The more detailed actions for achieving this subtask may need to be planned as well, and these detailed actions should be monitored and repaired if they fail (trying alternative ways of getting authorization, for example). In addition, the task scheduler may work in a dynamic environment where the set of tasks may change, so the planner must be capable of modifying its plans for achieving tasks dynamically. A planner like the one outlined in this chapter could therefore be suitable for this domain.

4.4.3 Local Planning: Conclusions

Local planning helps a node make intelligent control decisions. The planner we have described is useful in dynamically changing domains involving uncertainty about what goals to achieve and how to achieve them. It assumes that planning can be incremental, so that planned actions are not so tightly coupled that alternative actions to get to the same goals are impossible. The planner is therefore most appropriate in domains where it must plan constructive activities (incrementally assembling a solution or product out of partial results), where it is uncertain about what should be constructed and how to construct it, but where the domain

has enough flexibility so that if it fails to construct a result one way it has a good chance of finding another way.

Planning is a difficult problem, and to date the techniques developed have concentrated on solving certain aspects of the problem while simplifying or assuming away other aspects. As a result, different planning methodologies have been developed for different classes of planning problems. The planner described in this section is no exception: it is particularly useful for a certain class of problems while not as useful for others. Specifically, it is well suited for controlling the activities of a problem solver. Therefore, besides its practical contributions to the DVMT in particular, the planner contributes to planning technology in general because it represents mechanisms that are useful in a new class of problems that have these characteristics:

- the domain is dynamic (goals and potential activities may change), so that planning should be incremental and interleaved with execution;
- the goals are uncertain, so that the planner must resolve uncertainty about what goals it should work toward;
- the actions are uncertain, so that the planner must monitor and repair plans;
- the goals can be achieved in several ways, although some may be better (faster, cheaper) than others, so that the planner can react to unexpected events and action failures by changing plans to achieve goals differently;
- the goals are composed of several subgoals where predictions about achieving those subgoals can be made, possibly based on experience of achieving other subgoals;
- the goals may be simplified when deadlines must be met, such as reducing their scopes (number of subgoals that compose them) or decreasing their thoroughness (amount of effort spent on each subgoal).

Solutions did not appear so readily as before, and things were not so clear as they once seemed. Things were just not working out as planned. Nothing ran smoothly. Nothing was succeeding as planned. - Joseph Heller *(Good as Gold)*

Chapter 5

Local Planning: Experiments and Evaluation

Having described how the planner works in the previous chapter, we now examine how the planner affects problem solving. The first part of this chapter explores the activities of the planner and problem solver in a variety of experiments to better understand what the planner does. To fully understand the effects of the planner, these experiments not only examine how the planner improves control decisions, but also what the costs of those improvements are. It is important to remember that the planner's job is to reduce the time needed to solve problems by improving control decisions, but if the planner needs a lot of time to make these decisions, then the net result may be that the time needs increase—the time saved in problem solving is used up in planning! In many of these experiments, therefore, the discussion covers not only how the planner affects local decisions but also whether the costs of planning are acceptable.

The second part of this chapter presents an evaluation of the planner in terms of its costs, benefits, and limitations, drawing on the experimental results. The planner is summarized and open research issues are briefly discussed.

5.1 Local Planning Experiments

From the detailed description of the planner in the preceding chapter, it should be evident that the planner is fairly complex: it sketches out plans' long-term activities, it details their short-term actions, it makes predictions about the plans, it rates the plans to rank them, it monitors and repairs plans, it modifies its plans when relevant new information arrives, it decides when it has sufficiently pursued the plans, and it determines when to alter plans to meet deadlines. To better illustrate what the planner does, this section describes and evaluates the planner's activities in several experiments.

To simplify the discussion, the different experimental situations emphasize certain aspects of the planning mechanisms. The first set of experiments (Section 5.1.1) shows how the planner can rearrange activities to more quickly identify and generate promising solutions. These experiments stress how the planner improves problem solving by ordering the i-goals. In the second set of experiments (Section 5.1.2), the emphasis is on how the various plan-rating factors influence the choice of plans to pursue, and how this in turn affects problem solving. The third set of experiments (Section 5.1.3) illustrates how the planner monitors and repairs plans when planned actions fail to generate their expected results. To show how the planner modifies plans when new information arrives, the fourth set of experiments (Section 5.1.4) examines how varying times that sensor data arrives at the node affects planning and problem solving. Finally, the fifth set of experiments (Section 5.1.5) focuses on time issues: how the planner decides when to terminate problem solving and how it alters plans to meet deadlines.

The purpose of this section is not only to use experiments as illustrations of how the planner improves control decisions, but also to evaluate how cost effective the planner is. When evaluating the benefits and costs of the planner, the following factors are considered: how much does it improve control decisions (reduce the number of incorrect decisions), how much additional computation overhead does it require, and how much additional storage do the clustering-hierarchy and plans need. Since each control decision causes the invocation of a KSI, the first factor is measured using problem solving time: since each KSI is simulated to take 1 time unit, generating a solution at an earlier time means fewer KSIs were invoked so better control decisions were made. The second factor is measured as

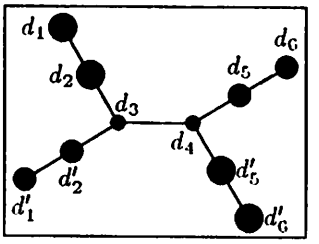
the actual computation time (runtime) needed to solve a problem, representing the combined costs of problem solving and control computation: if the combined problem solving and planning computation is less than the computation used when the node works without the planner, then the planning overhead is well spent. The third factor is computed by measuring how many problem solving and planning data structures were formed in the course of an experiment. Without the planner, the node generates hypotheses, goals, and KSIs as it progresses toward the result. The planner generates clusters (to find long-term goals) and plans. To roughly compare the storage needs of different experiments, the total number of hypotheses, goals, KSIs, clusters, and plans are compared: although the planner generates clusters and plans, it may improve problem solving so that fewer hypotheses, goals, and KSIs are formed.

5.1.1 Experiment Set 5.1: Planning to Resolve Uncertainty

The environment for the first set of experiments is shown in Figure 34, where the data in common regions is most weakly sensed. The data for sensed time 1 arrives at simulated time 1, data for sensed time 2 arrives at time 2, and so on: until node time 7, the node is simply receiving sensor data; from time 7 on, the node is solving the problem. In this experiment set, therefore, the node and its planner begin with all of the data and need not integrate more data over time (unlike the experiments in Section 5.1.4).

The previous chapter details how the attributes of this plan are determined and summarizes how plans evolve over time during problem solving. To recapitulate this summary, the plans are created at time 7 (Figure 34), and $plan_1$ is pursued until time 14, when the common i-goals have been achieved so the plan is divided into two plans: $plan_1$ now only extends d_3d_4 into d_2 and a new plan $plan_3$ is generated to extend d_3d_4 into d_2' (Figure 34, time 14). The track d_3d_4 is used by the goal processor to form a goal indicating where good data for sensed time 2 is most likely to lie (as was shown in Figure 33).

This goal overlaps only with the data in d_2', so the KSIs to work on this data have their ratings increased while the ratings for KSIs in d_2 are unchanged. Although originally the KSIs for d_2 were more highly rated (since they work with more strongly sensed data), the KSIs to work in d_2' are now more highly rated. The



d_i = data for sensed time i

● = strongly sensed

● = moderately sensed

● = weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5 d_6$

$ag_2 = d_1 d_2 d_3 d_4 d'_5 d'_6$

$ag_3 = d'_1 d'_2 d_3 d_4 d_5 d_6$

$ag_4 = d'_1 d'_2 d_3 d_4 d'_5 d'_6$

acceptable solutions: ag_3

Time	Plan	Rating	A-goals	V-e-cs	Current Result	Pend I-goals
7	$plan_1$	2719	ag_1, ag_2, ag_3, ag_4	(1)	none	(3 4 2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
14	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4213	ag_3, ag_4	(1)	$d_3 d_4$	(2 5 1 6)
18	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4316	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	4344	ag_3	(1)	$d'_2 d_3 d_4$	(5 1 6)
22	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4316	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	4240	ag_3	(1)	$d'_2 d_3 d_4 d_5$	(1 6)
26	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	0	ag_4	(1)	$d_4 d'_5$	(1 6)
	$plan_4$	4240	ag_3	(1)	$d'_2 d_3 d_4 d_5$	(1 6)
34	$plan_1$	4143	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	0	ag_4	(1)	$d_4 d'_5$	(1 6)
	$plan_4$	5906	ag_3	(1)	$d'_1 d'_2 d_3 d_4 d_5 d_6$	()

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node. The plans at various times are shown to indicate the evolution of the plans as problem solving progresses. For each plan is given its rating (out of 10000), the alternative-goals that it is expected to work toward, the Vehicle-event-classes of the vehicles it is attempting to track, the result it has generated to this point, and the pending i-goals in the order that they will be attempted.

Figure 34: Plan Summary for Experiment E5.1.2.

plan to work on d_2' ($plan_3$)is therefore pursued because its more highly-rated KSIs cause its rating to be highest. Similarly, when this plan is divided into separate plans to extend either to d_5 or to d_5' (Figure 34, time 18), goal processing affects KSI ratings so that $plan_4$ (to work on d_5) is preferred. After generating the track $d_2'd_3d_4d_5$, however, the plan to continue developing this correct track is no longer the most highly-rated plan: because the track spanning times 2–5 has a lower belief than was predicted (the actual belief was reduced because of vehicle turns which the prediction mechanisms cannot take into account), $plan_3$ to extend the track $d_2'd_3d_4$ to d_5' becomes more highly rated (its predicted result's belief is higher and its fraction-complete is sufficiently high, so its rating is higher). This plan is pursued (Figure 34, time 22) until the node is able to join d_5' to $d_2'd_3d_4$ because of vehicle turning constraints. This plan $plan_3$ is aborted (its rating is set to 0), and $plan_4$ is resumed (Figure 34, time 26) and pursued until the solution is generated (Figure 34, time 34).

The results for several experiments to evaluate the benefits and costs of the planner in this environment are summarized in Table 1. Experiments E5.1.1 and E5.1.2 illustrate how the planner can dramatically reduce the number of KSIs invoked during problem solving. Without the planner (E5.1.1), the node begins with the most highly sensed data (d_1, d_2, d_5', and d_6') that actually corresponds to *noise* and may have been formed due to sensor errors or echoes in the sensed area. The node attempts to combine this data through d_3 and d_4 but fails because of turning constraints, and then it uses the results from d_3 and d_4 to eventually work its way back out to the moderately sensed correct data. With the new mechanisms (E5.1.2), the planner begins in the area common to all four alternative-goals and then works outward. Through planning, the time needs to solve the problem are substantially reduced. The cost of planning is also acceptable in this case: the actual computation time required to solve the problem was reduced (E5.1.2 compared to E5.1.1); and the storage (total number of data structures) was also reduced because the number of goals (and KSIs to achieve them) is substantially lowered (the planner is smarter about forming only needed goals) and the number of hypotheses is also decreased (because fewer incorrect ones are formed).

The planner controls goal processing to generate and process only those goals that further the plan; if goal processing is done independently of the planner (E5.1.3), the overhead of the planner coupled with the only slightly diminished

Table 1: Experiment Summary for Experiment Set 5.1.

Expt	Plan	STime	Rtime	Hyps	Goals	KSIs	Cls	Pls	Store	Other
E5.1.1	no	57	17.5	189	254	76	0	0	519	-
E5.1.2	yes	34	8.8	150	51	47	59	4	311	-
E5.1.3	yes	40	17.3	163	205	64	59	4	495	ind goal
E5.1.4	no	57	21.2	247	276	98	0	0	621	-
E5.1.5	yes	74	19.8	293	138	97	69	6	603	*chw=0
E5.1.6	yes	47	15.6	234	74	79	69	6	462	-

Abbreviations

Plan: Are the new planning mechanisms used?
Stime: The simulated time (number of KSIs invoked) to find solution.
Rtime: The total runtime (computation time) to find solution (in minutes).
Hyps: The number of hypotheses formed.
Goals: The number of goals formed and processed.
KSIs: The number of KSIs formed.
Cls: The number of clusters formed in the clustering-hierarchy.
Pls: The number of plans formed.
Store: The total number of structures stored (storage costs).
Other: Additional aspects of the experiment (whether goal processing is independent or the *cost-heuristic-weight is 0).

goal processing overhead (the number of goals is only modestly reduced, comparing E5.1.3 with E5.1.1) nullifies the computation time saved on actual problem solving. The amount of storage needed by the node is also increased (E5.1.3 compared with E5.1.1). In addition, without the planner to control it, the goal processing builds goals and subgoals based on less complete results, and these less precise subgoals do not selectively increase the ratings of appropriate KSIs. For example, with the planner controlling goal processing, a goal to find data for sensed time 2 is only generated when the track covering d_3d_4 is formed, and this goal selectively increases the KSI ratings for d_2' as described above. Without the planner controlling goal processing, a goal to find data for sensed time 2 is formed earlier, based only on the result from d_3. Because it has no information about later vehicle locations, the goal indicates that data in any direction around d_3 is equally likely to be useful, and so increases the ratings of KSIs in both d_2 and d_2'—the KSIs in d_2 remain more highly rated. The planner then prefers the plan to work on d_2 over the correct plan. If the planner does not control goal processing, therefore, control decisions deteriorate and more KSIs may be invoked (E5.1.3 compared to E5.1.2).

The improvements in experiment E5.1.2 were due to the initial work done in the common areas d_3 and d_4. Because the expected cost of working in each area was expected to be the same, the preference for common i-goals dominated the ordering of the i-goals. In experiments E5.1.4–E5.1.6, areas d_3 and d_4 were flooded with numerous competing hypotheses at other event-classes. If the planner gives strong preference to working on common i-goals regardless of their cost (by setting the *cost-heuristic-weight to 0), then more KSIs are needed to develop all of these hypotheses (E5.1.5). In fact, the node would have solved the problem faster without the planner (E5.1.4). Since the planner developed an inefficient plan, the savings in computation and storage costs are very little. Estimating the relative costs of the alternative i-goals, the planner can determine that d_3 and d_4, although twice as common as the other areas, are likely to be more than twice as costly to work on. By considering both cost and commonality (by setting the *cost-heuristic-weight to 1), the planner forms a plan to develop the other areas first and then uses these results to more tightly control processing in d_3 and d_4. The simulated time needed to solve the problem is thus reduced (E5.1.6), along with the computation and storage overhead.

5.1.2 Experiment Set 5.2: Weighing Different Plan Rating Factors

In the experiment described in Figure 34, the planner temporarily deviates from the correct plan $plan_4$ to pursue $plan_3$ because the rating of $plan_3$ is higher. Because a plan's rating is based on a number of factors, the question naturally arises whether or not the weights for each factor are set suitably. In this experiment set, we explore the ramifications of weighing different factors in various ways, to understand both the need for each factor and the choice of weights for the remaining experiment sets (and the experiments in later chapters).

In the example in Figure 34, the planner chose to pursue an alternative plan ($plan_3$) after it had already completed more of $plan_4$, when in fact it should have continued with $plan_4$. As a result, the solution was found at time 34. In Table 2, the experimental results for that environment without the planner (E5.2.1) and with the planner (E5.2.2) are summarized again. To elicit better behavior from the planner, the weight for w_{fc}—the contribution of the fraction-complete rating factor—can be doubled: instead of each of the four normalized weights

Table 2: Experiment Summary for Experiment Set 5.2.

Expt	Plan	STime	Rtime	Store	Comments
E5.2.1	no	57	17.5	519	-
E5.2.2	yes	34	8.8	311	-
E5.2.3	yes	30	8.0	293	$2 \times w_{fc}$
E5.2.4	yes	38	9.7	329	$2 \times w_{pr}$
E5.2.5	yes	38	9.9	329	$2 \times w_{nr}$
E5.2.6	yes	38	9.6	329	$0 \times w_{nr}$
E5.2.7	yes	45	11.3	361	$197 \times w_{ag}$
E5.2.8	no	68	20.3	578	-
E5.2.9	yes	38	10.2	346	-
E5.2.10	yes	48	19.9	411	$2 \times w_{fc}$
E5.2.11	yes	30	8.4	293	$2 \times 2_{pr}$

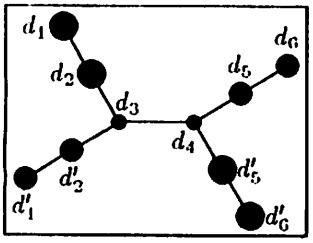
Abbreviations

Plan: Are the new planning mechanisms used?
Stime: The simulated time (number of KSIs invoked) to find solution.
Rtime: The total runtime (computation time) to find solution (in minutes).
Store: The total number of structures (hypotheses, goals, KSIs, clusters, plans) stored (storage costs).
Comments: Additional aspects of the experiment.

(see Section 4.3.2) being 0.25, w_{fc} is set to 0.4 and the other three are set to 0.2. When the experiment is rerun with these weights (Figure 35), $plan_4$ is never interrupted---once the planner has proceeded down one plan, it is much less prone to change to another. As a result, the solution is generated at time 30, which is in fact the *optimal* time [Durfee *et al.*, 1985a]. This experiment is summarized in Table 2, experiment E5.2.3.

Several more experiments were run to explore how changing the weights for other rating factors can affect node behavior. In experiment E5.2.4, the weight for the predicted result belief (w_{pr}) is set to .4, twice what the other weights are (Figure 36). Performance degrades compared to both E5.2.2 and E5.2.3 because the planner rated plans to develop d_2 and d'_5 highly: since strongly sensed data generally leads to more highly believed results, and since the planner has insufficient information for predicting how beliefs will be affected by vehicle turns, the best predicted results are for tracks involving d_1, d_2, d'_5, and d'_6. It is not until the plans involving this data fail that the correct plans are pursued.

Similarly, if the weight for the next result factor is twice the other weights, problem solving is also worse (E5.2.5). This time, the planner pursues plans that



d_i = data for sensed time i

= strongly sensed

= moderately sensed

= weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5 d_6$

$ag_2 = d_1 d_2 d_3 d_4 d'_5 d'_6$

$ag_3 = d'_1 d'_2 d_3 d_4 d_5 d_6$

$ag_4 = d'_1 d'_2 d_3 d_4 d'_5 d'_6$

acceptable solutions: ag_3

Time	Plan	Rating	A-goals	V-e-cs	Current Result	Pend I-goals
7	$plan_1$	2176	ag_1, ag_2, ag_3, ag_4	(1)	none	(3 4 2 5 1 6)
	$plan_2$	229	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
14	$plan_1$	3966	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	229	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4038	ag_3, ag_4	(1)	$d_3 d_4$	(2 5 1 6)
18	$plan_1$	3966	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	229	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4434	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	4476	ag_3	(1)	$d'_2 d_3 d_4$	(5 1 6)
22	$plan_1$	3966	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	229	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4434	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	4726	ag_3	(1)	$d'_2 d_3 d_4 d_5$	(1 6)
30	$plan_1$	3966	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	229	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4434	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	6588	ag_3	(1)	$d'_1 d'_2 d_3 d_4 d_5 d_6$	()

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node. The plans at various node times are shown to indicate the evolution of the plans as problem solving progresses. For each plan is given its rating (out of 10000), the alternative-goals that it is expected to work toward, the Vehicle-event-classes of the vehicles it is attempting to track, the result it has generated to this point, and the pending i-goals in the order that they will be attempted.

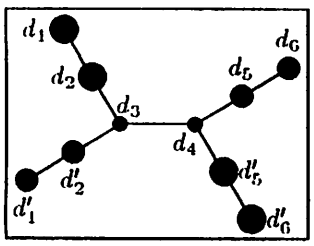
Figure 35: Plan Summary for Experiment E5.2.3.

generate the correct partial track $d_2' d_3 d_4 d_5$ but then gets sidetracked because the KSIs to extend this track are less highly rated than the KSIs to extend $d_3 d_4$ into the strongly sensed data.[1] By giving the KSIs such a large influence on plan ratings, the planner becomes much more likely to simply trigger the most highly-rated KSI to be pursued, which is exactly the decision that the node would have made without the planner. However, it is also wrong to completely ignore the KSIs. It is the influence of KSIs that causes the plan to extend $d_3 d_4$ into the less strongly sensed but correct data at d_2', for example. If the weight given to this rating factor is set to 0 ($w_{nr} = 0$), then performance also degrades (E5.2.6) relative to having the weights equal (E5.2.2).

Finally, the factor for the number of alternative-goals is generally a very minor factor: when it's weight is equal to the others, its influence is still very small (since it generally has values in the range of 1-10, compared with the other factors that often have values in the hundreds or thousands). The number of alternative-goals factor thus really acts as a tie-breaker between plans: if they are essentially identical in all other ways, the plan with the larger number of alternative-goals will be chosen. If the weight for this factor is set *very* high (.985) while the other weights are very low (.005), then alternative-goals do have a high influence. This experiment (E5.2.7) has very poor results: since the planner always pursues the plan with the largest number of alternative-goals, it begins by pursuing all of the plans until it has divided them all up so that each plan has one alternative-goal. Of the eight plans formed (one for each combination of the four alternative-goals and the two vehicle-event-classes), the planner then pursues the appropriate single plan, since once they have all been divided the other factors determine which will be pursued.

These experiments have shown how the influence of the various factors affect

[1] This is in part due to the planner's control of goal processing. The planner only triggers subgoaling when it faces uncertainty about how to extend a track—as is the case when it must decide whether to extend $d_3 d_4$ into d_2 or d_2'. Because subgoals alter KSI ratings, subgoaling in such situations can improve the planner's decisions. However, since the planner does not invoke subgoaling when there is no uncertainty—when it can only extend $d_2' d_3 d_4 d_5$ into d_1' and not d_1, for example—the KSIs for working in d_1' may not be as highly rated as they might if subgoaling were performed. Therefore, when the KSI ratings have a disproportionately large influence on plan ratings, the lack of subgoaling can cause poor plan choices. This is not, however, an error in the planner's use of subgoaling: since the purpose of subgoaling is to resolve uncertainty about how to extend a partial solution, it should not be invoked when there is no uncertainty. Rather, the error is in giving KSI ratings such heavy influence on plan ratings when the other factors should be considered as well.



d_i = data for sensed time i

● = strongly sensed

● = moderately sensed

● = weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5 d_6$

$ag_2 = d_1 d_2 d_3 d_4 d'_5 d'_6$

$ag_3 = d'_1 d'_2 d_3 d_4 d_5 d_6$

$ag_4 = d'_1 d'_2 d_3 d_4 d'_5 d'_6$

acceptable solutions: ag_3

Time	Plan	Rating	A-goals	V-e-cs	Current Result	Pend I-goals
7	$plan_1$	3980	ag_1, ag_2, ag_3, ag_4	(1)	none	(3 4 2 5 1 6)
	$plan_2$	429	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
14	$plan_1$	5045	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	429	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	5092	ag_3, ag_4	(1)	$d_3 d_4$	(2 5 1 6)
18	$plan_1$	5045	ag_1, ag_2	(1)	$d_3 d_4$	(2 5 1 6)
	$plan_2$	429	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4981	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	5002	ag_3	(1)	$d'_2 d_3 d_4$	(5 1 6)
22	$plan_1$	0	ag_1, ag_2	(1)	$d_2 d_3$	(5 1 6)
	$plan_2$	429	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4981	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	5002	ag_3	(1)	$d'_2 d_3 d_4$	(5 1 6)
26	$plan_1$	0	ag_1, ag_2	(1)	$d_2 d_3$	(5 1 6)
	$plan_2$	429	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4981	ag_4	(1)	$d'_2 d_3 d_4$	(5 1 6)
	$plan_4$	4763	ag_3	(1)	$d'_2 d_3 d_4 d_5$	(1 6)
30	$plan_1$	0	ag_1, ag_2	(1)	$d_2 d_3$	(5 1 6)
	$plan_2$	429	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	0	ag_4	(1)	$d_4 d'_5$	(1 6)
	$plan_4$	4763	ag_3	(1)	$d'_2 d_3 d_4 d_5$	(1 6)
38	$plan_1$	0	ag_1, ag_2	(1)	$d_2 d_3$	(5 1 6)
	$plan_2$	429	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	0	ag_4	(1)	$d_4 d'_5$	(1 6)
	$plan_4$	6069	ag_3	(1)	$d'_1 d'_2 d_3 d_4 d_5 d_6$	()

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node. The plans at various node times are shown to indicate the evolution of the plans as problem solving progresses. For each plan is given its rating (out of 10000), the alternative-goals that it is expected to work toward, the Vehicle-event-classes of the vehicles it is attempting to track, the result it has generated to this point, and the pending i-goals in the order that they will be attempted.

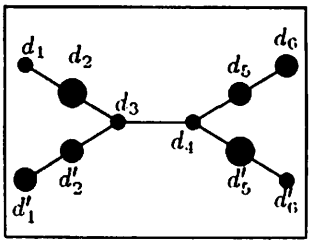
Figure 36: Plan Summary for Experiment E5.2.4.

the planner's choice of plans to pursue in a particular environment. In this environment, the fraction-complete and KSI factors seem to be the most important for rating plans appropriately. However, in other environments, other factors may be important. For example, in Figure 37 is an environment like the previous environment except that the tracks are symmetric (all turns are acceptable) and d_1 and d_6' are *weakly* sensed. In this environment, the best solution is still $d_1'd_2'd_3d_4d_5d_6$ because the overall belief of the moderately sensed data is higher than the combination of strongly and weakly sensed data. For comparison, the environment was run without the planner (E5.2.8) and with the planner weighting all factors equally (E5.2.9). The evolution of the plans for experiment E5.2.9 is shown in Figure 37, indicating that problem solving was not optimal: even though the predicted results for the correct track were higher, the influence of the more highly-rated KSIs in d_2 and d_5 caused plans to explore these areas to be more highly rated. However, these plans become less highly rated when data in d_1 and d_6' must be developed because this data is so weakly sensed. If the weight for fraction-complete is set to twice the other weights--as it was when optimal performance was achieved in E5.2.3—then the performance is worse (E5.2.10). Because the turns to d_2 and d_2' are equally sharp, the better KSIs belong to the plan to extend into the strongly sensed data d_2. Similarly, the better KSIs belong to the plan to extend into d_5' as well. As in Figure 37, the planner first forms $d_2d_3d_4d_5'$. Because the fraction-complete weight is so high, instead of moving on to other plans (since the KSIs to extend $d_2d_3d_4d_5'$ are so lowly rated), the planner continues working on the plans to extend into the weak data. The fraction-completed factor influences the planner to continue working on plans that it has already invested time into, but overly emphasizing this factor can cause the planner to continue pursuing bad plans. If instead the weight of the predicted result belief factor (w_{pr}) is twice the other weights, the planner pursues the correct plans from the start (since it predicts that the better result does not involve the most strongly sensed data) and the best solution is found in optimal time (E5.2.11).

Two things are clear from these experiments. The first is that no single set of weights will allow the planner to rate plans optimally in all situations. A static selection of weights for the rating factors may therefore provide good performance over a range of situations, but may be non-optimal for some of the situations. Because giving equal weights to the factors seems to generally result in good

balance of the factors, the remaining experiments in this and later chapters have equal weights.

The other conclusion that can be drawn from these experiments is that giving plans numeric ratings may not be the best way of choosing between them. Different, more symbolic representations for reasoning about uncertainty may prove useful in this planner, but as a simple first approximation, rating plans using a combination of numeric factors appears to provide a good if non-optimal way of deciding between alternative plans.



d_i = data for sensed time i

= strongly sensed

= moderately sensed

= weakly sensed

alternative-goals:

$ag_1 = d_1d_2d_3d_4d_5d_6$

$ag_2 = d_1d_2d_3d_4d'_5d'_6$

$ag_3 = d'_1d'_2d_3d_4d_5d_6$

$ag_4 = d'_1d'_2d_3d_4d'_5d'_6$

best solutions: ag_3

Time	Plan	Rating	A-goals	V-e-cs	Current Result	Pend I-goals
7	$plan_1$	2647	ag_1, ag_2, ag_3, ag_4	(1)	none	(3 4 2 5 1 6)
	$plan_2$	306	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
14	$plan_1$	4398	ag_1, ag_2	(1)	d_3d_4	(2 5 1 6)
	$plan_2$	306	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4010	ag_3, ag_4	(1)	d_3d_4	(2 5 1 6)
18	$plan_1$	4257	ag_2	(1)	$d_2d_3d_4$	(5 1 6)
	$plan_2$	306	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4010	ag_3, ag_4	(1)	d_3d_4	(2 5 1 6)
	$plan_4$	3997	ag_1	(1)	$d_2d_3d_4$	(5 1 6)
22	$plan_1$	2748	ag_2	(1)	$d_2d_3d_4d'_5$	(1 6)
	$plan_2$	306	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	4010	ag_3, ag_4	(1)	d_3d_4	(2 5 1 6)
	$plan_4$	3997	ag_1	(1)	$d_2d_3d_4$	(5 1 6)
26	$plan_1$	2748	ag_2	(1)	$d_2d_3d_4d'_5$	(1 6)
	$plan_2$	306	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	3857	ag_4	(1)	$d'_2d_3d_4$	(5 1 6)
	$plan_4$	3997	ag_1	(1)	$d_2d_3d_4$	(5 1 6)
	$plan_5$	4162	ag_3	(1)	$d'_2d_3d_4$	(5 1 6)
38	$plan_1$	2748	ag_2	(1)	$d_2d_3d_4d'_5$	(1 6)
	$plan_2$	306	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
	$plan_3$	3857	ag_4	(1)	$d'_2d_3d_4$	(5 1 6)
	$plan_4$	3997	ag_1	(1)	$d_2d_3d_4$	(5 1 6)
	$plan_5$	5089	ag_3	(1)	$d'_1d'_2d_3d_4d_5d_6$	()

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node. The plans at various node times are shown to indicate the evolution of the plans as problem solving progresses. For each plan is given its rating (out of 10000), the alternative-goals that it is expected to work toward, the Vehicle-event-classes of the vehicles it is attempting to track, the result it has generated to this point, and the pending i-goals in the order that they will be attempted.

Figure 37: Plan Summary for Experiment 5.2.9.

Table 3: Experiment Summary for Experiment Set 5.3.

Expt	Plan	STime	Rtime	Store	Comments
E5.3.1	no	74/78	20.1	699	-
E5.3.2	yes	46/57	14.7	443	-

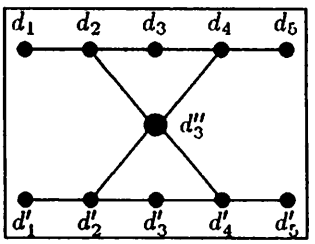
Abbreviations

Plan:	Are the new planning mechanisms used?
Stime:	The simulated time when each of the two solutions is found (time-for-first/time-for-second).
Rtime:	The total runtime (computation time) to find solution (in minutes).
Store:	The total number of structures stored (storage costs).
Comments:	Additional aspects of the experiment.

5.1.3 Experiment Set 5.3: Monitoring and Repairing Plans

The third set of experiments use the environment shown in Figure 38, where two solutions must be found, corresponding to two vehicles moving in parallel. Note that no areas are common to all alternative-goals. The experimental results for this environment are summarized in Table 3. Without the planner (E5.3.1), problem solving begins with the most strongly sensed data (the noise in the center of the area) and works outward from there. Only after many incorrect decisions to form short tracks that cannot be incorporated into longer solutions does the node generate the two solutions. In contrast, the planner allows the node to explore alternatives more effectively and recover from failed actions better, so that in terms of reducing the solution time, computational overhead, and storage overhead, the node with the planner is clearly superior (E5.3.2).

The data is received over the first 5 times. The clustering functions then develop the clustering hierarchy and allow the planner to recognize the six alternative-goals, four of which pass through d_3'' (the most common area). At time 6 (Figure 38), the planner initially forms $plan_1$, $plan_3$, and $plan_5$, beginning in d_3'', d_3, and d_3' respectively (it also triggers plans for the same data but with vehicle-event-class 2, and the lack of support causes these plans to be lowly rated). The time-order for these plans is (3 2 4 1 5): in $plan_1$ the planner prefers to begin at time 3 and work outward, and since this plan is divided to form $plan_3$ and $plan_5$ (initially all of the alternative-goals were grouped together), they inherit the same time-order. Since it works with more highly believed data at time 3



d_i = data for sensed time i

● = strongly sensed

● = moderately sensed

● = weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5$
$ag_2 = d'_1 d'_2 d'_3 d'_4 d'_5$
$ag_3 = d_1 d_2 d''_3 d_4 d_5$
$ag_4 = d_1 d_2 d''_3 d'_4 d'_5$
$ag_5 = d'_1 d'_2 d''_3 d_4 d_5$
$ag_6 = d'_1 d'_2 d''_3 d'_4 d'_5$

acceptable solutions: ag_1, ag_2

Time	Plan	Rating	A-goals	V-e-cs	Cur Result	Pend I-goals
6	$plan_1$	3732	ag_3, ag_4, ag_5, ag_6	(1)	none	(3 2 4 1 5)
	$plan_2$	562	ag_3, ag_4, ag_5, ag_6	(2)	none	(3 2 4 1 5)
	$plan_3$	3569	ag_1	(1)	none	(3 2 4 1 5)
	$plan_4$	428	ag_1	(2)	none	(3 2 4 1 5)
	$plan_5$	3569	ag_2	(1)	none	(3 2 4 1 5)
	$plan_6$	428	ag_2	(2)	none	(3 2 4 1 5)
9	$plan_1$	4093	ag_5, ag_6	(1)	d''_3	(2 4 1 5)
	$plan_7$	4093	ag_3, ag_4	(1)	d''_3	(2 4 1 5)
13	$plan_1$	4585	ag_6	(1)	$d'_2 d''_3$	(4 1 5)
	$plan_8$	5168	ag_5	(1)	$d'_2 d''_3$	(4 1 5)
20	$plan_8$**	5168	ag_5	(1)	$d'_1 d'_2$	(5)
21	$plan_8$*	0	ag_5	(1)	$d'_1 d'_2$	(5)
25	$plan_1$*	0	ag_6	(1)	$d''_3 d'_4$	(5)
28	$plan_7$**	4093	ag_3, ag_4	(1)	$d_2 d''_3 d'_4$	(4 1 5)
29	$plan_7$	4093	ag_3, ag_4	(1)	$d_2 d''_3$	(4 1 5)
30	$plan_7$	5061	ag_4	(1)	$d_2 d''_3$	(4 1 5)
	$plan_9$**	5644	ag_3	(1)	$d'_2 d''_3 d_4$	(1 5)
31	$plan_9$**	5644	ag_3	(1)	$d''_3 d_4$	(1 5)
32	$plan_7$	5061	ag_4	(1)	$d_2 d''_3$	(4 1 5)
	$plan_9$*	0	ag_3	(1)	$d''_3 d_4$	(1 5)
36	$plan_3$	3569	ag_1	(1)	none	(3 2 4 1 5)
	$plan_5$	3569	ag_2	(1)	none	(3 2 4 1 5)
	$plan_7$*	0	ag_4	(1)	$d_1 d_2$	(5)
46	$plan_3$	3569	ag_1	(1)	none	(3 2 4 1 5)
	$plan_5$	7325	ag_2	(1)	$d'_1 d'_2 d'_3 d'_4 d'_5$	()
57	$plan_1$*	0	ag_6	(1)	$d''_3 d'_4$	(5)
	$plan_3$	7325	ag_1	(1)	$d_1 d_2 d_3 d_4 d_5$	()
	$plan_5$	7325	ag_2	(1)	$d'_1 d'_2 d'_3 d'_4 d'_5$	()
	$plan_7$*	0	ag_4	(1)	$d_1 d_2$	(5)
	$plan_8$*	0	ag_5	(1)	$d'_1 d'_2$	(5)
	$plan_9$*	0	ag_3	(1)	$d''_3 d_4$	(1 5)

The plans at various node times are shown to indicate the evolution of the plans as problem solving progresses. Plans not included at a given time have not changed since the last time. For each plan is given its rating, alternative-goals, the types of vehicles, the result it has generated to this point, and the pending i-goals in the order that they will be attempted. Plans marked with a * have been aborted, and plans marked with ** have deviated from expectations and are either repaired (e.g., $plan_7$ times 28 and 29) or aborted (e.g., $plan_9$ times 31 and 32).

Figure 38: Plan Summary for Experiment E5.3.2.

and it has more alternative-goals, $plan_1$ is most highly rated. After developing d_3'' (Figure 38, time 9), $plan_1$ is divided into two plans to combine this data with either d_2 or d_2'. One of these equally rated plans, in this case $plan_1$, is chosen arbitrarily and forms the track $d_2'd_3''$, which then must be combined with d_4 or d_4', so this plan is again divided (Figure 38, time 13). Because the goal to extend $d_2'd_3''$ increases the ratings of KSIs in d_4 (favoring vehicles not turning), $plan_8$ to develop that data is more highly rated. However, after forming $d_2'd_3''d_4$, $plan_8$ is unable to extend this track into d_1' because of turning constraints. Only the result $d_1'd_2'$ can be formed (Figure 38), and the planner cannot repair the plan. The plan is aborted and $plan_1$ is pursued (Figure 38, time 21). This plan cannot join d_4' with $d_2'd_3''$, again because of turning constraints, and it thus is aborted (Figure 38, time 25).

Of the remaining plans, $plan_7$ is most highly rated and is pursued. With the goal to join d_2 to d_3'', the goal processor finds the track $d_3''d_4'$ formed by $plan_1$ on the blackboard. The combined track $d_2d_3''d_4'$ is unusable since d_4' is not consistent with both of $plan_7$'s alternative-goals (Figure 38, time 28). The plan repairing functions trigger the goal processor to form another KSI that combines only d_2 and d_3'', which is successfully accomplished (Figure 38, time 29). To pursue the different alternative-goals, moreover, the plan is divided, and the new plan, $plan_9$, is more highly rated and pursued. When attempting to join data in d_4 with d_2d_3'', the goal processor is only able to form a KSI involving hypothesis $d_2'd_3''$ (from $plan_1$ time 13) and the result formed by $plan_9$ is signaled by the plan monitor as deviating from the plan (Figure 38, time 30). The plan is repaired with a new KSI, and this time forms $d_3''d_4$ which also fails to include d_2 (Figure 38, time 31). The repairing functions cannot trigger any better KSIs, $plan_9$ is aborted, and $plan_7$ is pursued (Figure 38, time 32). This plan also aborts when it is unable to join d_1 to $d_2d_3''d_4'$ because of turning constraints (Figure 38, time 36). The planner then turns to $plan_5$, which it successfully completes (Figure 38, time 46). Finally, it also successfully completes $plan_3$ at time 57, and the status of all of the pursued plans are shown at this time in Figure 38.

5.1.4 Experiment Set 5.4: Incorporating Data Over Time

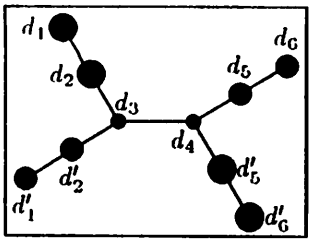
In the previous experiments, the data for all n sensed times were simulated to arrive at the node at times 1 through n. The node and planner therefore started problem solving at time $n + 1$ with all of the problem information. In this experiment set, data arrives at the node over time. The DVMT allows the user to specify the speed of node processing relative to the "sensed" world: for each sensed time, the user can specify the simulated time when the data arrives.

When all of the data arrives at the very beginning of problem solving, the case where node problem solving is slow relative to the outside world is simulated. Because the data is simulated to arrive much faster than it can all be processed, this type of situation stresses intelligent control to process only the most important data as quickly as possible. When data arrives at widely spaced intervals, the node has more time to process data before more data arrives. Problem solving is more spread out over time, and control is not as much of an issue: when it has time to exhaustively process data, why should it spend time deciding what data to process first? In this experiment set, a series of experiments are described where the relative rate of data arrival is varied.

At one extreme is the case where sensed time equals node time: the data for sensed time 1 arrives at node time 1, data for sensed time 2 arrives at node time 2, and so on. The experiment in Figure 34 shows how the planner allows the data to be processed in a timely fashion to generate the solution. Experimental results for this environment without the planner (E5.4.1) and with the planner (E5.4.2) are summarized in Table 4.

When the arrival times of sensor data are more spread out, the planning and problem solving activity changes. When the node is simulated to work three times faster than before, it receives sensor data for sensed time 1 at simulated time 1, data for sensed time 2 at time 4, data for sensed time 3 at time 7, and so on until the final data for sensed time 6 is received at time 16. Without the planner (E5.4.3), the node finds the solution at the same node time as before, although the actual computation time and storage needs are reduced.[2] With the

[2]The reduction in computation and storage costs is a result of reduced goal processing costs when data is incorporated over time. When the data arrives all at once, the node builds up several initial islands of high belief (partial tracks) and the goal processing must generate goals



d_i = data for sensed time i

● = strongly sensed

● = moderately sensed

● = weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5 d_6$

$ag_2 = d_1 d_2 d_3 d_4 d'_5 d'_6$

$ag_3 = d'_1 d'_2 d_3 d_4 d_5 d_6$

$ag_4 = d'_1 d'_2 d_3 d_4 d'_5 d'_6$

acceptable solutions: ag_3

Time	Plan	Rating	A-goals	V-e-cs	Current Result	Pend I-goals
2	$plan_1$	3171	ag_3, ag_4	(1)	none	(1)
	$plan_2$	301	ag_3, ag_4	(2)	none	(1)
	$plan_3$	3569	ag_1, ag_2	(1)	none	(1)
	$plan_4$	454	ag_1, ag_2	(2)	none	(1)
5	$plan_1$	3171	ag_3, ag_4	(1)	none	(1 2)
	$plan_2$	301	ag_3, ag_4	(2)	none	(1 2)
	$plan_3$	3569	ag_1, ag_2	(1)	none	(1 2)
	$plan_4$	454	ag_1, ag_2	(2)	none	(1 2)
8	$plan_3$	3537	ag_1, ag_2, ag_3, ag_4	(1)	d_1	(3 2)
	$plan_4$	283	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 2 1)
11	$plan_3$	3229	ag_1, ag_2, ag_3, ag_4	(1)	d_1	(3 4 2)
	$plan_4$	252	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 1)
14	$plan_3$	3637	ag_1, ag_2, ag_3, ag_4	(1)	d_1,d_3	(4 2 5)
	$plan_4$	273	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1)
17	$plan_3$	3518	ag_1, ag_2, ag_3, ag_4	(1)	d_1,d_3	(4 2 5 6)
	$plan_4$	285	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1 6)
18	$plan_3$	4554	ag_1, ag_2	(1)	d_1,$d_3 d_4$	(2 5 6)
	$plan_5$	4213	ag_3, ag_4	(1)	$d_3 d_4$	(2 1 5 6)
22	$plan_3$	0	ag_1, ag_2	(1)	$d_1 d_2 d_3$	(5 6)
	$plan_5$	4213	ag_3, ag_4	(1)	$d_3 d_4$	(2 1 5 6)
30	$plan_5$	4785	ag_3	(1)	$d'_1 d'_2 d_3 d_4$	(5 6)
	$plan_6$	4715	ag_4	(1)	$d'_1 d'_2 d_3 d_4$	(5 6)
34	$plan_5$	4639	ag_3	(1)	$d'_1 d'_2 d_3 d_4 d_5$	(6)
	$plan_6$	4715	ag_4	(1)	$d'_1 d'_2 d_3 d_4$	(5 6)
38	$plan_5$	4639	ag_3	(1)	$d'_1 d'_2 d_3 d_4 d_5$	(6)
	$plan_6$	0	ag_4	(1)	$d_4 d'_5$	(6)
42	$plan_5$	5963	ag_3	(1)	$d'_1 d'_2 d_3 d_4 d_5 d_6$	()

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node. The plans at various node times are shown to indicate the evolution of the plans as problem solving progresses. Plans not shown at a given time are assumed to be unchanged since the previous time.

Figure 39: Plan Summary for Experiment E5.4.4.

planner (E5.4.4), the node begins with less information about the problem and so the planner's decisions are less informed.

The behavior is summarized in Figure 39. To simplify this figure, the *eventual* alternative-goals are associated with plans even though the alternative-goals change (are extended) over time. After getting data for sensed time 1 (Figure 39, time 2), the planner forms 4 plans, one for each combination of data (d_1 and d_1') and vehicle types (1 and 2). It extends these plans for the data at sensed time 2 (Figure 39, time 5). When the data for d_3 arrives, the planner incorporates this into its plans and merges the plans together to pursue the common data (Figure 39, time 8). Note that even though d_1 has been developed, the plans can be merged since the plan cannot directly combine common data in d_3 with this result. The addition of data for sensed time 4 (Figure 39, time 11), sensed time 5 (Figure 39, time 14), and sensed time 6 (Figure 39, time 17) cause the plans to be extended. When the common data is processed, the plan is divided (Figure 39, time 18), with the past result for d_1 being associated only with the appropriate plan. The new plan's time-order (2 1 5 6) is patterned after the original plan's to most quickly generate a comparable result for sensed times 1–4. Since $plan_3$ is most complete, it is rated higher and is pursued and eventually aborted when it cannot form the track $d_1d_2d_3d_4$ because of vehicle movement constraints (Figure 39, time 22). In turn, $plan_5$ is pursued, and is divided to pursue competing alternative-goals (Figure 39, time 30). For a while, the plan to develop correct data (d_5) is followed, but the planner is diverted by $plan_6$ because its next activities are more highly rated (Figure 39, time 34). After this plan is aborted because of vehicle movement constraints (Figure 39, time 38), the correct plan is pursued until the solution is generated (Figure 39, time 42).

Next, the node is simulated to be even faster by making the interval between sensed data arrivals equal 6 (data arrives at 1, 7, 13, 19, 25, 31). Without the planner, the node's performance is not much affected (E5.4.5). Having this larger interval does not much affect the overall performance of the node with the planner (E5.4.6), but it does affect whether plans are merged. The evolution of the plans is summarized in Figure 40. After the first sensed data is received

and subgoals for each. When the data arrives over time, the node builds initial partial tracks only from the data it starts with, and then extends these into the data it receives later. Because it is extending fewer tracks at any given time, it is forming fewer goals and subgoals. The reduction in computation time and storage is therefore because of reduced control (goal processing) overhead.

Table 4: Experiment Summary for Experiment Set 5.4.

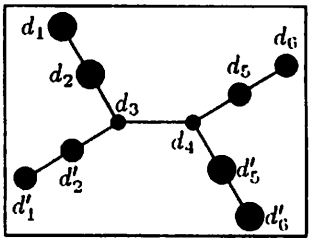
Expt	Plan	STime	Rtime	Store	Comments
E5.4.1	no	57	17.5	519	data interval of 1
E5.4.2	yes	34	8.8	311	data interval of 1
E5.4.3	no	57	12.6	453	data interval of 3
E5.4.4	yes	42	11.3	345	data interval of 3
E5.4.5	no	60	12.8	484	data interval of 6
E5.4.6	yes	44	11.7	170	data interval of 6
E5.4.7	no	63	14.4	515	data interval of 10
E5.4.8	yes	56	15.1	387	data interval of 10
E5.4.9	yes	56	13.8	359	data interval of 10, allow idle node

Abbreviations

Plan:	Are the new planning mechanisms used?
Stime:	The simulated time when each of the two solutions is found (time-for-first/time-for-second).
Rtime:	The total runtime (computation time) to find solution (in minutes).
Store:	The total number of structures (hypotheses, goals, KSIs, clusters, plans) stored (storage costs).
Comments:	Additional aspects of the experiment.

(Figure 40, time 2), the same four plans are developed as in Figure 39. When the next data is received (Figure 40, time 8), these plans are updated. The next received data that in the last experiment caused plans to be merged this time is received too late to cause plan merging (Figure 40, time 14). Since $plan_3$ has already developed results in d_1d_2, it cannot be merged with $plan_1$: when such a plan develops results in d_3 and combines them with past results, the track $d_1d_2d_3$ would be inconsistent with $plan_1$'s goals. The next data d_4 causes the plans to be further updated (Figure 40, time 20), but when $plan_3$ attempts to combine this data with $d_1d_2d_3$ it is unsuccessful and the plan aborts (Figure 40, time 23). Subsequent data for sensed time 5 modifies the plans (Figure 40, time 26). When the data for sensed time 6 arrives, the plans are updated appropriately, $plan_1$ also must be divided for competing alternative-goals (Figure 40, time 32). As in the past experiments, the correct plan is initially pursued, then the planner temporarily follows an incorrect plan with more highly-rated KSIs (Figure 40, time 36), and when this plan aborts (Figure 40, time 40) the correct plan is completed (Figure 40, time 44).

When data arrives even more infrequently—at intervals of 10—then not only does the planner have less information for making informed plans, but it also has



d_i = data for sensed time i

● = strongly sensed

● = moderately sensed

● = weakly sensed

alternative-goals:

$ag_1 = d_1d_2d_3d_4d_5d_6$

$ag_2 = d_1d_2d_3d_4d'_5d'_6$

$ag_3 = d'_1d'_2d_3d_4d_5d_6$

$ag_4 = d'_1d'_2d_3d_4d'_5d'_6$

acceptable solutions: ag_3

Time	Plan	Rating	A-goals	V-e-cs	Cur Result	Pend I-goals
2	$plan_1$	3171	ag_3, ag_4	(1)	none	(1)
	$plan_2$	301	ag_3, ag_4	(2)	none	(1)
	$plan_3$	3569	ag_1, ag_2	(1)	none	(1)
	$plan_4$	454	ag_1, ag_2	(2)	none	(1)
8	$plan_1$	3171	ag_3, ag_4	(1)	none	(1 2)
	$plan_2$	301	ag_3, ag_4	(2)	none	(1 2)
	$plan_3$	4800	ag_1, ag_2	(1)	d_1	(2)
	$plan_4$	454	ag_1, ag_2	(2)	none	(1 2)
14	$plan_1$	3015	ag_3, ag_4	(1)	none	(1 2 3)
	$plan_3$	4510	ag_1, ag_2	(1)	d_1d_2	(3)
	$plan_4$	283	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 2 1)
20	$plan_1$	3521	ag_3, ag_4	(1)	none	(1 2 3 4)
	$plan_3$	4581	ag_1, ag_2	(1)	$d_1d_2d_3$	(4)
	$plan_4$	283	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 1)
23	$plan_1$	3521	ag_3, ag_4	(1)	none	(1 2 3 4)
	$plan_3$	0	ag_1, ag_2	(1)	d_3d_4	()
	$plan_4$	283	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 1)
26	$plan_1$	5368	ag_3, ag_4	(1)	d'_1	(3 4 2 5)
	$plan_4$	283	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1)
32	$plan_1$	5094	ag_3	(1)	$d'_1d'_2d_3d_4$	(5 6)
	$plan_4$	283	ag_1, ag_2, ag_3, ag_4	(2)	none	(3 4 2 5 1)
	$plan_5$	4892	ag_4	(1)	$d'_1d'_2d_3d_4$	(5 6)
36	$plan_1$	4887	ag_3	(1)	$d'_1d'_2d_3d_4d_5$	(6)
	$plan_5$	4892	ag_4	(1)	$d'_1d'_2d_3d_4$	(5 6)
40	$plan_1$	4887	ag_3	(1)	$d'_1d'_2d_3d_4d_5$	(6)
	$plan_5$*	0	ag_4	(1)	$d_4d'_5$	(6)
44	$plan_1$	6308	ag_3	(1)	$d'_1d'_2d_3d_4d_5d_6$	()

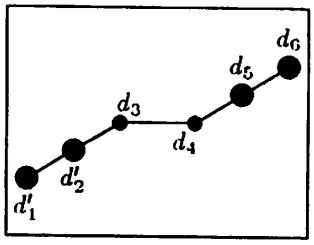
A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node. The plans at various node times are shown to indicate the evolution of the plans as problem solving progresses. Plans not shown at a given time are assumed unchanged from the previous time.

Figure 40: Plan Summary for Experiment E5.4.6.

less need for making informed plans: the node has enough time to exhaustively process data before new data arrives, so being intelligent about what data to process first is not as important. The last data arrives at time 51, and without the planner (E5.4.7), the node generates the solution at time 63. With the planner (E5.4.8), the node forms the solution at time 56--since it has already aborted the plan that uses d_6', it focuses on d_6 to generate the solution faster than if it did not have the planner. Note, however, that the computational overhead for planning exceeds the time saved by solving the problem only slightly faster. As a fail-soft mechanism, when the planner runs out of planned actions it invokes KSIs straight from the KSI queue based on their ratings. Thus, as long as there are any actions it can take, the node does not sit idle. When the planner only invokes planned actions (and the node sits idle the rest of the time), the solution is still formed at the same simulated time but the computational needs are reduced (E5.4.9). In this experiment, the node only invokes 48 KSIs and is idle for the remaining 8 time units waiting for more data to arrive.

The utility of the planner is also affected by the uncertainty in the data. When there is not much uncertainty in the data, the planner is not needed as much. For example, in Figure 41 is the basic environment in the other experiments in this set except that the strongly sensed noisy data (d_1, d_2, d_5', and d_6') is not included. Without the planner (Table 5, E5.4.10), the node solves the problem in 33 time units while with the planner (E5.4.11) the node needs 30 time units. Thus, the planner has only a small effect on the number of KSIs invoked and also makes only minor reductions in the computational time needs and storage needs of the node.

When the data has a lot of uncertainty, the planner's overhead can increase because the possible combinations of data (potential solutions) increases. For example, in Figure 42, data for two vehicles moving in parallel is so close that the clustering mechanisms recognize tracks made by crossing from the lower to upper track as potential solutions. The planner thus begins with 32 alternative-goals and must reason about their relationships and build plans to achieve them. Therefore, even though the node with the planner (E5.4.13) solves the problem faster than the node without the planner (E5.4.12), the computational overhead incurred by the planner is high: there may be times when planning should be postponed until the less sophisticated and cheaper control mechanisms have developed the data



d_i = data for sensed time i

● = strongly sensed

● = moderately sensed

● = weakly sensed

alternative-goals:
$ag_1 = d'_1 d'_2 d_3 d_4 d_5 d_6$
acceptable solutions: ag_1

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node.

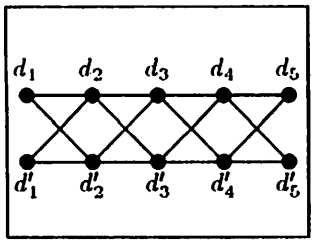
Figure 41: Environment for Experiments E5.4.10-1.

Table 5: Experiment Summary for Experiment Set 5.4 Continued.

Expt	Plan	STime	Rtime	Store	Comments
E5.4.10	no	33	7.0	265	low uncertainty
E5.4.11	yes	30	5.8	206	low uncertainty
E5.4.12	no	66/67	20.9	767	high uncertainty
E5.4.13	yes	29/53	19.0	440	high uncertainty

Abbreviations

Plan: Are the new planning mechanisms used?
Stime: The simulated time(s) to find solution(s) (if more than one, time-for-first/time-for-second).
Rtime: The total runtime (computation time) to find solution (in minutes).
Store: The total number of structures (hypotheses, goals, KSIs, clusters, plans) stored (storage costs).
Comments: Additional aspects of the experiment.



d_i = data for sensed time i

● = strongly sensed

● = moderately sensed

● = weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5$

$ag_2 = d_1 d_2 d_3 d_4 d'_5$

$ag_3 = d_1 d_2 d_3 d'_4 d_5$

$\vdots$

$ag_{32} = d'_1 d'_2 d'_3 d'_4 d'_5$

acceptable solutions: ag_1, ag_{32}

A problem environment is displayed, along with the possible solutions found by clustering the data and the acceptable solutions that can eventually be generated by the node.

Figure 42: Environment for Experiments E5.4.12-3.

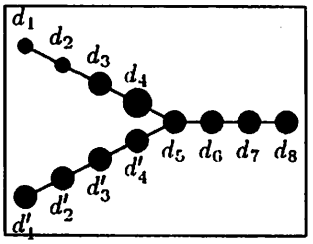
to a point where more selective use of the planning mechanisms can be made.

In summary, the planner is most useful when the node is receiving data faster than it can all be processed and when the data is noisy but not so noisy that the planner cannot initially generate a reasonably small set of potential solutions to consider. These experiments represent initial explorations to discover the costs and benefits of the planner when the rate of data arrival and the uncertainty in the data is varied.

5.1.5 Experiment Set 5.5: Dealing With Time Constraints

In the experimental environment shown in Figure 43, there are only two possible solutions: the vehicle can start either in the upper-left or lower-left corner. The track extending to the upper-left involves weak and strong data, but more weak than strong, while the lower track is moderately sensed throughout. The overall belief of the lower track is higher than the upper, and therefore represents the best solution. The experimental results are summarized in Table 6. For each experiment is shown the time when the best solution was found, the time the problem solving terminated (when appropriate), the sensed times of the solution track, the belief of the solution track, and comments about the experiment.

Depending on how conservatively it should solve the problem, the node may terminate problem solving only after it has explored every possible solution, or it may end as soon as it believes (based on its predictions) that it has found



d_i = data for sensed time i

= strongly sensed

= moderately sensed

= weakly sensed

alternative-goals:

$ag_1 = d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8$

$ag_2 = d'_1 d'_2 d'_3 d'_4 d_5 d_6 d_7 d_8$

best solutions: ag_2

A problem solving environment is displayed, along with the possible solutions found by clustering the data and the best solutions that can eventually be generated by the node (since in this case both of the possible solutions are valid solutions but one is better than the other).

Figure 43: Environment for Experiment Set 5.5.

Table 6: Experiment Summary for Experiment Set 5.5.

Expt	STime	TTime	Track	Belief	Comments
E5.5.1	40	57	1 – 8	*high*	Explore all solutions
E5.5.2	40	40	1 – 8	*high*	Stop with best predicted solution
E5.5.3	–	36	–	–	No predictions, deadline = 36
E5.5.4	36	36	1 – 8	*mod*	Predictions, deadline = 36, large scope
E5.5.5	32	32	1 – 8	*low*	Predictions, deadline = 32, large scope
E5.5.6	30	30	2 – 8	*low*	Predictions, deadline = 30, large scope
E5.5.7	28	30	4 – 8	*high*	Predictions, deadline = 30, high belief

Abbreviations

Expt:	The experiment
STime:	Time at which the best solution was found
TTime:	Time at which problem solving terminated
Track:	Times spanned by best solution track
Belief:	Belief in best solution track

the best solution. In experiment E5.5.1, the node explores all possible solutions, and therefore terminates problem solving much later than in experiment E5.5.2 where it stops as soon as it predicts that it has found the best solution. Although both experiments find the best solution equally fast, E5.5.1 spends a lot of time and energy verifying that it was the best solution while E5.5.2 predicted that the upper track would be inferior and did not pursue it. The predictions in these experiments are sufficiently accurate so that E5.5.2's termination decision is correct. In other situations such a decision might be premature--the predictions might underestimate the quality of as yet undeveloped results and the best solution might be missed. How conservative a node's termination decisions should be depends on the situation and how much time it has.

In this environment, a highly believed hypothesis spanning the entire track cannot be formed in less than 40 time units. A deadline of 36 time units was used in experiments E5.5.3 and E5.5.4. Since any partially developed track needs more work before it can be proposed as a solution (in this environment, the solution criteria insist that proposed solutions be at the pattern blackboard-level, and the planner develops tracks at the vehicle blackboard-level), a node without prediction abilities has no solutions by time 36 (E5.5.3). When it can predict that there is insufficient time to form the best solution (E5.5.4), the planner revises the plan to meet the deadline: told by the experimenter (the consumer of the solution) to favor scope over belief, it removes enough plan steps (KSs for supporting hypotheses) so that it still forms a solution covering all the sensed times but with only a moderate belief. With only 32 time units to work with (E5.5.5), the planner removes plan steps so that the solution covers all the sensed times but with low belief.

Given a deadline of time 30 (E5.5.6), the planner must employ both mechanisms since it still expects to exceed the deadline after it has removed all the unnecessary plan steps. It reduces the scope of the goal, dropping the earliest sensed time (time 1), and generates a hypotheses with low belief spanning times 2-8. Finally, given the same deadline of time 30 but told by the experimenter to generate a solution with high belief, the planner no longer drops steps (that would reduce belief) but instead reduces the scope of the solution to span times 4-8 (E5.5.7). Note that in these environments where each plan step (KSI) takes one time unit, the planner can drop just enough plan steps to meet the deadline

exactly (E5.5.3 - E5.5.5). When it needs to get high belief and drops entire i-goals for less important sensed times, the planner may not meet deadlines exactly (E5.5.7) since these i-goals take several steps to achieve.

5.2 Local Planning Evaluation

The experiments show how the planner forms and maintains plans, how the plans affect a node's behavior, and how the utility of the planner varies with the situation. By examining several situations, the general capabilities of the planner to resolve uncertainty, monitor and repair plans, modify plans when circumstances change, and meet deadlines were all investigated. Moreover, the planner's ability to improve control decisions with reasonable computation and storage overhead was shown.

Planning is not always the best thing to do under any circumstances. The planner is intended to improve control decisions, and is unnecessary when the node either has a lot of time to solve problems (and so does not need to make as intelligent decisions) or has little uncertainty about how to go about solving problems. Moreover, if the problem situation is extremely complex (large amounts of uncertain data), the planner will expend resources to generate control information that is still very uncertain. As described in Chapter 3, the complexity of forming the clustering hierarchy is quadratic with respect to the number of clusters: as the amount of data (and number of relationships between that data to keep track of) increases, costs of clustering increases even faster. In addition, as the number of alternative-goals increases, the number of plans increases: since each plan requires calculation of objective, long-term, short-term, prediction, and rating information, the amount of planning computation and storage increases in more complex situations.

There are two ways of addressing this problem: either to improve the planner's decisions about *when* and *how much* to plan, or to divide the data among several problem solvers when the amount of data becomes large enough to be potentially unmanageable. The latter approach of cooperative problem solving is the principal topic of the remaining chapters of this document. The former approach means that additional control mechanisms are needed to control the planner: meta-level control is needed. By using the new planner to control goal

processing, the benefits of goal processing (stimulating important activities) were achieved while the costs were reduced (since decisions about when to apply goal processing were made more intelligently by the planner). We expect that introducing control mechanisms such as the blackboard model of control developed by Hayes-Roth [Hayes-Roth, 1985] could promote a more intelligent use of the planner so that the benefits and costs of planning in a particular situation could be weighed and intelligent decisions about applying the planning mechanisms could be made. The development of such meta-level control components is an open area for future research (Chapter 9).

Because the planner changes the structure of blackboard-based problem solving, another important area for future research is studying how parallelism and the planner can be united in a common framework. The blackboard architecture as initially envisioned is a parallel system [Fennell and Lesser, 1977], with KSs acting on the blackboard concurrently and asynchronously. In practice, blackboard systems have generally been implemented on serial machines, so that difficult control decisions have to be made about which of the KSs should be executed at a given time. The planner works from this perspective as it attempts to find sequences of actions that effectively lead to results. If the blackboard system is built on a parallel architecture, the question arises whether the planner is still needed. The planner is needed if the number of parallel processors is less than the number of KSIs that could potentially execute in parallel, because then control is still needed to decide which actions should take place at a given time. Moreover, even when every KSI could be run in parallel, planning might be desirable to limit which KSIs are run since unnecessary parallelism can degrade performance because of contention for the shared blackboard. Given a parallel architecture, an important open research issue concerns whether the granularity of parallelism should be at the plan level (processors work on different plans), at the i-goal level (processors work on different i-goals of the same plan), or at the KSI level (processors work on different KSIs of the same i-goal).

There are several other areas where the planner opens avenues for further research. The planner allows temporal predictions about forming solutions that allows the exploration of issues in real-time problem solving [Lesser and Pavlin, 1987]. The planner also gives a higher level view of problem solving behavior so that errorful behavior (because of incorrect assumptions or problem solving

knowledge, for example) can be detected, and the plans can be used to help diagnose where the errors occur. Finally, the planner provides information about a node's activity at a level of detail that is appropriate for coordinating and organizing nodes to solve problems cooperatively. This is the subject of the remaining chapters.

... 'twixt such friends as we
Few words suffice ...

—Shakespeare *(The Taming of the Shrew)*

Chapter 6

Recognizing Partial Global Goals

A problem solver forms local goals and plans to solve its local problems. These problems, however, may be subproblems of some larger problems. In a vehicle monitoring domain, for example, one node may be locally responsible for tracking vehicles in a particular area while other nodes are responsible for tracking vehicles in other, contiguous areas. These nodes are together working on the larger problem of tracking vehicles in their combined areas. After successfully solving their subproblems—tracking vehicles in their individual areas—the nodes must communicate about their subproblem solutions so that the larger problem of mapping vehicle movements in the overall area is solved. The nodes thus *cooperatively* solve the overall problem by individually solving their interacting subproblems and integrating the subproblem solutions.

To cooperate effectively, the nodes should coordinate their plans. They should consider how forming and communicating about partial solutions to their subproblems will lead to solving the larger problem, and should plan their local activities accordingly. Before they can plan activities to solve the larger problem, however, they must first identify that in fact they are working on subproblems of a larger problem. To plan for cooperation, the nodes need to identify the larger

goals that they are cooperating to achieve. These goals are called **partial global goals** (PGGs): they are *global* in the sense that they can encompass the local goals of several nodes, but only *part* of the entire network of nodes may contribute to a single PGG. When nodes individually form their local goals and plans, they need to communicate about these goals and plans to recognize PGGs, and then build **partial global plans** (PGPs) to achieve the PGGs.

To identify PGGs for groups of nodes, the planner needs non-local information—it needs a *network-model* to represent the plans and goals of individual nodes and of groups of nodes. The focus of this chapter is on how nodes model each other and the network as a whole, and use these models to recognize PGGs and determine *when* they should cooperate; in the next chapter, we describe how these nodes then plan *how* they should cooperate to achieve these PGGs. A node's network-model thus consists of node-models (one for each node) and a set of PGPs (indicating groups of potentially cooperating nodes). This chapter describes what information is in a network-model and where this information comes from, how PGGs are identified, and how PGPs are used to represent and communicate about more global goals and activities. The next section provides background information about related research—modeling other nodes (or more generally agents) and groups of nodes are common issues in distributed systems and natural language processing research. In the subsequent section, an overview of the mechanisms for representing and reasoning about different nodes' goals and plans is presented. Next, these mechanisms are described in detail in Section 6.3, including a detailed example of how these techniques are applied in a specific cooperative situation. Finally, the last section of this chapter generalizes the ideas presented, indicating how they could be used in other domains.

6.1 Background

Modeling other agents is an important topic in distributed computing research in general: in distributed problem solving, each problem solver needs to know enough about others to coordinate problem solving; in distributed database systems, each local database manager needs to know whether other databases are accessible and where to look in case of inconsistencies in information; in a distributed operating system, each processor needs to know about the available resources at

other nodes when determining how to assign computing tasks. Modeling is also used in fields such as natural language understanding where decisions about what information to communicate and how to communicate it must be based on some knowledge about how the recipient of the information will respond to it.

The two most important aspects of one node's model of another are what type of information is stored in the model (at what level does one node view another) and how the node got that information. The type of information stored depends on how the model is to be used, determining how much detail one node needs about another. At one extreme, the information might be facts about what the other node knows and what it does with this knowledge. For example, in the system described by Konolige, a node's model of another node consists of the logical assertions it knows the other node believes and the inference operators that it knows the other node has for generating new assertions [Konolige, 1984]. Similarly, Rosenschein and Genesereth describe a system where a node's model of another consists of the logical statements it knows that the other possesses [Rosenschein and Genesereth, 1987]. A model at this much detail allows nodes to develop extremely specific expectations about each other since they can essentially duplicate each other's reasoning.

At the other extreme, the information might be very rough specifications about the general behavior of another node. In the DVMT, for example, the organizational structure gives each node a model of the others: the responsibilities for problem solving that each possesses and the general characteristics (spatial, temporal) of hypotheses and goals that each might produce. Based on this model, a node does not know how another node forms results, when those results will be formed, or if it is even forming any results at a given time. All a node knows is that *if* another node is forming results, *then* those results will meet the very general specifications.

Between these extremes are models of nodes' goals, plans, and behavior. For example, in Georgeff's process model approach, nodes only use specifications of each other's *outward* behavior to identify cases where their plans may interfere [Georgeff, 1984]. Cohen's speech acts system uses nodes' goals in their models of each other [Cohen, 1978]. Carver *et al.* describe an office information system that models the user in terms of the user's plans [Carver *et al.*, 1984]. In all of these systems, the level of detail was chosen so that interacting nodes could

model their interactions without modeling the internal actions that others were taking to get to those interactions.

The other important aspect of modeling nodes is how a node gets its models of others. In some systems, nodes' models of each other are statically defined. For example, in the DVMT the organizational information is established during network creation (or at specific "reorganization" intervals) and is not maintained as part of basic network activity. It is because organizational information is so infrequently updated that it should be so underspecified, as discussed in Chapter 2. In other systems, a node builds a model of another by observing its activity and inferring its plans based on that activity. In an office automation system, for example, the user's actions are analyzed to hypothesize what the user's plans and intentions are [Carver *et al.*, 1984]. In most systems, however, a node builds a model of another based on what the other node tells it: the node communicates information about itself to help others model it better. Nodes that know they are cooperating (or interacting in some other way) can explicitly communicate information that will affect other nodes to better achieve their mutual (or selfish) goals.

The nodes' models of each other described in the next sections are built by having nodes explicitly communicate with each other about their local plans (and goals, since a plan's goals are represented as its objectives). However, because the purpose of modeling each other is to improve how they coordinate their interactions, nodes need not exchange detailed information about their plans but instead just long-term information about their projected activities, indicating what results they expect to generate and when. The planner described in the past few chapters thus builds plans that are particularly appropriate when it comes to modeling other nodes: by representing plans at both a long-term and short-term level of detail, the long-term information crucial for coordination can be extracted and exchanged easily.

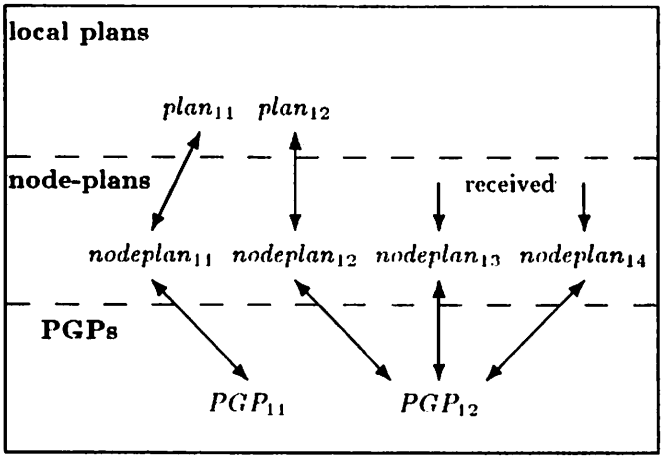
Modeling groups of nodes involves basically the same issues: deciding at what level of detail the goals and activities of the group should be represented, and where this information should come from. A group of cooperating nodes can be represented as a single component in an organization [Pattison *et al.*, 1985], or by a multi-node plan. These models of group activity can be inherently part of a node's knowledge, can be inferred based on the activity of the group, or can

be explicitly communicated about. Our approach to modeling groups of nodes is the same as modeling individual nodes: nodes represent a group's activities in a plan structure where the plans are at an appropriate level of detail, and node's communicate about these plans to build up a view of how groups of nodes are behaving in the network.

6.2 Overview of the Mechanisms for Recognizing Partial Global Goals

To recognize opportunities for cooperation, a node needs a dynamic model of network activity (Figure 44). The node has the *local plans* that it forms based on its view of the problem situation. For example, node 1 in Figure 3 will have two local plans, and the one to form the track d_4–d_{12} is outlined in Figure 44. The node's planner summarizes each of its local plans into a *node-plan* that specifies the goals of the plan, the long-term order of the planned activities, and an estimate of how long each activity will take (based on the time costs of the plan's past and current activities). The planner uses this information to generate the node-plan's *plan-activity-map*: a series of activities, where each activity has a predicted starting time, ending time, and result track. The node-plan for node 1's local plan to form track d_4–d_{12}, for example, is outlined in Figure 44. Since node-plans have much less detailed information than local plans and do not point to local data structures, nodes can cheaply exchange them and can reason about others' node-plans as they can their own. Thus, nodes can communicate node-plans to build up models of each other [Corkill, 1979; Georgeff, 1984; Konolige, 1984].

A node's planner scans the model of the network to recognize *partial global goals* (PGGs). A PGG is global in the sense that it may (but does not necessarily) encompass the local goals of several nodes, and is partial in that only part of the network might participate in it. The planner identifies PGGs by comparing local goals and using simplified knowledge (in this domain, about allowable vehicle movements) to determine whether they are part of a larger goal (tracking the same vehicle). For each PGG, the planner forms a *partial global plan* (PGP) that represents the concurrent activities and intentions of all of the nodes that are working in parallel on different parts of the same problem (to potentially solve



$plan_{12}$: cluster information = (cl_{11} ... cl_{1n})
goal track = ((4 $region_4$) ... (12 $region_{12}$))
goal vehicle types = (1 2 ...)
long-term times order = (4 5 ... 12)
long-term cost estimates = ((4 c_4) ... (12 c_{12}))
past actions = (4 $action_1$ ... $action_i$)
pending actions = (4 $action_{i+1}$ ... $action_n$)

$nodeplan_{12}$: goal track = ((4 $region_4$) ... (12 $region_{12}$))
goal vehicle types = (1 2 ...)
long-term times order = (4 5 ... 12)
long-term cost estimates = ((4 c_4) ... (12 c_{12}))
plan-activity-map = ((t_0,t_0 + c_4,d_4)
(t_0 + c_4,t_0 + c_4 + c_5,d_4-d_5) ...)
plan = $plan_{12}$

PGP_{12}: participants = ($plan_{12}$ $plan_{21}$ $plan_{31}$)
goal track = ((1 $region_1$) ... (15 $region_{15}$))
goal vehicle types = (1 2 ...)
plan-activity-map = ((t_0,t_0 + c_4,d_4) (t_0,t_0 + c_1,d_1) ...)
s-c-graph = (3 d_1-d_3) + (1 d_4-d_9) → (1 d_1-d_9) ...
communication = ((3 1 d_1-d_3) (1 2 d_1-d_9) ...)

The network-model of node 1 from Figure 3 is graphically depicted along with simplified views of the data structures. A **local plan** has pointers to local data clusters, a goal track (a region for each sensed time) and vehicle types, the long-term order for processing data (data for sensed time i, then j, etc) and the estimated cost for each (time-cost pairs), and finally the specific KSs to process the next data. A **node-plan** has the local plan's goals and long-term information, and has an plan-activity-map (each activity has a start-time, end-time, and result-track). A **PGP** points to participating plans, combines their goals, interleaves their node-plans' plan-activity-maps, and has a solution-construction-graph and communication predictions (for example, node 3 forms d_1-d_3 and node 1 forms d_4-d_9 and node 1 combines them into d_1-d_9).

Figure 44: An Example of a Node's Network-Model.

it faster). For example, the PGP to generate the overall track d_1–d_{15} (Figure 3) by combining the plans of nodes 1, 2, and 3 is outlined in Figure 44. The planner interleaves the participating node-plans' plan-activity-maps to recognize the relative timing of the nodes' activities and to discover how activities might be reordered to avoid harmful interactions (such as performing redundant activities) and to promote helpful interactions (such as providing predictive information sooner).

Given a suitably ordered set of activities for the participating nodes, the planner uses this view of how nodes will act to develop expectations about how they will interact. It estimates when a node will complete a group of activities that together form a sharable result, and forms a *solution-construction-graph* that indicates how and where results should be integrated. For example, in the situation of Figure 3, node 3's track d_1–d_3 and node 1's track d_4–d_9 may be combined at node 1 to form track d_1–d_9 (Figure 44). The planner uses its network-model to assign integration tasks to nodes with available computation resources or suitable expertise. Thus, while the plan-activity-map provides details about how each node will form its own results, the solution-construction-graph provides a high-level view of how the nodes are pooling resources and working together. This view of node interactions helps nodes avoid wasting communication resources because they can better identify important communication actions (to send track d_1–d_3 from node 3 to node 1 as in Figure 44, for example).

To summarize, a network-model has three types of information:

local plan: The representation of a plan maintained by a node that is pursuing the plan. Contains information about the plan's objective, the order of major plan steps, how long each is expected to take, and detailed actions (KSs) that have been taken or will be taken.

node-plan: The representation of a plan that nodes communicate about. Contains information about the plan's objective, the order of major plan steps, and how long each is expected to take. Details about short-term actions are not represented.

PGP: The representation of how several nodes are working toward a larger goal. Contains information about the larger goal, the major plan steps that are

occurring concurrently, and how the partial solutions formed by the nodes should be integrated together.

A PGP can be formed for any number of nodes with compatible local goals. Initially, a node's PGPs correspond only to its local plans, but, as information from other nodes arrives, it builds larger, more encompassing PGPs. Because nodes build their network-models asynchronously and over time, they may be incomplete or out-of-date and cooperating nodes may have inconsistent PGPs. The extent and quality of a node's network-model and how it is formed depends on the *meta-level organization*: the communication topology, capacity, delay, and reliability; the coordination responsibilities of different nodes; the credibility that a node has in coordination information from other nodes (which determines their authority relationships); and so on. The distributed problem solving network is therefore organized both in terms of problem solving responsibilities (the domain-level organization) [Corkill, 1983; Corkill and Lesser, 1983] and in terms of coordination responsibilities (the meta-level organization).

The meta-level organization is statically defined during network creation, although our future research directions include developing organizational self-design mechanisms for the network. A node sends its node-plans to those nodes specified in the organization, perhaps to a particular coordinator-node, or maybe to all other nodes so that they recognize PGPs individually (as in Figure 44 where node 1 receives node-plans from 2 and 3 and forms its own PGPs). When it coordinates other nodes, a node may send them PGPs to guide their actions, but two nodes with equal authority may also exchange PGPs to negotiate about (converge on) a consistent view of coordination. A node that receives a node-plan or PGP considers the sending node's credibility when deciding how (or whether) to incorporate the new information into its network-model: it can follow a highly-rated PGP from a much trusted coordinator-node, but may disobey coordination requests if the credibility and ratings of its local information is superior. The meta-level organization therefore influences how nodes may converge on consistent views of network activity and how responsive they will be, allowing various levels of autocracy and democracy, obedience and insubordination.

Although they often convey similar information, node-plan and PGP messages serve different purposes. In some organizations, a node might not have authority to locally change a received PGP even if it believes the changes represent im-

provement. By sending its local view as a node-plan, it might persuade a node with more authority to change the PGP, or it could persuade other low-authority nodes that they should all change their PGPs together (in a kind of "grass-roots" movement). PGP messages say how nodes *are* working together, while node-plan messages provide context for deciding how nodes *might* cooperate. Finally, nodes could send only some of a PGP's information (leaving out the planned activities of some participants) so that recipient nodes have insufficient context to recognize and suggest improvements. This enforces consistent views and reduces computation at recipient nodes, since they blindly follow the PGP and cannot explore alternatives.

6.3 Details about the Mechanisms for Recognizing Partial Global Goals

By communicating the objectives and long-term information of their plans, nodes can exchange information at the right level of detail and build up models of each other that they can then use to recognize when they should cooperate and how they should cooperate. In the next section, the principal data structures for modeling the network are described. Section 6.3.2 outlines how node-models are initialized and then Section 6.3.3 discusses how node-plans are built. After this, Section 6.3.4 describes how nodes communicate about node-plans. In Section 6.3.5, issues in maintaining node-models in changing situations are presented. Section 6.3.6 then discusses how the planner scans node-models to recognize more global goals, and Section 6.3.7 summarizes the contents of network-models. Finally, Section 6.3.8 gives detailed examples of how these mechanisms work.

6.3.1 Data Structures for Modeling Nodes and the Network

To determine whether and how they should cooperate with others, nodes exchange *node-plans*, which are summaries of their local plans. The attributes of a node-plan are shown in Figure 45. A node-plan has a name, and has attributes specifying the name of the local plan on which is based, the node that has the plan, the rating of the plan, and the creation-time of the plan. To indicate the goals of the plan, the node-plan copies the plan's objective track and vehicle type

name	A unique name (symbol) pointing at the node-plan structure	
plan-name	The name of the local plan summarized in this node-plan	
node	The node that has the local plan summarized in this node-plan	
rating	The rating of the local plan summarized in this node-plan	
creation-time	When the summarized local plan was created or last updated	
objective	track	The expected track for the local plan
	v-event-classes	The vehicle-event-classes (types of vehicles) being tracked by the local plan
long-term	time-order	The order the local plan will pursue i-goals, where each i-goal is identified by its data's sensed time
	time-predictions	The predicted or actual time needs of each i-goal and for the entire local plan
	tracking-levels	The blackboard-level(s) where hypotheses will be integrated into tracks
	activity-per-time	The basic activities (KSs) executed to achieve an i-goal
	plan-activity-map	Sequence of plan-activities: when i-goals will be pursued and partial results will be formed
record	PGPs	The partial global plans the node-plan participates in
	sent-to-nodes	Associates nodes that were sent this node-plan with the creation-time of the sent node-plan
future	start-time	Expected start time of future node-plan
	planned-by	What node predicted the future node-plan
	triggering-PGP	The PGP that triggered the creation of the future node-plan
	basis	The names of the local plans on which the future node-plan is based and their creation-times

Figure 45: The Attributes of a Node-Plan.

information. To represent the long-term activities of the plan, the node-plan also contains the plan's time-order (order of i-goals), time-predictions (expected time needed to complete each i-goal), tracking-levels (how the i-goal results will be integrated), and activity-per-time (the basic types of KSs used to achieve an i-goal).

The planner uses the time-order, time-predictions, and objectives of the plan to generate an *plan-activity-map* that explicitly associates starting and ending times with activities. That is, given a time when the plan is started, the plan-activity-map indicates when the i-goals will be pursued and when different partial solutions will be generated. The plan-activity-map is a sequence of plan-activities, where each plan-activity summarizes the set of detailed actions that together achieve an i-goal. The attributes of a plan-activity are shown in Figure 46. It has

start-time	When the activities to achieve the i-goal are expected to begin
end-time	When the activities to achieve the i-goal are expected to end
node	The node performing the activities to achieve the i-goal
focus-area	The basic specifications (time-regions) of the data processed
expected-result	The expected results combining the new partial result from the focus-area with relevant results from previous plan-activities

Figure 46: The Attributes of a Plan-Activity.

a start-time and an end-time, indicating when the first detailed action begins and when the last action that achieves the i-goal ends. Associated with a plan-activity is the node where the actions to achieve the i-goal take place. The plan-activity's principal area of focus—what information it is processing—is represented (in the vehicle monitoring domain, this focus corresponds to the i-goal's sensed time and the possible regions where data may be developed). Finally, the plan-activity indicates its expected results, combining the results in its area of focus with results formed by previous plan-activities (in the vehicle monitoring domain, the expected result is the partial track formed by combining the new time-regions with previously developed partial tracks).

A node-plan also has attributes to record how it has been used and where (and when) it has been transmitted. When the planner identifies a group of node-plans that together are working toward a partial global goal, it forms a PGP that points to each of these node-plans. The node-plans, in turn, have a record of the PGPs in which they participate. Recorded with a node-plan is also information about where and when it has been transmitted. When a node has a node-plan that another node might find useful, it should send it to that node; to avoid sending it repeatedly to the same nodes, it should record for each node-plan the nodes to which it has been sent. Moreover, because local plans can be modified and these modifications are propagated to the node-plans, some information about the version of the node-plan transmitted to each node should be maintained. Since the creation-times of local plans and node-plans are changed when the plans are altered, the node-plan associates with each node to which it has been sent the node-plan's creation-time. Thus, when a node-plan is modified and its creation-time changed, the planner can identify which nodes have obsolete versions of the node-plan and can respond by transmitting appropriate information to them.

The future attributes of a node-plan are also included in Figure 45. The future

attributes are used when the node-plan is created because a PGP indicates that a node will likely develop a plan in the future: either it predicts that a vehicle will move into that node's sensed area, or it expects that tasks (data to process) will be passed to that node. For a future node-plan, the attributes specify when in the future the local plan is expected to start, what node formed the prediction (since one node may predict that another will have plans in the future), the name of the PGP that triggered the creation of the future node-plan, and the basis for the prediction. This last attribute indicates the names and creation-times for the plans that the PGP used when making the prediction, so that if the future node-plan is transmitted elsewhere the recipient nodes can check to see if the basis for the prediction is still valid considering their possibly more recent versions of these plans.

A node has a node-model for each node, including itself. When a node forms a node-plan locally or receives a node-plan from another node, it inspects the node-plan's node attribute and stores the node-plan in the appropriate node-model. A node-model thus represents one node's view of another, where the completeness and currency of this view depends on what node-plans it has received from the other node and when.

Besides node-plans, a node-model also contains information about a node's more general characteristics (Figure 47). Many of the more general characteristics of nodes are defined as part of the network organizational structure: what resources does the node have (KSs, sensors); what types of partial solutions is the node likely to produce (what spatial regions do its sensors cover); how long does the node take to perform various tasks (how efficient are its KSs); and what are the expected communication delays between nodes. This information is important for determining where nodes should transmit partial results to be combined into overall results, where they should pass tasks to make better use of network resources, and where more data is likely to be sensed based on related partial solutions.

A node-model contains some of this information. The *ave-communication-delays* indicates for each of the other nodes the expected delay a message will incur when sent to or received from the node: when coordinating their interactions, nodes need to know how long it will take to exchange partial solutions. The *expected-integration-costs* indicate the expected amount of time the node needs

node	The node being modeled	
node-plans	A set of node-plans for the node	
general-characteristics	ave-communication-delays	The expected transmission delay between this node and any other node
	expected-integration-costs	The expected time the node needs to integrate two partial solutions
	expected-processing-costs	The expected time the node needs to process sensor data into combinable results
	sensed-regions	Where the node receives sensor data

Figure 47: The Attributes of a Node-Model.

to integrate two partial solutions (in the vehicle monitoring domain, to combine two tracks into a longer track): to effectively plan how to integrate the results of their subproblems, nodes need to identify which of them is best able to perform this integration. The *expected-processing-costs* estimates the expected amount of time a node needs to process (synthesize) sensor data: if one node is overwhelmed with data, then some of this data could be passed to other nodes. Finally, the *sensed-regions* indicates the areas where a node can receive sensor data: partial solutions (tracks) can be used to estimate where future data for related partial solutions (that extend those tracks) will probably arrive, and nodes with sensed regions covering those areas should therefore expect data at some future time.

In this research, we assume that these more general characteristics of nodes are determined when the network is established and initially organized, and that this organization remains stable. Thus, the problem solving responsibilities and capabilities of nodes, the locations and ranges of sensors, and the communication topology linking nodes with sensors and with other nodes are constants that are accessible to the planner. Given an initial, general organization, the purpose of the planning mechanisms is to determine whether and how nodes can cooperate effectively in a specific situation, and for this purpose their models of each other change only when their local plans change.

A node's *network-model* is therefore composed of a set of node-models (one for each node including itself), and a set of partial global plans representing groups of cooperating nodes. Given node-models of themselves and other nodes, the nodes identify any larger partial global goals that they can achieve through cooperation, and a partial global plan is created for each of these PGGs. The

name	A unique name (symbol) pointing at the PGP structure	
participants	A list of the node-plans that participate in this PGP	
node	The node that created this PGP	
nodes	A list of the names of nodes participating in this PGP	
creation-time	When the PGP was created or last updated	
rating	The rating of the PGP	
objective	track	The expected complete track when the pieces from the separate node-plans are combined
	v-event-classes	The vehicle-event-classes (types of vehicles) being tracked
long-term	tracking-levels	The blackboard-level(s) where hypotheses will be integrated into tracks
	plan-activity-map	Sequence of plan-activities: when i-goals will be pursued and partial results will be formed
	solution-construction-graph	What partial results will be integrated and indication where and when

Figure 48: A Subset of the Attributes of a PGP.

PGG attributes are part of the PGP: just as local plans and node-plans have information about their objectives (goals), a PGP also has an objective, where its objective is to achieve a PGG. A PGP also has information about how the PGG can be achieved. In this chapter, our concern is building network-models that the planner uses to identify PGGs, so we only need consider a subset of a PGP's attributes (Figure 48). The other attributes of a PGP are discussed in the next chapter.

A PGP has a name, and attributes indicating what node formed the PGP and when it was created or last updated. The PGP also points to the node-plans that are pursuing parts of the overall PGG of the PGP, and what nodes are participating in achieving the PGG. The attributes of the PGG are represented as the PGP's objective information: the overall track that can be generated by combining the results of the individual node-plans, and the types of vehicles being tracked by the PGP. Because future node-plans are often based on the long-term information of a PGP, several of these attributes are briefly described here; a more complete description is given in the next chapter. Among the long-term attributes of the PGP are the tracking-levels where the node-plans will build partial solutions, the plan-activity-map that interleaves the plan-activities of the participating node-plans, and the solution-construction-graph that indicates where and when the separate partial solutions generated by the nodes will be

integrated into the overall solution.

The nodes not only need to coordinate their problem solving, they also need to coordinate how they coordinate. Recognizing PGGs and building PGPs is itself a distributed problem solving task—a "meta-task" since solving the coordination problem influences how the domain problem is solved. Thus, although nodes could exchange node-plans indiscriminately and individually recognize partial global goals and plans, they can more effectively solve the problem of how to coordinate if they are organized. For example, instead of having each node figure out how the network should be cooperating (so that the nodes are duplicating much of each other's reasoning), the nodes may be organized so that only some of them are responsible for determining whether and how nodes should cooperate. These nodes should in turn have authority over the remaining nodes so that their decisions about how to cooperate will be followed—but all nodes should maintain some autonomy to respond to unexpected changes in their local situations.

To organize coordination activity, each node is provided with a copy of the *meta-level organization*, whose attributes are shown in Figure 49. The *node-plan-communication-alist* associates with each node the nodes that it should send node-plans to. Similarly, the *PGP-communication-alist* associates with each node the nodes that it should send PGPs.[1] To influence how nodes respond to received information, the meta-level organization specifies how much credibility a node gives to it. Authority is implemented in this way. For example, if $node_1$ gives low credibility to node-plans it gets from $node_2$, then PGPs that involve those node-plans will be less highly rated. On the other hand, if $node_1$ gives high credibility to node-plans from $node_2$, then it will give higher ratings to PGPs that involve them. Because a node pursues its most highly-rated PGPs, a node that has more credibility also has more influence (authority) on the activities of others. The *node-plan-credibility-factors* specify for each node the credibility that node has in node-plans from each of the other nodes: the received node-plan's rating is multiplied by that factor. Similarly, the *PGP-credibility-factors* specify the same information for received PGPs. Finally, the *coordination-responsibilities* indicate for each node what nodes it is responsible for coordinating. Usually, it is

[1] In the future, we hope to extend the meta-level organization to allow the user to specify the characteristics of node-plans and PGPs that should be sent to a particular node so that communication about this information can be more selective.

node-plan-communication-alist	For each node, what nodes it should send node-plans
PGP-communication-alist	For each node, what nodes it should send PGPs
node-plan-credibility-factors	The credibility a node has in received node-plans
PGP-credibility-factors	The credibility a node has in received PGPs
coordination-responsibilities	What nodes each node should coordinate

Figure 49: The Attributes of the Meta-Level Organization.

responsible for coordinating itself, but it may be responsible for coordinating several nodes—it may be their "leader." Coordination responsibilities are described further in the next chapter.

6.3.2 Initializing Node-Models

Each node has a node-model for itself and each of the other nodes. The node-models are initially empty. When the node generates node-plans from its local plans, it inserts these into its model of itself. Similarly, when it receives node-plans from another node, it inserts these into its node-model of that node. Because nodes' plans and goals can change unpredictably over time, maintaining suitable node-models requires that nodes communicate about their node-plans to keep each other up to date.

The more general characteristics of a node are found in the organizational information when they are needed. To develop a sensible organization, any organization designer would have information about the nodes' communication topology, their capabilities for processing data and for integrating partial solutions, and what areas their sensors cover. Thus, as part of its static (or infrequently changing) domain-level organization, a node has this information, and nodes need not communicate about these attributes.

6.3.3 Building Node-plans

A node gets node-plans from three different sources: the planner may build them from local plans; the node may receive them from another node; or the planner may use PGPs to predict that a node will have a local plan at some future time. Moreover, to explicitly represent a lack of any local plans, a node may form an "idle" node-plan, which indicates that the node has no planned activities. Idle node-plans are useful when the nodes attempt to move tasks to balance their

loads: if one node knows that another is idle, then it can safely pass tasks to it; without explicit idle node-plans, a node cannot be sure whether another node is idle or if it simply has not communicated about its plans.

Three ways of building node-plans—from local plans, from PGPs, and from a lack of local plans—are described in this section. The mechanisms for building node-plans from received information are described in the next section.

Building Node-plans from Local Plans

Given a freshly created or modified local plan, the planner updates its node-model of itself by creating or modifying a node-plan for that local plan. It begins by first identifying whether a node-plan already exists for an older version of the local plan: it scans the node-plans in its node-model of itself to see if any of their plan-name attributes match the name of the local plan. If one is not found then a new node-plan structure is created to represent the local plan. The new or existing node-plan is then given some attributes directly from the local plan: plan-name, creation-time, objectives, and most of the long-term plan information. The node-plan's node is simply the name of the node that has this local plan (which is the current node). To compute the node-plan's rating, the planner takes the rating of the local plan and multiplies it by the node's credibility in its own node-plans (which it finds in the meta-level organization). Thus, a node's credibility in its own plans (and therefore its authority over itself) is determined by the meta-level organization.

Because the node-plan's plan-activity-map associates specific intervals of time with the plan's i-goals, the planner cannot compute this information until it knows when it will start working on the plan, which in turn depends on the other plans that it could work on. This attribute is therefore set later when the planner decides what local plan it will pursue next. However, if the node-plan is being modified because its local plan has been changed, then it may already have a plan-activity-map. In this case, the planner deletes from the plan-activity-map any plan-activities for the future. Since the local plan may have been pursued for a while and some i-goals completed, the plan-activities for those completed i-goals represent actual past activities, not potential future activities, and so should be permanently associated with the node-plan.

The attributes specifying what PGPs use the node-plan and where it has been

sent are set later by the planner. Because the node-plan is based on an existing local plan, its future attributes are empty.

Building Future Node-Plans

A future node-plan is a prediction about future activities that, as yet, no local plan represents. By examining a PGP's objectives and planned activities, the planner recognizes that a node currently not working on a part of the PGP may do so in the future. A future node-plan for a node is generated for any of three reasons:

- the objective track of a PGP indicates that the vehicle being tracked will likely move into the sensed area of the node, so that the node will begin receiving sensor data at some future time;
- the planned activities for integrating partial results (in the PGP's solution-construction-graph) indicate that the node will receive partial results (tracks) to combine at some future time;
- the planner has represented in the PGP a possible transfer of data (tasks), where some other node may pass data (tasks) to this node so that its computational capabilities or expertise are better used.

In the first case, the planner takes the current objective track of the PGP and uses expectations about how vehicles move to predict its likely future course. If this course passes through a node's sensed area, then a future node-plan can be hypothesized for that node. The future start-time for that node-plan would be when the new sensor data is expected to start arriving. The node-plan's objectives correspond to the expected course of the vehicle through the node's area, its long-term information and use are left empty, its future planned-by attribute is the name of the node that is building the node-plan, its future triggering-PGP is the name of the PGP, and its future basis is a list of the plan-names for the participating node-plans. Note that any node with the PGP could form the future node-plan for the node -a node can hypothesize a future node-plan for another node and save that node-plan in its model of the other node.

In the second case, the planner builds a future node-plan if a PGP indicates that certain integration tasks will take place at a particular node in the future.

For example, two nodes that build partial solutions for a larger problem need to combine these somehow. One of them could pass its partial solution to the other, which would then combine them. Alternatively, they could each recognize that a third node has expertise or available computing power that makes it the best node to integrate the pieces, so both nodes pass their partial solutions to the third. A node with this PGP can build a future node-plan, specifying that the third node will have these integration activities in the future, and this node-plan is stored in the third node's node-model. By using the general characteristics about the nodes involved, the planner can estimate when the partial solutions will reach the third node, and so, can approximate the start-time for the future node-plan. The node-plan's objectives are those of the combined partial solutions, its long-term and use information are empty, and its future information is set as in the first case.

In the third case, the planner has identified that some tasks --some i-goals to achieve--could potentially be transferred from one node to another node. The proposed recipient node would be more capable of performing the tasks: it may be idle or performing unimportant activities, or it may have expertise for performing these tasks. When the planner recognizes such a situation, it can modify a PGP so that the PGP represents the recipient node performing the tasks, and then it can pass the PGP on to the recipient node. The recipient node builds a future node-plan indicating that it may perform the tasks in the future. However, when it builds this node-plan it considers its other node-plans. The node may have a different view of its plans and goals than the node that formed the PGP, and the future node-plan reflects its view. The future node-plan's objective is built from the potentially passed tasks (i-goals), its long-term plan-activity-map is constructed to indicate when the planner expects that the node can perform these tasks, and its future information is set as in the first two cases. By transmitting this node-plan back, the node-plan acts as a response to the PGP: the PGP represents a request for information about how the node predicts it could perform the task, and the node-plan represents this information. If the information agrees with the proposed transfer, the tasks may be passed, and this process is covered in more detail in the next chapter.

Future node-plans are only constructed when they could affect current plans, so if the planner can coordinate the nodes based solely on its current information,

then it need not hypothesize future node-plans. The planner needs to predict future node-plans when it is deciding whether it should transfer tasks (i-goals) or partial solutions for integration because it should not transfer tasks that may interfere with activities that the node will need to perform in the future. Future node-plans are therefore most important when the transfer of tasks is being considered, and to reduce control overhead in the current implementation are not generated otherwise.

Building Idle Node-Plans

When a node has no local plans, either because it never had any or because those that it had have been completed or aborted, then the node should model this inactivity in an idle node-plan. An idle node-plan has no objectives, and no long-term, use, or future information. It's plan-name is nil and its rating is 0. It does have a creation-time (when the node became idle) and the name of the node that is idle. The idle node-plan is locally incorporated in the node's model of itself, and can be transmitted to other nodes as well. When a node knows that another is idle, then its planner can develop PGPs that represent passing tasks to that node to make better use of its computational resources. However, when a node that was idle is no longer idle, it must inform other nodes of this fact so that they do not hypothesize further task-passing. Because node-plans change over time, nodes must maintain their node-models to reflect as accurately as possible the current and predicted plans and goals of other nodes. This topic is described after the methods for communicating about node-plans are discussed.

6.3.4 Meta-Level Communication About Node-plans

Each node is responsible for communicating about node-plans so that the nodes responsible for recognizing cooperative opportunities and for responding to these opportunities are adequately informed about other nodes' plans and goals. Because they are exchanging meta-level information about how they are solving the problem instead of domain-level information about the problem itself, this exchange of information is called meta-level communication. The meta-level organization specifies the coordination responsibilities of the nodes and the meta-level communication patterns that are the basis for deciding where node-plans should

be sent and received. For example, if the network is organized so that a single node acts as coordinator, then each of the other nodes should direct their node-plans only to that node. Alternatively, if each node is responsible for coordinating itself, then nodes should broadcast their node-plans so that each can individually recognize when and how it should cooperate with others.

Meta-level communication also involves communicating about PGPs. Because the focus of this chapter is on building node-models and identifying PGGs, the discussion of how PGPs are communicated is delayed until the next chapter where the attributes and reasons for communicating PGPs are more fully explained. However, since node-plans can be transmitted in response to received PGPs, it is important to mention at this point that PGPs can be communicated.

Transmitting Node-Plans

A node transmits a node-plan for one of four reasons: to inform other nodes of a new or changed local plan of this node; to relay node-plans that it received on to other nodes; to inform other nodes of idle node-plans of this node, or to respond to coordination information (a PGP) that it received from another node. When determining which of its *local node-plans* (node-plans in its node-model of itself) to transmit, the node first must decide which of its node-plans are relevant: since it is unlikely to pursue its more lowly-rated plans, the node may safely ignore them when trying to coordinate with others. The first step in choosing local node-plans to send is thus identifying a set of highly-rated node-plans (where the width of the window between highest and lowest rated of these is a user supplied parameter *node-plan-communication-window). The planner then retrieves from the meta-level organization the recipient nodes for the node-plans. For each combination of node-plans and recipient nodes, the planner determines whether the node-plan has changed since it was last sent to that node, if it has been sent before. This determination is made by checking the node-plan's current creation-time against the sent-to-nodes information for the recipient node. The node-plan is sent to any recipient nodes that do not have a current version of the node-plan, and the node-plan's sent-to-nodes information is updated.

The planner also checks other nodes' node-models to see if any of their node-plans should be forwarded on to other nodes. Because the communication topology might not allow nodes to communicate directly, a node might receive a node-

plan that, according to the meta-level organization, should be passed on to other nodes. To identify such node-plans, the planner scans through its models of other nodes and determines for each node-plan what nodes should know about that node-plan (by using the node attribute of the node-plan and the meta-level organization). When it discovers, according to the node-plan's sent-to-nodes information, that the most recent version of the node-plan has not been transmitted to a node that should know about it, then the planner transmits the node-plan to that node.

The third reason for sending a node-plan is to inform other nodes about idle node-plans. When nodes are capable of transferring tasks among themselves, it is important that they know which nodes are busy and which are not. Thus, whenever a node becomes idle, it should send an idle node-plan to whatever nodes could use it: the meta-level organization specifies the nodes that are responsible for coordinating the node, where these responsibilities include the transfer of tasks when needed.

The fourth reason for sending a node-plan is as a response to a received PGP that was requesting information about whether and when the node could perform transferred tasks. By responding to the received PGP, the node provides the other node with information that will help it decide where to send the tasks. As previously described, when a node receives a PGP requesting such information, the node builds a future node-plan that indicates when (and whether) the node expects that it could pursue the transferred tasks. These activities are discussed in more detail in the next chapter, Section 7.3.8.

To transmit a node-plan, the planner extracts the attributes of the node-plan and builds a node-plan message with this information. The message has a predefined structure so that the recipient node knows how to extract the attributes out again to create a local copy of the node-plan.

In the current implementation, future node-plans are only transmitted in response to requests for information: only when the node receives a PGP indicating a potential transfer of tasks does it transmit back a future node-plan representing its view of how it could pursue those tasks. Future node-plans can be generated for other reasons. Specifically, the planner can build expectations about data the node will sense or integration tasks it will perform in the future based on a PGP. The planner builds these predictions when it must decide whether to transfer cur-

rent tasks (activities that could be pursued immediately) to the node—to make an informed decision, it must determine whether the transferred tasks are likely to interfere with potential future tasks at the node. However, since any node with the PGP can recognize such future node-plans for a node, these node-plans are not transmitted. In addition, because future node-plans may never materialize (the expected events may not occur), the planner must inspect future node-plans regularly to determine whether they are still valid: future node-plans represent assumptions that may have to be retracted later. By having nodes form future node-plans locally only when they are needed and by not exchanging them, the number of future node-plans in a node's node-models is minimized so the costs of keeping track of them and retracting them is minimized (see Section 6.3.5).

For example, consider the three nodes with adjacent sensed areas depicted in Figure 50. $Node_1$ has received data indicating that a vehicle passed through its area, $node_2$ has no data, and $node_3$ has data indicating that a vehicle is passing through its area and will probably move into $node_2$'s area next (shown by the dotted line extending the track). Given this situation where $node_2$ is initially idle (and sends idle node-plans to the other nodes), $node_1$ and $node_3$ should recognize that they should *not* send tasks to $node_2$ because it will shortly receive data of its own. $Node_3$ has sufficient local information to predict $node_2$'s future node-plan and will not transfer tasks to $node_2$. If $node_3$ does not communicate with the others about its node-plans (based on the meta-level organization), however, then $node_1$ will have no reason not to transfer some of its i-goals to $node_2$, and these transferred tasks will later interfere with $node_2$'s locally generated tasks. If $node_3$ only communicates with $node_2$, then $node_2$ will hypothesize its future node-plan. To prevent $node_1$ from transferring tasks to it, $node_2$ could relay $node_3$'s node-plans (or pass PGPs that include those node-plans) on to $node_1$ so that $node_1$ can identify the future node-plan itself. Alternatively, $node_2$ could transmit the future node-plan directly. Finally, $node_2$ can wait for $node_1$ to request information about whether it can work on transferred tasks, and respond with a future node-plan that indicates it will be unable to do these tasks because it expects local tasks to arrive.

If $node_2$ transmits its future node-plan, then every node specified in the meta-level organization as a recipient of the future node-plan will receive it: if these nodes are part of a larger network, many nodes that have no use for this node-plan

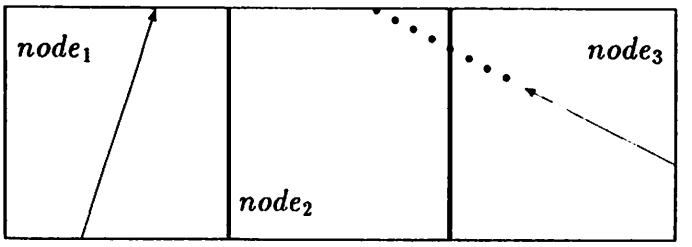


Figure 50: An Example Involving Future Plans.

could also receive it. Since maintaining node-models is more difficult with future node-plans, it is preferable to use the other mechanisms. Enough information should be exchanged so that $node_1$ should be able to identify the future node-plan based on its own PGP, and just in case the nodes have inconsistent PGPs, $node_1$ should request information from $node_2$ before any tasks are passed.

Receiving Node-Plans

Before it can process a received node-plan message, the planner must first determine whether the node-plan was transmitted in response to a particular PGP or whether its purpose is to update node-models to better identify opportunities for cooperation. If the node-plan message has no triggering-PGP attribute then it is not a response to a PGP, and the planner should introduce the received information into the appropriate node-model. First, it determines what node-model to update by inspecting the node-plan message to find out which node has the summarized plan. Next, it determines whether it already has some version of the received node-plan in that node-model. Because the recipient node builds its own node-plan structures for received node-plans and inserts these into the node-model, node-plans corresponding to the same local plan will be named differently at each node. However, the plan-name attributes of these node-plans will be the same (although nodes that receive the node-plan do not have the local plan structure corresponding to that name). An existing node-plan matches the node-plan message if they have the same plan-name. Also, because plans may be modified and merged, an existing node-plan matches the node-plan message if it is subsumed by the node-plan message: if the objectives of the newly received node-plan are consistent with (and possibly more extensive than) the existing node-plan.

A local version of the received node-plan is either retrieved from the appropriate node-model (if a matching node-plan exists) or a new node-plan structure is created and inserted into the node-model. The attributes of the node-plan message are then used to update the local version of the node-plan. First, the creation-time of the newly received node-plan is compared with the creation-time of the local version, and the local version is only updated if its creation-time is empty (the version was just created) or is less than the creation-time of the node-plan whose attributes are contained in the message. That is, it is possible that this node locally has more recent information about the plan summarized in the node-plan message (it received a more current message from somewhere else), and thus should not update its version of the node-plan with less current information. When the information contained in the message is more current than the existing node-plan's information, the attributes of the node-plan are extracted from the message and replace the existing node-plan's attributes.

The only attribute of this received node-plan that the receiving node alters is its rating: the node multiplies the received rating and the credibility-factor for the transmitting-node from the meta-level organization. Thus, a received node-plan from a low-credibility node has its rating lowered, and a received node-plan from a high-credibility node has its rating raised. Because nodes pursue highly-rated PGPs, where a PGP's rating is based on the participating node-plans' ratings, node-plans from high-credibility nodes are more apt to affect a node's decisions, and thus high-credibility nodes exert authority. In addition, note that a node-plan that is received, incorporated, and later relayed on to other nodes may be passed on with a different rating than it had at the originating node: a node passes on not what it was told (the message it received) but what it believes (the node-plan it locally added). The meta-level organization determines how node-plans will be relayed, and thus can be used to have nodes pass on skewed views of others to influence other nodes' behavior.

When the received node-plan has a triggering-PGP, then it is treated as a response to a transmitted PGP. In this case, the planner retrieves the PGP and stores the message in one of the PGP's fields. When the planner must decide whether the transfer of tasks hypothesized in the PGP should be carried out, it examines the messages stored with the PGP. These messages are responses to the PGP, and the planner uses them to determine where (and whether) tasks should

be transferred.

6.3.5 Maintaining Node-Models

Nodes communicate about node-plans to build models of each other, but because their problem situations can change over time these models must be updated when needed. For example, when a node that has been working on one plan shifts to another, then it should send out node-plans that reflect this change: not only information about the node-plan that it is now working on, but also information about the node-plan that it has shifted away from. Since another node may have been expecting to coordinate with the original plan, it is important that this node be told when work on the original plan ceased and what parts of the plan had been completed. The information about when the plan ceased comes from the node-plan's creation-time, which is updated when this plan is no longer the one being followed, and the information about when the plan ceased comes from the node-plan's plan-activity-map, which contains plan-activities for all i-goals achieved in the past. If it knows what parts of the plan have been completed, the recipient node can better determine what parts it should work on or, if it expects that the node will resume work on the plan (because it believes that the node will receive network information that will make the plan more attractive), the recipient node can hypothesize when future activities will take place because it knows what parts have already been accomplished.

Node-models therefore represent possibly incomplete, inconsistent, and out-of-date views of other nodes, and PGPs based on node-models are thus tentative. One node may have a model of another that indicates that that node is idle. If it later receives a node-plan indicating that that node is active (is pursuing actions), then it should delete the idle node-plan from its model of that node. Similarly, if it has a future node-plan as part of its model of another node, then the node must keep track of that node-plan and delete it either when it receives the node-plan from the other node (indicating that the future node-plan is now a node-plan based on a current local plan) or when sufficient time has elapsed that it should have gotten such a node-plan. In the latter case, the predicted future node-plan never materialized, and the node must respond to the failed assumption: it might change how it interacts with that node. If a node aborts a node-plan, then other nodes' node-models of this node must be updated accordingly: the node-

plan should be removed from the node-model and any PGGs that the node-plan participates in must be adjusted accordingly.

Because node-models represent assumptions about other nodes' plans and goals, maintaining node-models is like maintaining any assumption-based information: sometimes assumptions are retracted and the effects of changed assumptions must be propagated. Because assumptions about a node's behavior are contained in any number of node-models distributed among nodes in the network, maintaining these models (propagating changes) is a complicated process. To simplify things in this research, nodes are not permitted to exchange especially volatile information (such as future node-plans based on possible extensions to PGPs' tracks), so that they only need communicate about what they are doing and have done in the past, not what they might do.

The other simplification we make is that we allow nodes to transmit all of their updating information. Because these messages incur communication delays and because nodes only communicate about their most highly-rated node-plans, nodes still may have inconsistent, incomplete and out-of-date information. However, because we assume that communication between them allows useful update information to eventually reach nodes that need it, the planner does not need mechanisms to infer likely changes to other nodes' behavior. Making such inferences can be difficult and very costly, since one node is trying to model the problem solving of others. The planner currently avoids these problems by assuming that communication channels can handle the meta-level communication needed by the nodes to maintain each other's node-models and coordinate their activity. In Chapter 9, one of the future research directions discussed covers these issues further.

6.3.6 Recognizing Partial Global Goals

Through meta-level communication, a node's planner builds an awareness not only of its own current plans and goals, but also of other nodes' current plans and goals. By recognizing how the goals of different nodes can be part of a larger partial global goal, the planner identifies opportunities for cooperation—it finds PGGs. For each PGG it builds a PGP that represents the objectives of the cooperation and, as will be described in the next chapter, will eventually indicate how these objectives might be achieved.

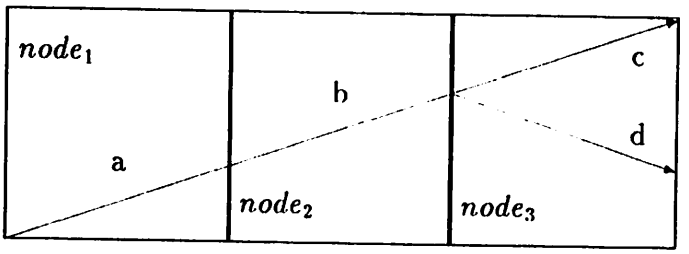


Figure 51: An Example Involving Alternative Combinations of Plans.

The process of recognizing PGGs can be costly both in terms of computation and storage. The computation costs can be high because the planner could compare many possible combinations of node-plans' objectives to identify all the possible PGGs. The storage costs can be high because the planner may have to save many *partial-combinations* of objectives as well. For example, in Figure 51 there are two PGGs, each represented as a sequences of track sections: *abc* and *abd*. When the planner constructs PGGs, it cannot simply merge new pieces into an existing PGG: if it merges *c* with the PGG it has so far constructed for *ab*, then it cannot merge *d* into the result and must rederive *ab*. Instead, the planner makes a copy of the PGG *ab* before merging it with *c* so that it can later merge it with *d* as well. Because copies of partial-combinations are saved as PGGs are formed, generating PGGs can be expensive in storage.

The process of identifying PGGs and representing each as a PGP is as follows. For each new or modified node-plan in any of its node-models, the planner generates a partial-combination whose objectives are the same as the node-plan's. Moreover, if a modified node-plan represents an aborted plan, then the planner finds all the PGGs that this node-plan participates in and treats each of the other participants of these PGGs as modified node-plans. This allows the planner to find alternative PGGs for them that do not include the aborted node-plan.

Because PGGs that only involve unmodified node-plans need not be changed, the planner only looks for PGGs involving the new and modified node-plans. It uses each of the partial-combinations from these node-plans as a starting point. For each, it compares the partial-combination with the node-plans in the node-models, and forms a new partial-combination when a node-plan can be combined with the partial-combination. A node-plan can be combined with a partial-combination if: it is potentially tracking vehicles of the same type as the other participants in the combination (the node-plan's vehicle-event-classes

intersect with each of the participants' vehicle-event-classes), and its objective track can be combined with the partial-combination's objective track. Tracks can be combined if they meet the vehicle movement constraints that the clustering mechanisms use—if the same vehicle could be responsible for both tracks. When these criteria are met, the node-plan information is combined with the partial-combination's information into a new partial-combination: the planner combines the vehicle types as the union of the node-plan's and the old partial-combination's, and it combines the tracks together with the same functions it uses when it combines tracks of alternative-goals together (Chapter 4). In addition, the partial-combination records what node-plans participate in it.

Whenever a new partial-combination is formed, the planner adds it to the list of starting points and the process continues until every node-plan has been checked against the partial-combinations. At this point, subsumed partial-combinations are removed. For example, each of the 4 track sections in Figure 51 represents a separate node-plan (where $node_3$ has two node-plans). At the end of forming combinations, the partial-combinations are *a*, *b*, *c*, *d*, *ab*, *bc*, *bd*, *abc*, and *abd*. The planner removes any subsumed partial-combinations, leaving *abc* and *abd*. These are then compared with any existing PGPs, and if any represent modifications to a PGP the PGP is modified appropriately (its objectives are set to those of the new partial-combination). If any partial-combinations are not represented by existing PGPs, a new PGP is formed whose objectives are set to those of the new partial-combination, and whose participants are also retrieved from the partial-combination.

Because it only uses new and modified node-plans as starting points, the planner can avoid generating all possible combinations each time node-plans change. In Figure 51, for example, if the node-plan to build track section *d* is later modified (possibly extended with new data), then the planner generates partial-combinations *d*, *bd*, and *abd* only (since the PGGs involving *c* will be unaffected), and of these then *abd* will be used to modify the existing PGG. More examples are provided in Section 6.3.8.

The process of recognizing PGGs can be costly because, in the worst case, the planner may need to consider every possible combination of node-plans. This is why it is important that nodes restrict meta-level communication to only exchanging highly-rated node-plans: not only is communication resource usage re-

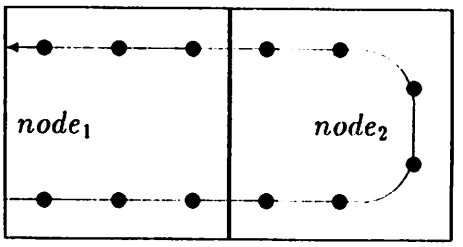


Figure 52: Two Local Plans Part of the Same PGP.

duced, but also the number of node-plan combinations at the recipient nodes is decreased. The possible drawback to restricting communication to only the best node-plans is that some combination of less highly-rated node-plans might together form a highly-rated PGP. Thus, the time needs of the planner are reduced at the potential cost of making worse decisions. The user supplied parameter *node-plan-communication-window can be adjusted to strike an effective balance between minimizing costs and minimizing the risk of missing useful information.

Even when meta-level communication is restricted, there are potentially a large number of combinations. If we assume that node-plans in the same node-model need not be compared (because if they were part of a larger goal the local plans would be merged locally), then if there are n node-models with p node-plans each, the number of possible combinations of any length other than 0 is $|(1+p)^n - 1|$, or on the order of p^n. For a large number of nodes and node-plans, it can get extremely expensive to compute these combinations. Moreover, the assumption that node-plans in the same node-model need not be compared is not true: for example, consider the situation in Figure 52 where a single node has two distinct pieces of a larger goal. When all possible combinations of 1 or more of all $p \times n$ node-plans must be considered, there are $|2^{p \times n} - 1|$ such combinations possible.

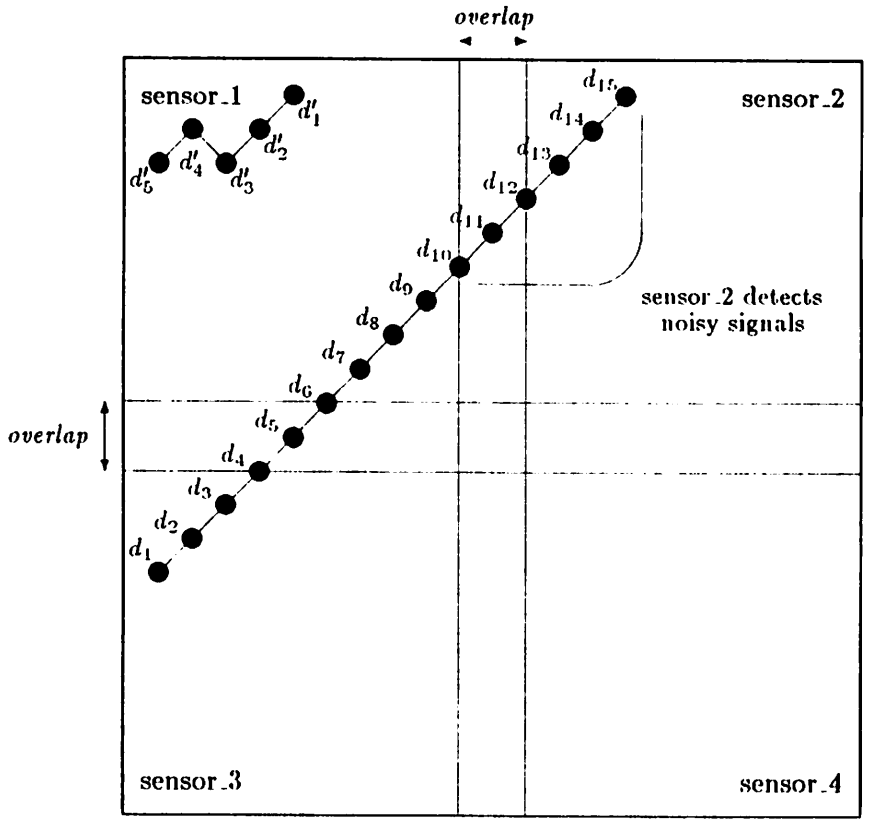
In the current implementation, the planner checks all possible combinations at least once, but as new node-plans arrive and existing node-plans are modified, the planner uses these as starting points to avoid checking combinations that could not possibly be changed. By keeping track of what has changed and what needs to be checked, the planner can reduce the computation and storage demands. If the number of node-models and amount of meta-level communication increases, however, other methods for reducing the computational complexity will be needed. To get a full view of the entire network, the planner finds all possible

PGGs—even those that the current node does not participate in, because the planner can make better decisions if it knows that a node to cooperate with is also likely to cooperate with other nodes. By only finding the PGGs that the node can participate in, the maximum possible combinations with one node-plan per node is $\lfloor p(p^n - 1)/(p - 1) - 1 \rfloor$ (which is still order p^n) and the maximum possible combinations of all node-plans is $\lfloor 2^{p \times n}/2^n - 1 \rfloor$ (which is still exponential). Another way of reducing the number of combinations to consider is to restrict the set of nodes that the planner searches to find PGGs. By specifying in the meta-level organization that only some nodes send node-plans to the node, the number of combinations it needs to consider can be reduced. The last way to reduce the number of combinations is to represent the combined activities of a group of nodes in a single "node-plan", and this is discussed as a future research direction in Chapter 9.

6.3.7 Network-Models

A node's network-model thus has two principal components: a set of node-models and a set of PGPs. The planner develops the node-models from local plans and from received messages about other node's node-plans. Nodes communicate to keep their node-models up-to-date. The planner develops PGPs by recognizing groups of node-plans that are working toward a single, larger goal. Because nodes can also exchange PGPs, they communicate to keep their views of cooperation among nodes consistent.

The meta-level organization specifies whether nodes will communicate about node-plans, PGPs, both, or neither. The patterns of communication determine how the nodes will coordinate their coordination activities: they may channel node-plans to a single "coordinator" which builds and distributes PGPs for the entire network; they may broadcast node-plans to each other and never exchange PGPs because they individually form their own; or they may exchange node-plans and PGPs, because their somewhat different models of each other may lead to inconsistent PGPs and by exchanging PGPs the nodes can converge on a consistent view of cooperation. The style in which they cooperate therefore depends on how they form and respond to their network-models, and thus depends on the meta-level organization.



The four overlapping sensors detect signal data at particular locations for discrete sensed times (the dots with associated times). Sensor 2 is faulty and not only generates signal data at the correct frequencies for each location but also detects noisy signals at spurious frequencies for each location. Node i is connected to sensor i.

Figure 53: Four-Sensor Configuration with Sensed Data.

6.3.8 Examples of Finding Partial Global Goals

To illustrate the process of finding PGGs, consider the problem environment of Figure 53. There are four overlapping sensors, and each is assigned to a different node. $Sensor_2$ is faulty and generates data at correct frequencies but also forms noisy signals at spurious frequencies. $Node_1$ receives data d'_1–d'_5 and d_4–d_{12}, $node_2$ gets data d_{10}–d_{15}, $node_3$ gets data d_1–d_6, and $node_4$ gets no data at all. Each node uses its data to form local plans. Each then summarizes its plans into node-plans, and then forms PGPs based on its current node-models (which initially just contain information about its own plans).

Node	Model	Entry	Participants	Track	V-types	Sent?
1	1	np_{1a}	-	d_1'-d_5'	(1)	yes
		np_{1b}	-	d_1' d_5'	(2)	no
		np_{1c}	-	d_4-d_{12}	(1)	yes
		np_{1d}	-	d_4-d_{12}	(2)	no
	All	PGP_{1a}	np_{1a}	d_1'-d_5'	(1)	no
		PGP_{1b}	np_{1b}	d_1'-d_5'	(2)	no
		PGP_{1c}	np_{1c}	d_4-d_{12}	(1)	no
		PGP_{1d}	np_{1d}	d_4-d_{12}	(2)	no
2	2	np_{2a}	-	d_{10}-d_{15}	(1 2 3 4 5)	yes
	All	PGP_{2a}	np_{2a}	d_{10}-d_{15}	(1 2 3 4 5)	no
3	3	np_{3a}	-	d_1-d_6	(1)	yes
		np_{3b}	-	d_1-d_6	(2)	no
	All	PGP_{3a}	np_{3a}	d_1-d_6	(1)	no
		PGP_{3b}	np_{3b}	d_1-d_6	(2)	no
4	4	np_{4a}	-	nil	nil	yes

Figure 54: Example Initial Contents of Nodes' Network-Models

Assume that the data arrives at sequential node times, so that problem solving does not begin until after all of the data has arrived. When the nodes form local plans, therefore, they have all of their local data available. In Figure 54 are shown the initial network-models for each node: each node has models of the individual nodes (although if the model for a node is empty it is not shown in the figure) and of all nodes (its PGPs). $Node_1$ has formed four local plans, two for d_1'-d_5' (with different vehicle types) and two for d_4-d_{12} (with different vehicle types). Each of these local plans is represented by a node-plan in its node-model of itself, and each serves as the basis for a PGP. $Node_2$ has formed a single local plan for d_{10}-d_{15}, where this plan encompasses several vehicle types: because the spurious frequency data formed by $sensor_2$ corresponds to several types of vehicles, $node_2$ is unable to narrow down the possible vehicle types. This local plan is summarized in $node_2$'s node-model of itself, and a PGP is formed from this. $Node_3$ has formed two local plans for data d_1-d_6, each with a different vehicle type. Each of these is summarized by a node-plan in its node-model of itself, and these node-plans are used to form PGPs. Finally, $node_4$ has no local plans, and forms an idle node-plan that it stores in its node-model of itself.

Nodes exchange their most highly-rated node-plans to build up models of each other, and each of the node-plans sent to other nodes are indicated in Figure 54.

When a node receives node-plans from others, it incorporates them into the appropriate node-models and modifies the PGPs. The situation where nodes have exchanged highly-rated node-plans is shown in Figure 55. Each node has some information about the others and can build up more global PGPs, but nodes do not necessarily have identical network-models since they do not communicate every node-plan. $Node_1$ builds node-models for the other nodes, and recognizes more global PGPs: PGP_{1c} where nodes 1, 2, and 3 all participate in forming the complete track d_1–d_{15} (since the participating node-plan's tracks are compatible and they share the objective of tracking a vehicle of type 1); and PGP_{1d} where nodes 1 and 2 participate in forming the track d_4–d_{15} (because $node_3$ does not send its node-plan for vehicle type 2, the track does not extend into its area). $Node_2$ builds node-models for the other nodes and recognizes PGP_{2a} where nodes 1, 2, and 3 participate in forming the track d_1–d_{15} (since they all share node-plans with vehicle type 1). $Node_3$ and $node_4$ similarly recognize the same PGP.

The nodes initially had different network-models, and even after meta-level communication about their node-plans their network-models are not identical because they selectively exchange their most highly-rated node-plans. However, the nodes all recognize two highly-rated PGPs: to generate a track covering d_1–d_{15} where nodes 1, 2, and 3 participate; and to generate a track covering d'_1–d'_5 where node 1 is the lone participant.

Note that the common PGP covering d_1–d_{15} has numerous vehicle types. Even though two of its participant node-plans are of vehicle type 1, one of its participant node-plans is for several vehicle types including type 1. When $node_2$ receives node-plans from 1 and 3 (which it locally represents as np_{2c} and np_{2d}), it could divide its local plan into two—one for vehicle type 1 and the other for the remaining vehicle types. In the current implementation, it does *not* divide the plan based on received meta-level information; it must wait to receive domain-level messages (hypotheses) that provide concrete, predictive information. Forcing nodes to exchange predictive domain-level information allows us to emphasize and explore issues in planning cooperative activities: one node must plan actions to generate results that it can send to another node to help that node better solve its problems.

As problem solving proceeds, $node_1$ forms and transmits a predictive partial solution to $node_2$: using the mechanisms described in the next section, it forms and sends a track covering d_8–d_9 with vehicle type 1. $Node_2$ uses this data to

Node	Model	Entry	Participants	Track	V-types	Sent?
1	1	np_{1a}	-	d'_1-d'_5	(1)	yes
		np_{1b}	-	d'_1-d'_5	(2)	no
		np_{1c}	-	d_4-d_{12}	(1)	yes
		np_{1d}	-	d_4-d_{12}	(2)	no
	2	np_{1e}	-	d_{10}-d_{15}	(1 2 3 4 5)	yes
	3	np_{1f}	-	d_1-d_6	(1)	yes
	4	np_{1g}	-	nil	nil	yes
	All	PGP_{1a}	np_{1a}	d'_1-d'_5	(1)	no
		PGP_{1b}	np_{1b}	d'_1-d'_5	(2)	no
		PGP_{1c}	np_{1c},np_{1e},np_{1f}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{1d}	np_{1d},np_{1e}	d_4-d_{15}	(1 2 3 4 5)	no
2	2	np_{2a}	-	d_{10}-d_{15}	(1 2 3 4 5)	yes
	1	np_{2b}	-	d'_1-d'_5	(1)	yes
		np_{2c}	-	d_4-d_{12}	(1)	yes
	3	np_{2d}	-	d_1-d_6	(1)	yes
	4	np_{2e}	-	nil	nil	yes
	All	PGP_{2a}	np_{2a},np_{2c},np_{2d}	d_1-d_{15}	(1 2 3 4 5)	no
	All	PGP_{2b}	np_{2b}	d'_1-d'_5	(1)	no
3	3	np_{3a}	-	d_1-d_6	(1)	yes
		np_{3b}	-	d_1-d_6	(2)	no
	1	np_{3c}	-	d'_1-d'_5	(1)	yes
		np_{3d}	-	d_4-d_{12}	(1)	yes
	2	np_{3e}	-	d_{10}-d_{15}	(1 2 3 4 5)	yes
	4	np_{3f}	-	nil	nil	yes
	All	PGP_{3a}	np_{3a},np_{3d},np_{3e}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{3b}	np_{3b}	d_1-d_6	(2)	no
		PGP_{3c}	np_{3c}	d'_1-d'_5	(1)	no
4	4	np_{4a}	-	nil	nil	yes
	1	np_{4b}	-	d'_1-d'_5	(1)	yes
		np_{4c}	-	d_4-d_{12}	(1)	yes
	2	np_{4d}	-	d_{10}-d_{15}	(1 2 3 4 5)	yes
	3	np_{4e}	-	d_1-d_6	(1)	yes
	All	PGP_{4a}	np_{4c},np_{4d},np_{4e}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{4b}	np_{4b}	d'_1-d'_5	(1)	no

Figure 55: Example Subsequent Contents of Nodes' Network-Models

Node	Model	Entry	Parts	Track	V-types	Trans
1	All	PGP_{1a}	np_{1a}	d'_1-d'_5	(1)	no
		PGP_{1b}	np_{1b}	d'_1-d'_5	(2)	no
		PGP_{1c}	np_{1c},np_{1e},np_{1f}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{1d}	np_{1d},np_{1e}	d_4-d_{15}	(1 2 3 4 5)	no
2	2	np_{2a}	-	d_8-d_{15}	(1)	yes
		np_{2f}	-	d_{10}-d_{15}	(2 3 4 5)	yes
	All	PGP_{2a}	np_{2a},np_{2c},np_{2d}	d_1-d_{15}	(1)	no
	All	PGP_{2b}	np_{2b}	d'_1-d'_5	(1)	no
	All	PGP_{2c}	np_{2f}	d_{10}-d_{15}	(2 3 4 5)	no
3	All	PGP_{3a}	np_{3a},np_{3d},np_{3e}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{3b}	np_{3b}	d_1-d_6	(2)	no
		PGP_{3c}	np_{3c}	d'_1-d'_5	(1)	no
4	All	PGP_{4a}	np_{4c},np_{4d},np_{4e}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{4b}	np_{4b}	d'_1-d'_5	(1)	no

Node-plans and PGPs not listed from the previous figure are assumed unchanged.

Figure 56: Example of Still Later Contents of Nodes' Network-Models

divide its local plan into a plan to form the track d_8-d_{15} with vehicle type 1, and another plan to form the track d_{10}-d_{15} with the remaining vehicle types. These changes are reflected in its node-model of itself and in its PGPs, as is shown in Figure 56. The node-plans for these changed local plans are communicated, but because of communication delays the nodes will have somewhat different views of important PGPs before the changed node-plans are received (Figure 56).

Finally, when these node-plans are received by other nodes and their node-models are updated, then the view of highly-rated PGPs becomes consistent once again, as is shown in Figure 57.

Node	Model	Entry	Parts	Track	V-types	Trans
1	1	np_{1c}	-	d_4-d_{12}	(1)	yes
		np_{1d}	-	d_4-d_{12}	(2)	no
	2	np_{1e}	-	d_{10}-d_{15}	(1)	yes
		np_{1h}	-	d_{10}-d_{15}	(2 3 4 5)	yes
	All	PGP_{1a}	np_{1a}	d'_1-d'_5	(1)	no
		PGP_{1b}	np_{1b}	d'_1-d'_5	(2)	no
		PGP_{1c}	np_{1c},np_{1e},np_{1f}	d_1-d_{15}	(1)	no
		PGP_{1d}	np_{1d},np_{1h}	d_4-d_{15}	(2 3 4 5)	no
2	2	np_{2a}	-	d_{10}-d_{15}	(1)	yes
		np_{2f}	-	d_{10}-d_{15}	(2 3 4 5)	yes
	All	PGP_{2a}	np_{2a},np_{2c},np_{2d}	d_1-d_{15}	(1)	no
	All	PGP_{2b}	np_{2b}	d'_1-d'_5	(1)	no
	All	PGP_{2c}	np_{2f}	d_{10}-d_{15}	(2 3 4 5)	no
3	2	np_{3e}	-	d_{10}-d_{15}	(1)	yes
		np_{3g}	-	d_{10}-d_{15}	(2 3 4 5)	yes
	All	PGP_{3a}	np_{3a},np_{3d},np_{3e}	d_1-d_{15}	(1)	no
		PGP_{3b}	np_{3b}	d_1-d_6	(2)	no
		PGP_{3c}	np_{3c}	d'_1-d'_5	(1)	no
	All	PGP_{3d}	np_{3g}	d_{10}-d_{15}	(2 3 4 5)	no
4	2	np_{4d}	-	d_{10}-d_{15}	(1)	yes
		np_{4f}	-	d_{10}-d_{15}	(2 3 4 5)	yes
	All	PGP_{4a}	np_{4c},np_{4d},np_{4e}	d_1-d_{15}	(1)	no
		PGP_{4b}	np_{4b}	d'_1-d'_5	(1)	no
	All	PGP_{4c}	np_{4f}	d_{10}-d_{15}	(2 3 4 5)	no

Node-plans and PGPs not listed from the previous figure are assumed unchanged.

Figure 57: Example of Final Contents of Nodes' Network-Models

The previous example assumes a meta-level organization where each node broadcasts its most highly-rated node-plans, and the nodes individually form PGPs to recognize when they should cooperate. In an alternative meta-level organization, consider what happens when nodes 1, 2, and 3 send node-plans to $node_4$, and $node_4$ forms PGPs and sends them back to the other nodes. The initial situation is identical to the previous example (Figure 54), except that $node_4$ does not transmit is idle node-plan (since it does not transmit node-plans at all). After it receives node-plans from the other nodes, $node_4$ forms PGPs; meanwhile, the other nodes' network-models have not changed, as shown in Figure 58. Note that the node-plans in $node_4$'s node-models are not transmitted, but the two PGPs that it forms are both sent to the other nodes, when relevant.

The mechanisms for communicating about PGPs, which are more fully described in the next chapter, are used by $node_4$ to decide where to send PGPs: it sends a PGP to each node that participates in it. Other nodes do not need the PGP since $node_4$ is coordinating the nodes (if each node is forming PGPs and coordinating with others, then each may need to know about all highly-rated PGPs whether or not they participate in them). When the other nodes receive the PGPs from $node_4$, they incorporate these into their network-models. They match a received PGP to any local participating node-plans, and through these node-plans find any existing PGPs to modify. If the received PGP is unrelated to any local node-plans a new PGP is created for it. The resultant network-models are shown in Figure 59.

Once again, when $node_2$ receives a predictive partial solution from $node_1$, it divides its local plan into two. It sends the changed node-plans to $node_4$, which then modifies the PGPs and distributes them. The advantage of using $node_4$ as a coordinating node is that the other nodes need not process and store as much information; but the disadvantage is that the other nodes do not get the more global PGPs as early when they must wait for $node_4$ to send back PGPs instead of forming PGPs locally.

These examples show the basic activities in forming, maintaining, and communicating about node-models and network-models in two different meta-level organizations. In each case, enough information is exchanged so that each node becomes sufficiently aware of important PGPs so that they recognize that they are working on larger global plans. Because this chapter is chiefly concerned with

Node	Model	Entry	Participants	Track	V-types	Sent?
1	1	np_{1a}	-	d_1^i–d_5^i	(1)	yes
		np_{1b}	-	d_1^l–d_5^l	(2)	no
		np_{1c}	-	d_4–d_{12}	(1)	yes
		np_{1d}	-	d_4–d_{12}	(2)	no
	All	PGP_{1a}	np_{1a}	d_1^l–d_5^l	(1)	no
		PGP_{1b}	np_{1b}	d_1^l–d_5^l	(2)	no
		PGP_{1c}	np_{1c}	d_4–d_{12}	(1)	no
		PGP_{1d}	np_{1d}	d_4–d_{12}	(2)	no
2	2	np_{2a}	-	d_{10}–d_{15}	(1 2 3 4 5)	yes
	All	PGP_{2a}	np_{2a}	d_{10}–d_{15}	(1 2 3 4 5)	no
3	3	np_{3a}	-	d_1–d_6	(1)	yes
		np_{3b}	-	d_1–d_6	(2)	no
	All	PGP_{3a}	np_{3a}	d_1–d_6	(1)	no
		PGP_{3b}	np_{3b}	d_1–d_6	(2)	no
4	4	np_{4a}	-	nil	nil	no
	1	np_{4b}	-	d_1^l–d_5^l	(1)	no
		np_{4c}	-	d_4–d_{12}	(1)	no
	2	np_{4d}	-	d_{10}–d_{15}	(1 2 3 4 5)	no
	3	np_{4e}	-	d_1–d_6	(1)	no
	All	PGP_{4a}	np_{4c},np_{4d},np_{4e}	d_1–d_{15}	(1 2 3 4 5)	yes
		PGP_{4b}	np_{4b}	d_1^l–d_5^l	(1)	yes

Figure 58: Example Where Node Forms PGPs for Network.

Node	Model	Entry	Participants	Track	V-types	Sent?
1	1	np_{1a}	-	d'_1-d'_5	(1)	yes
		np_{1b}	-	d'_1-d'_5	(2)	no
		np_{1c}	-	d_4-d_{12}	(1)	yes
		np_{1d}	-	d_4-d_{12}	(2)	no
	All	PGP_{1a}	np_{1a}	d'_1-d'_5	(1)	no
		PGP_{1b}	np_{1b}	d'_1-d'_5	(2)	no
		PGP_{1c}	np_{1c}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{1d}	np_{1d}	d_4-d_{12}	(2)	no
2	2	np_{2a}	-	d_{10}-d_{15}	(1 2 3 4 5)	yes
	All	PGP_{2a}	np_{2a}	d_1-d_{15}	(1 2 3 4 5)	no
3	3	np_{3a}	-	d_1-d_6	(1)	yes
		np_{3b}	-	d_1-d_6	(2)	no
	All	PGP_{3a}	np_{3a}	d_1-d_{15}	(1 2 3 4 5)	no
		PGP_{3b}	np_{3b}	d_1-d_6	(2)	no
4	4	np_{4a}	-	nil	nil	no
	1	np_{4b}	-	d'_1-d'_5	(1)	no
		np_{4c}	-	d_4-d_{12}	(1)	no
	2	np_{4d}	-	d_{10}-d_{15}	(1 2 3 4 5)	no
	3	np_{4e}	-	d_1-d_6	(1)	no
	All	PGP_{4a}	np_{4c},np_{4d},np_{4e}	d_1-d_{15}	(1 2 3 4 5)	yes
		PGP_{4b}	np_{4b}	d'_1-d'_5	(1)	yes

Figure 59: Example Distributing PGPs Among Nodes.

identifying PGGs, these examples illustrate how nodes communicate about node-plans and PGPs so that they become mutually aware of PGGs. The next chapter describes how the nodes develop PGPs that better coordinate their activities so that they cooperate on PGGs more effectively.

6.4 Generalizing the Mechanisms for Recognizing Partial Global Goals

Most distributed computing systems have network-models: at least one of the computing agents has a model of network activity. Computing agents use the model to make control decisions about how the tasks in the network should be assigned and pursued for the network as a whole to work as effectively as possible. Sometimes these models are very general (the basic responsibilities and capabilities of each agent), while at other times they are extremely detailed (exactly what tasks each node is doing). The choice as to what level of representation is needed in a network-model depends on the characteristics of the network and the task domain, such as the cost of communication, the speed of communication, the rate at which tasks are completed, and the rate at which capabilities and responsibilities evolve.

If a computing node takes an average of t time units to complete a task, then, to coordinate individual tasks, nodes must be able to exchange coordination messages in time less than t. Otherwise, by the time the messages are exchanged they are obsolete. Similarly, if nodes communicate about individual tasks, then they must be able to exchange messages every t time units. Nodes that communicate details about individual tasks need communication channels that are faster than the individual tasks and that have sufficient bandwidth to communicate separately about each task. When communication channels are slower, then nodes should communicate about groups of tasks for their more extended future. Also, when communication channels have less bandwidth, then nodes should not exchange details about individual tasks but instead should summarize groups of tasks.

The level of detail for modeling the network chosen in this research is based on assumed relationships between the communication channels and the local node capabilities. We assume that communication is generally slower than the time

needed to perform a primitive task (execute a KS), so that nodes should not exchange messages about individual KSIs but should communicate about groups of tasks. In addition, we assume that communication is sufficiently expensive to warrant being selective about what coordination information to exchange, but where it is less expensive for nodes to communicate needed coordination information than it is to have nodes infer that information by duplicating each other's reasoning. As a result, our mechanisms were developed to communicate about entire plans and the long-term activities being taken in those plans: the level of detail allows nodes to reason about coordinating the construction of entire solutions without detailing the construction of each hypothesis along the way to those solutions. Because they exchange only the long-term and objective information about plans, the nodes reduce communication costs: the contents of each message are decreased (since detailed short-term information is not exchanged) and the number of messages is decreased (since the objectives and long-term plans change less often than the short-term details).

If the costs and delays of communication decrease, then the level of detail should be changed: it may be advantageous for nodes to coordinate individual actions (KSs). In the extreme situation where communication is extremely cheap and rapid, nodes can be so closely coupled that they resemble a parallel processing system where they schedule tasks as a group. On the other hand, if the costs and delays of communication increase, then nodes may need to communicate at another level of detail, exchanging information not about individual plans (which may be completed by the time the message arrives at its destination) but about groups of plans. In the limit where communication becomes very costly and slow, nodes should only communicate about general capabilities and responsibilities—they should only maintain a basic organizational framework and not coordinate their more specific activities.

Useful representations of node activities all have one thing in common, however, no matter what the characteristics of the communication channels and task domain are. Although there is no such thing as a completely "general" representation for network-models, all useful network-models simultaneously represent activities at different levels of detail. The detail needed for making local control decisions is different from that needed for identifying when nodes are working on larger, shared goals, which in turn is different from that needed to determine

where future tasks are likely to occur.

The network-models described in this chapter enable nodes to represent information at appropriate levels of detail depending on how that information will be used. With its models of local and network activity, a node can not only make informed decisions about its local activities, but can also recognize when it should cooperate with others, how it should cooperate with others, and what the basic capabilities of others are. Because it has knowledge at the right level of detail for each type of control decision it must make, a node can make better control decisions.

In the description of the data structures used in the network-models, some attempt was made to motivate the more general aspects of what is being represented. Although the contents of most of the fields in the various structures are domain-dependent, the basic concepts behind what is being represented and why are more general:

- node-plans summarize local plans so only the aspects that affect outward behavior (interactions with other nodes) are represented;
- a plan-activity-map represents a view of how the node will behave (what results it will produce and when);
- the more general characteristics of a node represent its basic capabilities and responsibilities;
- PGPs represent the shared objectives and planned interactions between cooperating nodes;
- and the meta-level organization identifies the coordination responsibilities and communication patterns among nodes.

Thus, although the specific *form* that the knowledge represented in these structures takes is domain-dependent, the *type* of knowledge represented is domain-independent.

The plans generated locally by nodes are well suited to summarization because they explicitly represent the plans' objectives, long-term plans, and short-term detailed plans. Building node-plans is therefore a relatively simple task of extracting the appropriate information. Since most planners in other domains develop plans

hierarchically, the relevant information about plans at different levels is generally accessible. Planners that do not have such a representation would need mechanisms to cluster primitive actions into larger groups. Similarly, the mechanisms for meta-level communication and for maintaining network-models are straightforward and generally useful: their main function is to keep track of information being exchanged. Finally, the mechanisms for recognizing PGGs involve domain information, since identifying objectives that are part of a larger goal necessitates domain knowledge about how local goals could be related. The principal issue involved in these mechanisms is in reducing the combinatorics involved in identifying when and how nodes should cooperate.

A node's model of the network provides the basis for identifying cooperative situations. In domains where tasks are decomposed by some central authority, such mechanisms are unnecessary—when tasks are distributed, their relationships with other tasks can be distributed with them. However, in many domains, and particularly in domains where distributed computing systems are most effective, tasks (subproblems) are inherently distributed among nodes. The nodes therefore must communicate information to recognize opportunities for cooperation. In this chapter, we have shown how nodes can build network-models to identify shared PGGs; in the next chapter, we describe how nodes decide how to coordinate their actions after they have identified that they should cooperate.

The highest and best form of efficiency is the spontaneous cooperation of a free people. -Woodrow Wilson

Chapter 7

Coordination Through Partial Global Planning

When nodes identify opportunities for cooperation, they must then plan their actions and interactions so that they cooperate effectively. This chapter focuses both on how a node can plan cooperative activities and on how nodes can arrive at consistent views of network coordination.

To recognize how actions and interactions should be modified for more effective cooperation, a node begins with its view of network activity (from its network-model) and with some initial bias towards certain types of cooperation. This initial bias is the node's knowledge about the network's goals of cooperation (described in Chapter 1). For example, if the network should generate results as rapidly as possible (as opposed to more slowly generating results with higher confidence), then the goals of cooperation include avoiding the redundant derivation of results for verification. As another example, if the network should reduce the number of results exchanged between nodes, then the node should plan activities so that only the most encompassing local results are developed and exchanged. As a final example, if the network should balance problem solving load among nodes, then the node should coordinate the exchange of tasks between nodes.

When modifying actions and interactions to improve coordination, a node uses the long-term view of nodes' activities from the node-plans contained in the network-model: by coordinating long-term activities (intermediate-goals) instead of detailed actions (KSIs), a node lowers the cost of coordination since it reasons about less information and needs to recalculate coordination decisions less frequently (the long-term plans of a node change less often than the details), but nodes may occasionally take detailed actions that are not coordinated since they do not model each other at this level. The node determines whether the concurrent activities of cooperating nodes could be altered to improve network problem solving. It knows the order in which each node will pursue major subgoals (intermediate-goals), and has predictions about how long each subgoal will take, so the node can represent the interleaved activities of nodes to roughly estimate when they will develop various partial solutions. If the node locally recognizes situations where some node plans to develop unimportant (or redundant) partial solutions before more important partial solutions, then, by rearranging the activities of that node, the node plans more effective and coordinated activity. It then uses this more global view of coordinated activity to alter local plans so that they are more useful for network problem solving: the planner changes the long-term plans (order of intermediate-goals) of a local plan based on the order proposed in the more global view (from the PGP).

If nodes have complete, up-to-date, and consistent network-models, then they could all arrive at the same view of network coordination. However, because they communicate limited amounts of information and their messages take time to be received, nodes seldom have identical views of the network. In a distributed system, the question therefore arises as to how the network as a whole should develop and distribute coordination information. Should each node decide for itself how to coordinate with others or should responsibility for coordination rest with a small number of "coordinator" nodes? How should a node balance its need to be predictable (so that it coordinates with others in an expected way) with its need to be responsive (so that it reacts to unexpected local information in a timely manner)? In short, what style of cooperation should nodes use (how should they be organized) so that they develop, pursue, and maintain more global plans in dynamically changing environments?

A suitable style of cooperation depends on the needs and characteristics of the

network, including the capabilities of the various nodes, the types of tasks the nodes must pursue, the distribution of tasks in the network, the communication bandwidth and delays, and the reliability of the nodes and communication links. No single style of cooperation is effective in all situations: if all tasks enter the network at a single node, then task-passing is a priority while communication to recognize when nodes are working on pieces of the same task is less important; if tasks are inherently distributed, the opposite holds true; if nodes are reliable, then a centralized coordinator might be useful; if nodes are unreliable, then each must take more responsibility for itself.

Whereas previous mechanisms for coordinating distributed problem solvers were developed to support a specific style of cooperation, the representations and mechanisms for planning and control developed in the last chapter and this chapter lay the foundation for more general coordination techniques. Each of the various styles of cooperation can be achieved within the framework of forming and communicating about partial global plans. The last chapter described how a node develops a network-model and uses this model to recognize when a group of nodes should cooperate. In this chapter, we described the techniques for deciding how nodes should cooperate and for distributing these decisions among cooperating nodes. In the next section, background information is briefly discussed, and the following section provides an overview of the mechanisms developed for partial global planning (PGPlanning). Section 7.3 then describes these mechanisms in more detail: how nodes generate partial global plans, how nodes use partial global plans to improve local problem solving and communication decisions, how nodes communicate about partial global plans to reach a more consistent view of cooperation, and how communication of partial global plans allow nodes to predict future activities and to recognize alternative assignments of tasks in the network. The final section of the chapter generalizes these mechanisms to indicate how they are appropriate in other domains and could be implemented in those domains.

7.1 Background

Issues and approaches in coordinating distributed problem solvers (and distributed computing systems in general) were discussed in Chapter 1 and will

not be repeated in detail here. It is important to recall that previous approaches to coordinating problem solvers have stressed particular styles of cooperation, such as having a centralized coordinator, or forming contracts, or exchanging local information based on organizational knowledge to eventually converge on solutions. Each of these approaches was also geared to different goals of cooperation, such as ensuring against resource conflicts, or decomposing and distributing tasks, or exploring how local results should be combined into more global results.

An important goal of this research is to develop a more general framework for cooperation—one that lets nodes cooperate in different styles and pursue a variety of cooperative goals. The mechanisms we introduce are based on the view that coordination is a planning task. To cooperate effectively, nodes need to plan complementary actions and anticipate their interactions. Nodes need more global plans that represent their current view of how they are cooperating, even though in dynamically changing environments each may have a somewhat different view of cooperation than the others. Coordinating through a centralized leader (that plans for the group), or through contracts (which are plans shared by the contractor and the contractee), or through exchanging partial results to converge on solutions (where nodes share an organizational plan about what types of information to exchange), all involve plans. The partial global planning framework lets nodes coordinate in any of these styles without resorting to different mechanisms for different styles. Moreover, by reasoning about how local plans fit together into partial global plans and about the concurrent actions and interactions of nodes, the nodes can cooperate to achieve any of a number of goals such as avoiding redundant or conflicting activities, transferring activities from overburdened nodes to underutilized nodes, or integrating local results into more global results.

7.2 Overview of Partial Global Planning

A PGP's *plan-activity-map* interleaves the concurrent activities of the participating node-plans, and each activity has an estimate of when it begins, when it ends, what task (part of a track) it is working on, and what results (tracks) it will produce. The planner scans the plan-activity-map to find activities that are more useful (such as activities that form important results to share) or less useful (such as activities that unnecessarily form redundant results). Each activity

is rated based on attributes such as its expected time cost, its expected result quality, how it will be affected by preceding activities, and how it will affect later activities. The planner attempts to reorder activities to move more highly-rated ones earlier in the plan. For example, node 1 (Figure 3) has a plan to build track d_4–d_{12} and, since it locally has no reason to prefer certain activities over others, it chose (4 5 6 7 8 9 10 11 12) as the ordering (Figure 44). When the activities of nodes 2 and 3 are incorporated in the PGP, however, node 1's activities are no longer equally rated: because they provide node 2 with predictive information, the activities for generating tracks neighboring node 2 are more highly rated (for example, the partial track d_8–d_9), but because they may generate redundant results, the activities for times 4–6 and 10–12 have their ratings lowered. By rating activities in alternative orderings, the planner determines that node 1's activities should be reordered to (9 8 7 10 11 12 6 5 4).

To reduce planning overhead, the planner does not guarantee an optimal ordering (which would require a large search) but instead uses a less costly hill-climbing algorithm that generates a satisfactory ordering: it begins with the plan-activity-map built from the node-plan plan-activity-maps (which it expects each node to currently be following), and rates the activities. It then reorders the activities so that the most highly rated occur earlier, and then rerates the activities given this new order. Because the ratings of activities depend on their relative ordering, reordering them may reduce ratings of some activities and raise ratings of others. The sum of the ratings before and after reordering are compared: if the original ordering has a higher total, then it is used; if the new order is better, then the process repeats until no better ordering can be found.

The planner uses the plan-activity-map to form the solution-construction-graph. It first identifies the earliest times that different pieces of the overall solution will be generated and at what nodes, and then determines when and where they should be integrated into a single answer. For example, after reordering the plan-activity-map as described above for the PGP to form track d_1–d_{15} (Figure 3), the planner makes a solution-construction-graph specifying that tracks d_1–d_6 from node 3 and d_7–d_{11} from node 1 should be combined at node 1, and the resulting track d_1–d_{11} should be combined at node 2 with that node's track d_{12}–d_{15}. Alternatively, in a slightly different network where node 4 has very good integration expertise (KSs), the solution-construction-graph has nodes 1, 2, and 3 send their

tracks to 4 for integration. The planner builds the solution-construction-graph by: finding the pair of partial results that can be combined earliest (time when both could be present at the same integrating node plus an estimate of how long that node will take, depending on its expertise, to combine them); adding the combination as a new partial result; and then repeating this process until a complete result is formed. This inexpensive, iterative algorithm generates a graph that is acceptable although possibly non-optimal (see Section 7.3.4).

The solution-construction-graph improves communication decisions since a node has a more global view of where results are needed than it has with a more local view [Durfee *et al.*, 1985b; Durfee *et al.*, 1987]. For example, it knows that track d_1–d_6 should be sent from node 3 to node 1. A node's planner can also make better local decisions by identifying whether it is or is not responsible for a particular result, and how much time it has to generate that result. In situations with multiple solutions, integration responsibilities for a solution are assigned to one node so that others can more quickly move on to other solutions. Also, since two partial results cannot be integrated until they are both at the integrating node, the planner may identify cases where the node has some time to spare: it predicts that n time-units are needed to form and send the result, but that the other result will not be ready for $n + i$ time-units. Our mechanisms allow the planner to work on important activities for other PGPs (perhaps generating predictive information) during the other i time-units, treating the $n+i$ time-units as a window in which the task to generate the result can be moved around [Vere, 1983].

Planning Node Activities

The planner reasons about the concurrent actions of nodes and about their potential interactions to find the next action for the node to take, as shown in Figure 60. It first uses any received network information (node-plans and PGPs) to update the network-model. It then finds the current-PGP: the PGP that specifies activities that the node should do at this time. The local plan that contributes to this PGP is updated and the next action is found. For example, if the PGP indicates that the local plan should develop data in a different order, the plan's long-term information is changed to reflect this and, if necessary, detailed short-term actions are found for the next data to process. When it is updated, the local plan may

A node's planner will:

1. receive network information;
2. find the next action using network-model:
   (a) update local abstract view with new data;
   (b) update network-model, including PGPs, using changed local and received information (factoring in credibility based on source of information);
   (c) map through the PGPs whose local plans are active, for each:
      i. construct the plan-activity-map, considering other PGPs;
      ii. find the best reordered plan-activity-map for the PGP;
      iii. if permitted, update the PGP and its solution-construction-graph;
      iv. update the affected node-plans
   (d) find the current-PGP (this node's current activity);
   (e) find next action for node based on local plan of current-PGP;
   (f) if no next action (local plan inactive) then go to 2b (local plans may have changed), else schedule the next action;
3. transmit any new and modified network information.

Figure 60: The Principal Planning Activities.

become inactive—it may not yet have data in the area where it is expected to work—so several PGPs may need to be tried before one with an active plan is chosen and node problem solving can continue (Figure 60, steps 2d–f). Finally, any highly-rated PGPs or node-plans that have been altered are sent to whatever other nodes should be informed (based on the meta-level organization). Sending only highly-rated information can reduce communication costs and the number of PGPs (combinations of node-plans), but may cause views to be inconsistent or important PGPs to be missed. A parameter specifies how highly-rated information must be to be sent. When more or less complete communication would improve planning, the experimenter (and in future implementations perhaps the node itself) may alter this parameter.

When finding the current-PGP, the planner first updates its set of local plans based on any new data from its sensors or other nodes (Figure 60, step 2a). The new data modifies the abstraction hierarchy, and the planner forms new plans for new potential solutions and modifies existing plans whose clustered information has changed. Node-plans are created for any new plans and the node-plans of modified plans are updated. The planner then updates the network-model using

new and updated node-plans either formed locally or received (Figure 60, step 2b). It updates the PGPs of any updated node-plans and uses new node-plans to either update existing PGPs (if compatible) or to generate new PGPs (if incompatible with all current PGPs). If any PGPs have been modified or created (as a result of changed node-plans or reception of credible PGPs), the planner then checks the set of PGPs, merging any that are now compatible and separating any that are no longer compatible. For example, if the same vehicle passes through a node's area twice, then the node initially develops separate PGPs for the two potential tracks. If it later receives node-plans from other nodes indicating that its two potential tracks are connected, then it merges the PGPs into a single larger PGP.

Once the network-model is updated, the planner proceeds to find the current-PGP. It first finds the PGPs to consider (leaving out PGPs that have already failed to generate useful actions because their local plans are inactive) and orders them. It also decides what nodes to plan for—usually just the current node, but if this node is also to coordinate others it should plan for them as well. The planner then steps through the PGPs from most highly-rated down (Figure 60, step 2c), updating their plan-activity-maps and solution-construction-graphs, until all the desired nodes are planned for or no PGPs remain. For example, when the nodes it should plan for do not all participate in the same PGPs, the planner must update multiple PGPs until a current activity is found for each node.

When updating a PGP, the planner first generates a current plan-activity-map by interleaving the activities of each of the participating plans (Figure 60, step 2ci). For the local plan, the planner uses the past and predicted steps straight from the plan data structure. For non-local plans, the planner can get the activities from two potential sources: from a received node-plan (if there is one) or from the plan-activity-map of the PGP (it may have been received from a node that had the node-plan). If the planner has information from both sources, it chooses between them using the information accompanying received node-plans and PGPs specifying how current and credible the model of the plan is. If the planner has neither source, it must have received the PGP (it could not have formed it locally without the node-plan) with the sending node intentionally holding back information so that this node could not generate its own (possibly better) plan-activity-map but instead must blindly follow the plan-activity-map supplied with the PGP.

If it forms one, the planner checks the plan-activity-map against any PGPs for which it has already planned. A node that participates in this PGP may also be part of a previously formed PGP, and the plan-activity-maps are compared to make sure the node is not expected to do two things at once. When there is a conflict, the node's activities in this (less highly-rated) PGP are moved to future, non-conflicting times. The planner then uses the hill-climbing algorithm previously described to reorder activities for better coordination (Figure 60, step 2cii). The sum of the activity ratings for the new plan-activity-map is multiplied by the credibility of the node's own plans, and this value is compared with the value of the previous plan-activity-map (if any).[1] If the value of the new plan-activity-map is higher, the planner updates the PGP with the new plan-activity-map (Figure 60, step 2ciii), forms the solution-construction-graph and communication expectations, and modifies its local plans and their node-plans based on the better activities (Figure 60, step 2civ). When the plan-activity-map is improved but the credibility factors (authority relationships) do not allow the planner to change the PGP, the planner can transmit the node-plans that it believes should be modified: it assumes that if nodes with authority have the same view of these node-plans that it has, then those nodes will modify the PGP appropriately and the modified PGP will eventually be sent back to this node.

Once all of the PGPs have been updated and an action has been found for the node, the final step is to send out any important modified node-plans and PGPs, as described previously (Figure 60, step 3). To make local control decisions based on the best, most up-to-date view of network activities, a node invokes the entire series of activities—from modifying the network-model, to developing PGP plan-activity-maps, to sending out new information—each time it needs to choose an action to take. If the node wants to conserve computational resources, it can do these activities less often (or restrict itself to only working with a smaller set of only the most highly-rated plans and PGPs) at the cost of possibly making poorer control decisions. Because nodes are asynchronously performing coordination activities and are interleaving these activities with problem solving, they must each balance the costs and benefits of these mechanisms.

[1] A PGP has a previous plan-activity-map if it was received or previously formed locally, and its value is the sum of its activity ratings multiplied by the credibility of the node that generated it.

7.3 The Partial Global Planning Mechanisms in Detail

The partial global planning mechanisms extend the local planning mechanisms in much the same way that the local planning mechanisms extend a node's basic control scheme. Because local plans combine individual actions into purposeful, coherent sequences, control decisions are based on a more complete view of how actions may work toward local long-term goals. In a similar way, partial global plans combine several local plans into coordinated groups, so that control decisions are based on how local plans and their actions may work toward network long-term goals.

The planning mechanisms retain their role of determining what action (KSI) to invoke next, as in Figure 22. The partial global planning mechanisms are introduced into the planner described in Chapter 4. Just as the local planning mechanisms use local plans to improve decisions about the best KSI, the partial global planning mechanisms use PGPs to improve decisions about the best local plan to pursue. Schematically, the dispatcher in the local planner that simply chooses the most highly-rated local plan (Figure 23) is replaced with the partial global planning mechanisms that determines the local plan to pursue based on the PGPs (Figure 61).

The architecture of the partial global planner (also referred to as the **PGPlanner**) is sketched in Figure 62. The PGPlanner's network-model is composed of several node-models and the PGP queue. How node-models are generated and maintained was described in the previous chapter: node-plans are formed from the node's local plans and from received node-plan messages. The PGP generator scans the node-plans in the node-models to recognize PGGs, and it forms and maintains PGPs, as was also described in the previous chapter. This chapter examines the remaining mechanisms. The PGP communication mechanisms allow nodes to exchange PGPs to update each other's network-models and to converge on consistent views of cooperation when possible. The dispatcher finds a set of highly-rated PGPs that can be active concurrently (that are being pursued by different groups of nodes). The PGP executor then generates additional information about active PGPs: how the activities of the participating nodes may interact and could potentially be reordered to better achieve the PGP's objectives; where

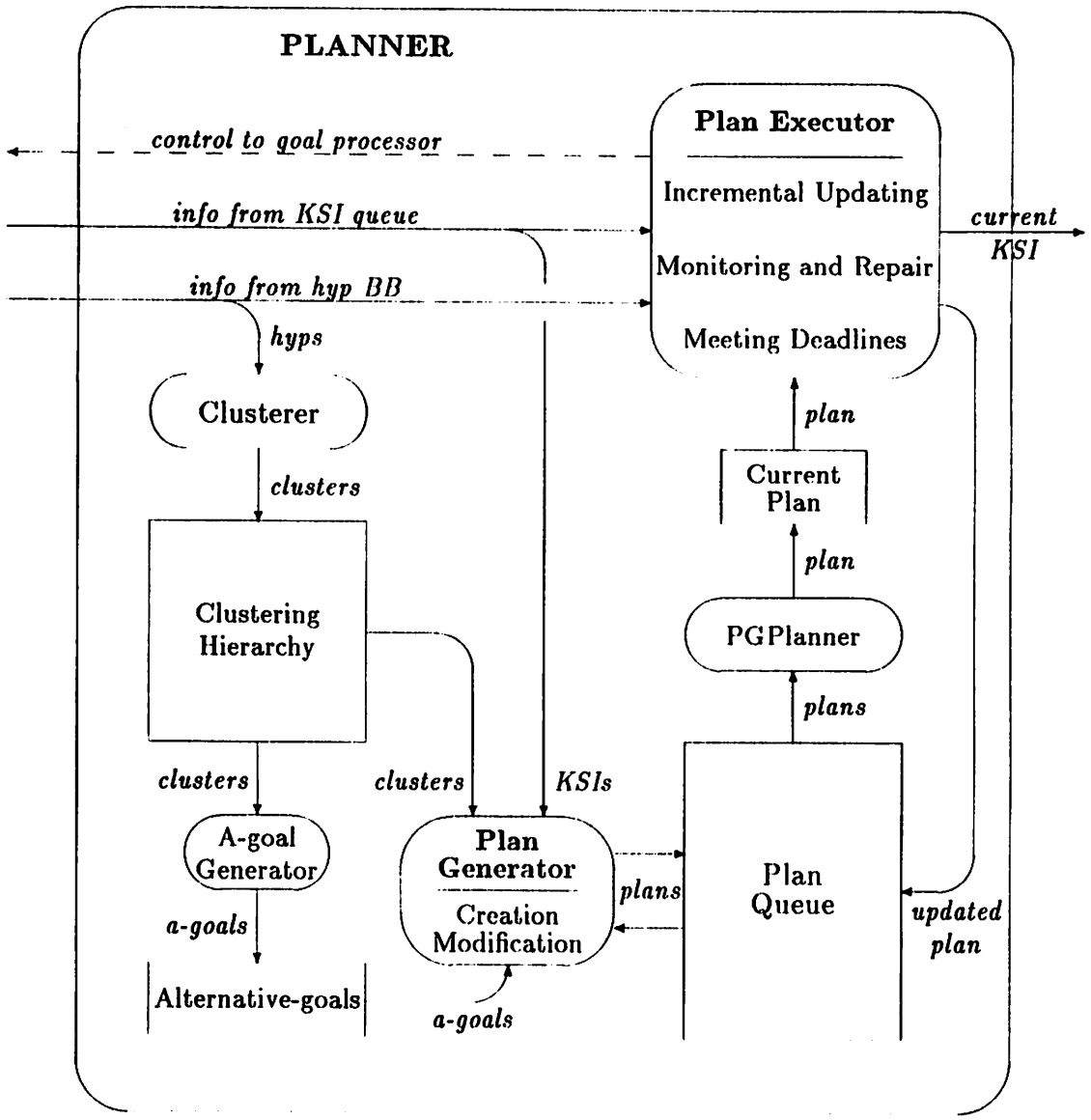


Figure 61: The Architecture of the Planner With the PGPlanner.

and when the partial solutions being formed at various nodes should be combined to construct the entire solution; what partial solutions should be communicated where to generate the entire solution; how local plans associated with the PGPs should be altered to reflect the better ordering of activities, and what local plan to pursue next.

Section 7.3.1 describes the PGP data structures in detail. Next, the basic sequence of partial global planning actions are summarized in Section 7.3.2. The mechanisms for building a view of concurrent activities (Section 7.3.3), for determining how solutions should be constructed (Section 7.3.4), and for identifying which partial solutions should be communicated and where (Section 7.3.5), are then described. How these mechanisms are used to control the local activities of the node are then detailed in Section 7.3.6. Nodes exchange PGPs to communicate about their potentially different views of network coordination, and communication about PGPs is outlined in Section 7.3.7. Section 7.3.8 describes how nodes modify and transmit PGPs to propose future activities for nodes (task-passing) and how nodes use PGPs to predict future local tasks. The basic sequence of partial global planning actions is then discussed again (Section 7.3.9), indicating how the different mechanisms interact to give the node the best view for making control decisions. Finally, examples of how the mechanisms work are given in Section 7.3.10.

7.3.1 Partial Global Planning Data Structures

When the PGP generator scans the node-models to form and modify PGPs, it is identifying opportunities for cooperation (PGGs). The description of PGPs in the previous chapter therefore focused on the attributes necessary for indicating *when* a group of nodes should cooperate. This section describes PGPs more fully, discussing the representation of *how* this group of nodes should cooperate.

The attributes of a PGP are shown in Figure 63. Several of these attributes were described in the last chapter: the PGP has a name, a list of participating node-plans, the name of the node that formed the PGP, a list of the nodes that participate in the PGP, the time when the PGP was created or last modified, a rating (combining the ratings of the participating node-plans), and objectives (the expected track and vehicle types of a solution). The PGP's long-term information includes a specification about what type of results will be integrated

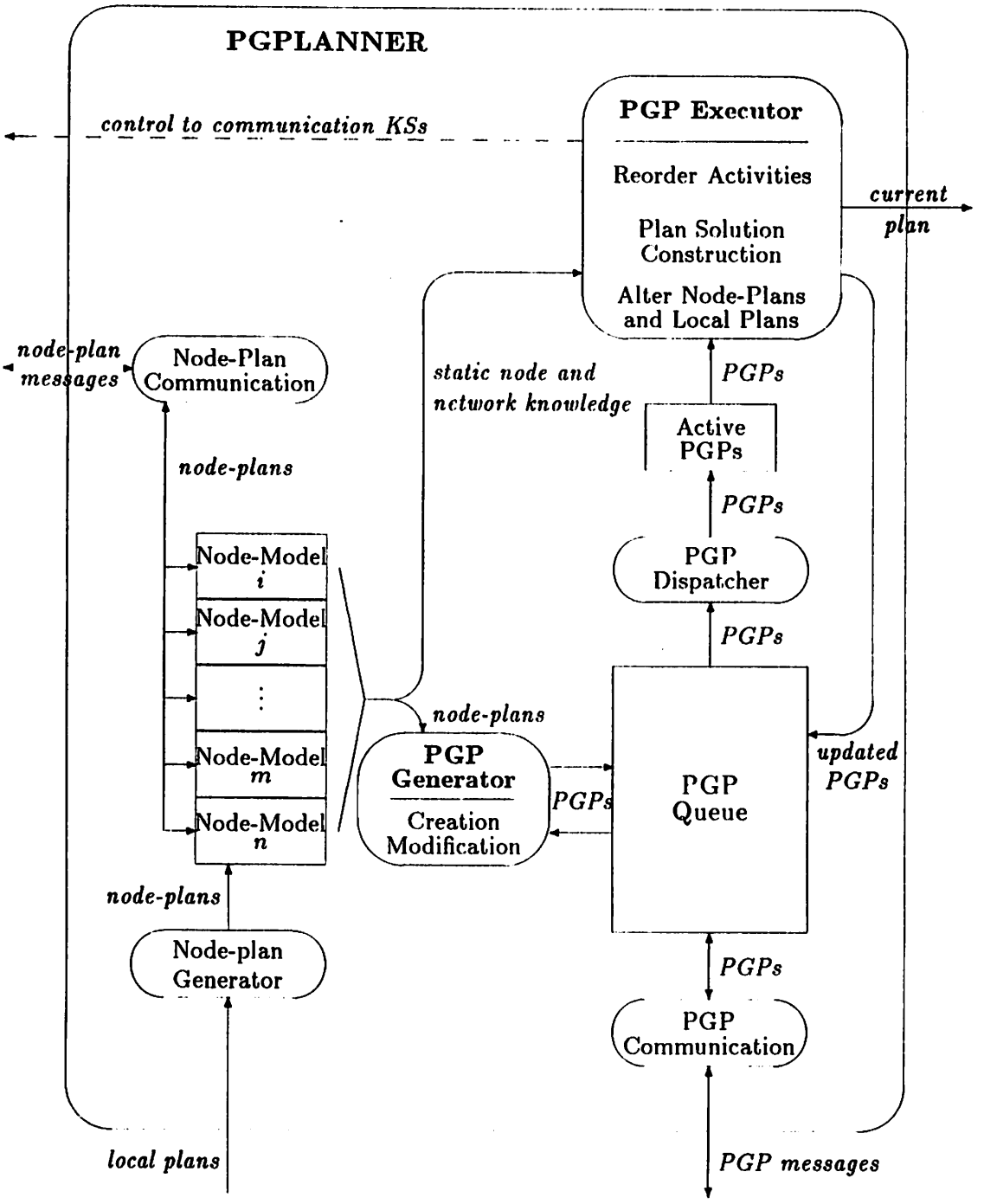


Figure 62: The Architecture of the PGPlanner.

name	A unique name (symbol) pointing at the PGP structure	
participants	A list of the node-plans that participate in this PGP	
node	The node that created this PGP	
nodes	A list of the names of nodes participating in this PGP	
creation-time	When the PGP was created or last updated	
rating	The rating of the PGP	
objective	track	The expected complete track when the pieces from the separate node-plans are combined
	v-event-classes	The vehicle-event-classes (types of vehicles) being tracked
long-term	tracking-levels	The blackboard-level(s) where hypotheses will be integrated into tracks
	plan-activity-map	Sequence of plan-activities: when i-goals will be pursued and partial results will be formed
	solution-construction-graph	What partial solutions will be integrated and indication where and when
	communication	What partial solutions will be exchanged between which nodes
record	sent-to-nodes	Associates nodes that were sent this PGP with the creation-time of the sent PGP
	node-and-plans	Associates with each participating node the name(s) of the participating plan(s)
	plans-and-times	Associates with each participating plan-name the creation-time of that plan
future	to-nodes	Where PGP is sent to elicit future node-plans
	sent-time	When PGP was sent to elicit future node-plans
	responses	Node-plans received in response to the PGP
	respond-to	Node where future node-plan responses are sent
	response-label	Label (PGP-name) to accompany responses
	source-node	Where tasks to be passed come from
	communication	What tasks to send, from where to where

Figure 63: The Attributes of a PGP.

(the blackboard-levels for combining tracks), and representations that indicate the concurrent activities of nodes and how they should cooperate, as is detailed next.

The PGP's plan-activity-map represents the concurrent activities of each participating node. The plan-activity-map is an ordered list of plan-activities, where each plan-activity represents a separate i-goal for a node to achieve. The attributes of a plan-activity were discussed in the previous chapter (Figure 46), and include a begin-time, an end-time, the node performing the activity, an area of focus, and an expected result (partial solution). The PGP's plan-activity-map interleaves plan-activities from each of the participating nodes based on their

begin-time	When the activities to form the partial solution begin
end-time	When the activities to form the partial solution end
duration	How much time the activities need to form the partial solution
nodes	A list of the nodes that are forming the partial solution
expected-result	The basic specifications (time-regions) of the partial solution
supporting-pss	PGP-partial-solutions combined in this PGP-partial-solution
supported-pss	PGP-partial-solutions this PGP-partial-solution is combined into

Figure 64: The Attributes of a PGP-partial-solution.

end-times—to coordinate the nodes, the PGPlanner needs to identify when they will form partial results, not when they will start to form them. By scanning through the plan-activity-map, the PGPlanner can recognize the concurrent activities of the various nodes at an appropriate level of detail: instead of recognizing what specific actions they take concurrently (which would entail a large amount of quickly outdated information), the PGPlanner only needs to identify which i-goals are being pursued concurrently.

Given a plan-activity-map, the PGPlanner forms a *solution-construction-graph* to represent how the partial solutions contributed by each participating node should be integrated into an overall solution. The PGPlanner finds groups of plan-activities for a node and summarizes them as a *PGP-partial-solution*. The attributes of a PGP-partial-solution are shown in Figure 64. Each PGP-partial-solution has a begin-time (when the group of plan-activities begin), an end-time (when the group of plan-activities end), a duration (how much time the plan-activities require), a list of nodes (the node(s) that are building the partial solution), an expected result (a partial track), a list of supporting PGP-partial-solutions (the PGP-partial-solutions that were integrated to form this PGP-partial-solution), and a list of supported PGP-partial-solutions (the PGP-partial-solutions in which this PGP-partial-solution is a component). The leaves of the solution-construction-graph thus represent partial solutions formed at individual nodes, the root of the graph represents the complete solution, and the paths between leaves and the root indicate how and where partial solutions should be integrated. The mechanisms for forming the solution-construction-graph are outlined later, along with discussions about why the duration may be less than the difference between the begin-time and end-time and why several nodes may build the same partial solutions.

The long-term communication information explicitly represents the commu-

nication expectations developed in the solution-construction-graph. By scanning through the solution-construction-graph, the PGPlanner identifies what partial solutions must be transmitted from one node to another, and builds a communication expectation of the form:

(from-node to-node time expected-result)

where the from-node and to-node indicate the source and destination of the message, the time is an estimate of when the message will be transmitted (based on the PGP-partial-solution's end-time), and the expected-result represents the specifications of the hypothesis to be transmitted (its time-locations). The long-term communication information also represents communication that is intended for purposes other than solution construction. For example, if the PGPlanner recognizes that a earlier version of some partial solution should be transmitted for predictive purposes, it represents this exchange in the long-term communication information as well.

The PGP contains a record of where and when it itself has been transmitted in the past, so that the PGPlanner can determine where and when a new or modified PGP should be sent. The PGP also contains a record that indicates the local plans from which it is built, what nodes created those local plans, and when those local plans were created or last updated. Because nodes communicate about PGPs, it is important that they be able to recognize when a received PGP corresponds to an existing local PGP, and to do this they need to compare the information on which each is based. Since they each give local names to PGPs and node-plans, they record the original names given to local plans by the nodes that created them (and transmitted as the plan-name attribute of a node-plan), and use these names as a common basis for determining whether two PGPs correspond to cooperation among the same plans. Thus, each PGP has a node-and-plans attribute that associates with each node the name(s) of its local plan(s) that are participating in the PGP. The PGP also has a plans-and-times attribute that associates with each local plan name the creation-time of that plan (so that given PGPs for the same plans, the PGPlanner can determine which was built from the more recent information).

Finally, the PGP contains information about potential future cooperation. The principal use of this information is for task-passing: when these attributes

are set, the PGP includes information proposing a transfer of tasks, and a node that receives such a PGP will generate a future node-plan in response to it. The PGP's future attributes include: a list of to-nodes where tasks could be transferred, the time when the PGP was sent to those nodes, a list of responses in the form of node-plan messages received for the PGP, the name of the node where responses should be sent, a label for the responses so that the node can associate a received response with the appropriate PGP, the name of the node where the tasks to transfer currently reside (which may or may not be the same as the node proposing the transfer), and the communication expectations representing the exchange of tasks once this has been determined.

Besides the PGP data structures, the PGPlanner uses the data structures for network-models described in the previous chapter, including the meta-level organization and the general characteristics of nodes (stored in their node-models).

To make decisions about how to modify local plans to improve cooperation, the PGPlanner also needs to know the goals of cooperation. Preferences for how to cooperate are currently represented numerically in the *cooperation-parameters* data structure. These parameters are are outlined in Figure 65, and are part of the meta-level organization (so that currently all nodes have identical cooperation-parameters). The PGPlanner uses these parameters to determine a cost for each of the various activities that a node can perform. By using numeric parameters, the PGPlanner can make simple comparisons for conflicting goals of cooperation. For example, if the nodes have a goal of influencing each other only when by doing so they can significantly improve network coordination, then balancing the conflicting goals of improving coordination and maintaining autonomy is difficult. If the cost of maintaining autonomy is half that of improving coordination, then for activities to be reordered the expected quality of coordination (network solution time) by reordering must be at least twice that that would have been achieved otherwise. With numeric values indicating the cost of each of these goals, the overall costs of activities strike a balance between them.

The current set of factors that affect how (and whether) the PGPlanner reorders tasks are redundancy, reliability, duration, predictiveness, locally-predicted, independence, and diversity. *Redundancy* is the cost associated with redundantly deriving results, so that redundant activities are more costly than those that are not. On the other hand, *reliability* is the cost of *not* verifying

redundancy	Cost of redundantly deriving results
reliability	Cost of not verifying results through rederivation
duration	Cost of performing more expensive (longer duration) activities before less expensive
predictiveness	Cost of generating predictive results later
locally-predicted	Cost of not locally pursuing adjacent activities
independence	Cost of deviating from the locally generated order for activities
diversity	Cost of working on results near to results being developed at other nodes
time-cushion	Acceptable amount of time deviation between when actions and interactions were predicted to occur and when they actually occur
solution-construction-redundancy	Number of nodes that should redundantly integrate results to increase reliability
min-task-duration	The minimum expected time a task must require to consider transferring it to another node
task-strategy	The strategy for extracting information (bids) for tasks (focused or broadcast)
task-redundancy	The number of nodes from which information should be requested (for focused strategy)

Figure 65: The Cooperation-Parameters.

results through rederivation, so activities that rederive previous results are less costly than those that do not. Thus, to avoid rederiving redundant results, the redundancy cost should be high and the reliability cost low; while the opposite settings promote verification. *Duration* is the cost of performing more expensive (time-consuming) activities before less expensive activities, and it favors ordering activities so that cheaper activities are pursued sooner. *Predictiveness* is the cost of generating results that are not predictive ahead of those that are, to promote orderings where predictive results are formed earliest. *Locally-predicted* is the cost of generating results that do not extend previous results: since problem solving is most effective when previously formed results are extended into new areas, this factor penalizes orderings where a node forms separate pieces of its result and later merges them. *Independence* is the cost of moving an activity from its position in the initial ordering (formed locally by a node based on the cluster information as described in Chapter 4). Since a node orders its activities based on local information that is not available to the PGPlanner (not part of a node-plan message since it is very context dependent), the independence factor specifies how costly it is to change that ordering. Finally, *diversity* is the cost of performing

an activity "near" activities done at other nodes. To avoid redundantly deriving results in the future, a node should develop results as far as possible from those being developed by other nodes, where "nearness" of activities is measured by how close the sensed time of their data is. How these cost parameters are used is more completely described in Section 7.3.3.

Another important piece of information affecting cooperation is the *time-cushion* that represents the network's view of a "significant" amount of time. The time-cushion is the largest amount of time that a node's predictions about when actions or interactions will take place can deviate before it should react to those deviations. The time-cushion represents the balance between predictability and responsiveness, since a small time-cushion forces nodes to respond more frequently to deviations while a large time-cushion allows them to continue working on their plans in essentially the way that they had expected to despite deviations.

The *solution-construction-redundancy* indicates how many nodes should be responsible for integrating a pair of partial solutions. In a reliable network (where nodes do not fail), no redundancy is needed and this parameter is set to 1. However, in less reliable networks, the nodes should not depend on one node successfully developing a combined solution. When the solution-construction-redundancy is set to n, it means that the partial results will be sent to n different nodes, each of which will integrate the pieces into larger results. Obviously, n should not be larger than necessary to ensure that some node successfully combines the partial results, since having more than 1 node form the combined result is a duplication of effort.

The other information in the cooperation-parameters is used by the PGPlanner to make task-passing decisions, where a task is some set of related activities. The *min-task-duration* indicates the minimum size (expected time needs) of a task that is worth transferring. Because transferring a task requires communication and computation overhead, it should not be done when the costs (time spent by a node) of the task are small. The *task-strategy* specifies whether the PGPlanner should request information (bids) from selected nodes or from all nodes. Since the PGPlanner has a model of the network, it usually focuses requests to those nodes that it believes are underutilized. However, if its network-model is substantially incomplete or out-of-date, broadcasting the request may be appropriate. Finally, the *task-redundancy* parameter indicates the number of underutilized nodes from

which the PGPlanner should request information (in the focused task-passing strategy). A low value reduces the communication and computation of task-passing (since fewer nodes must process and respond to the request), but a high value makes finding the best node more likely. The task-passing mechanisms that use these parameters are more fully discussed in Section 7.3.8.

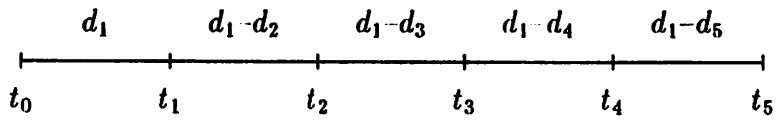
7.3.2 Partial Global Planning

Before getting into the details of how the PGPlanner computes the various attributes of PGPs, it may be helpful to get a higher-level view of the PGPlanner's activities. First, recall that planning and execution are interleaved in the node, so that the partial global planning mechanisms are invoked at intervals during problem solving. Whereas the local planning mechanisms are typically invoked after each action to monitor, repair, and update the local plan,[2] the partial global planning mechanisms could be invoked less often, depending on the dynamics of the problem and of the network. That is, if the environment, network, and node characteristics are relatively stable, then PGPs need not be altered often.

A node invokes the PGPlanner to find current plans and activities for the set of nodes that it is responsible for coordinating (based on the meta-level organization). Because it is working in a dynamic world, the PGPlanner should not completely map out every future action and interaction. However, because decisions about current activities can depend on projected future activities, the PGPlanner must develop a sufficient view of how nodes will act and interact to determine which of their possible activities nodes should pursue. The PGPlanner thus may consider several PGPs at a given time and project the future interactions between nodes involved in each as it makes decisions about which of its plans a node should pursue and whether it should alter that plan to cooperate more effectively.

When the PGPlanner is invoked, it begins by processing any node-plan and PGP messages and by updating node-plans of any changed local plans. If its network-model has changed, the PGPlanner scans the network-model to recognize new or modified PGGs, and creates or modifies PGPs as needed. The PGPs are stored on the PGP queue based on their ratings (where their ratings are a

[2]However, the mechanisms to update the clustering hierarchy and modify plans based on changing circumstances do not have to be invoked after each action.



Activity begins at time t_0. From t_0–t_1, the node works on data in d_1, from t_1–t_2 the node works on data in d_2 and combines it with past results to form d_1–d_2, and so on until it forms the result d_1–d_5 at time t_5.

Figure 66: An Example Node-Plan Plan-Activity-Map

function of the ratings of the participating node-plans). Unlike the local planner (or KSI dispatcher) that must only find the next plan (or KSI) for local problem solving, the PGP dispatcher must find a *set* of active PGPs because different PGPs can be pursued concurrently by different groups of nodes. Depending on its coordination responsibilities (represented in the meta-level organization), the PGPlanner is responsible for finding PGPs for some subset of the nodes in the network: if it is a coordinator, it must find enough PGPs so that each of the nodes it should coordinate is accounted for, while if nodes plan for themselves, then a node must scan through PGPs until it finds what it should currently be doing. Note, however, that even a node that only plans for itself may have to examine several PGPs because it must see whether the nodes it expects to cooperate with are also cooperating in other PGPs, and if so how that may affect this node's PGP.

For the most highly-rated active PGP, the PGPlanner first builds the PGP plan-activity-map to interleave the activities of the participating nodes. Each of the PGP's participating node-plans contains information about the local plan's i-goals, the order in which it will pursue these i-goals, and predictions about how much time is required for achieving each i-goal. The PGPlanner builds a plan-activity-map from this information: it assumes that the activities begin at the current time, and builds a sequence of plan-activities where the next plan-activity begins after the previous one ends. For example, a plan-activity-map for a node that builds a track with data d_1–d_5 is graphically depicted in Figure 66. The PGPlanner builds plan-activity-maps for each node-plan to recognize the concurrent activities of the nodes. An example is shown in Figure 67.

Next, the PGPlanner determines whether these activities could be reordered

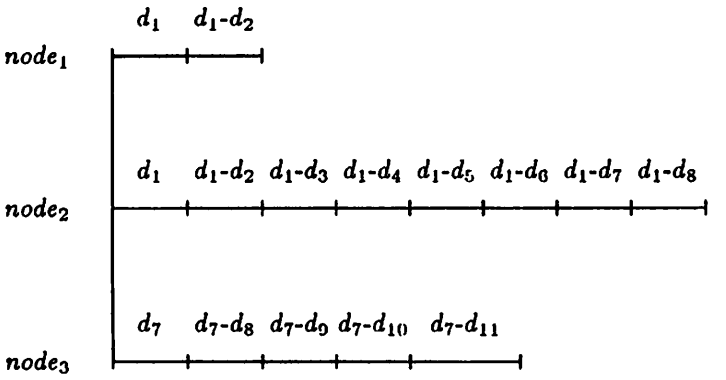


Figure 67: Concurrent Plan-Activity-Maps for Three Nodes.

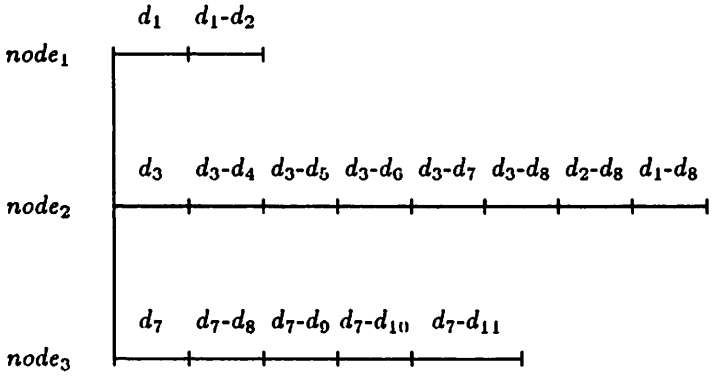


Figure 68: First Reordering of Concurrent Plan-Activity-Maps.

to improve network performance. These reordering decisions depend on the network's goals of cooperation: the PGPlanner applies knowledge about what the nodes should try to achieve through cooperation. For example, a goal of cooperation may be to avoid redundantly deriving partial solutions. In the example in Figure 67, $node_1$ and $node_2$ share activities d_1 and d_2, and $node_2$ and $node_3$ share activities d_7 and d_8. To avoid redundancy between $node_1$ and $node_2$, the activities of $node_2$ are reordered to move redundant activities to a later time (Figure 68). To avoid the possible redundancy between $node_2$ and $node_3$ (since the predicted durations of activities may be somewhat off), $node_2$'s activities can be further reordered to move d_7 and d_8 to later times (Figure 69). With this reordering, $node_2$ and $node_3$ work on their unshared activities first and then work toward their shared activities.

When it has found a reasonable ordering of plan-activities, the PGPlanner

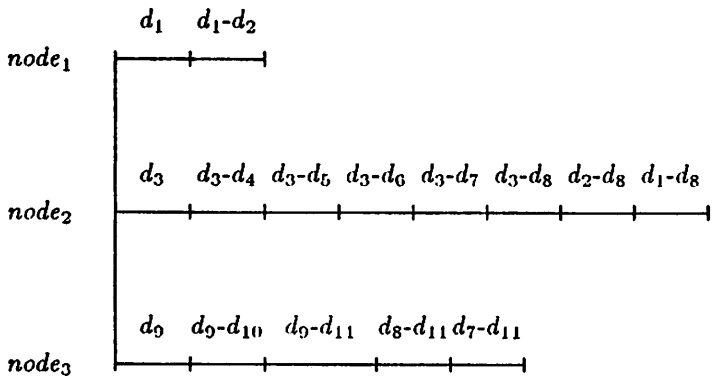


Figure 69: Final Reordering of Concurrent Plan-Activity-Maps.

groups sequences of activities into PGP-partial-solutions. It identifies when a sequence of activities all contribute to forming a larger partial solution, and explicitly represents this group. For example, given the plan-activities in Figure 69, the PGPlanner recognizes that $node_1$ will form partial solution d_1–d_2 over the interval spanned by both activities. Similarly, it recognizes that $node_2$ will form d_3–d_7 (its other activities work on data that has already been processed) and $node_3$ will form d_{11}–d_8 (d_7 will have already been covered by $node_2$). These partial solutions and their intervals are shown in Figure 70. With this representation, the PGPlanner can then determine how the various partial solutions should be integrated into complete solutions. Since $node_1$ completes its partial solution soonest, it sends it off to $node_2$ (so that $node_2$ continues working on its partial solution while the partial solution from $node_1$ is in transit). Similarly, $node_3$ completes its partial solution before $node_2$ and so should send that partial solution on to $node_2$. When $node_2$ has completed its partial solution, it will have received or will be about to receive the other partial solutions and can integrate the pieces into an overall solution.

The PGPlanner thus proposes how the nodes can construct a solution, shown graphically in Figure 70. With this view, the PGPlanner can also reason about what hypotheses need to be exchanged and where, so communication decisions can be made more intelligently. The PGPlanner explicitly represents as part of the PGP the basic specifications of hypotheses to transmit, what nodes should send them, and what nodes should receive them. Once this is complete, the PGPlanner has roughly mapped out how the nodes will act (what activities they

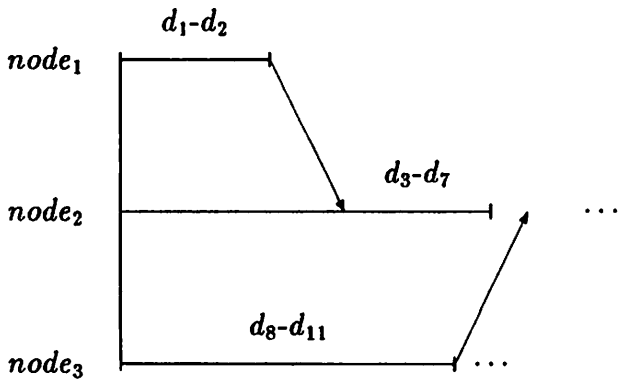


Figure 70: Proposed Solution Construction Activities.

take and approximately when) and how the nodes will interact (what hypotheses they exchange and what nodes integrate partial solutions).

This simple view of the PGPlanner's activities avoids several important issues, especially the dynamics of PGPlanning (since planning is interleaved with execution), the interactions between PGPs that may concurrently be active, and how the PGPlanner recognizes when tasks should be transferred. First of all, the view of how the PGPlanner constructs the initial plan-activity-map is fine when problem solving begins, but things get more complicated when the nodes have progressed on some plans and not others and the PGPlanner must represent the past, present, and future plan-activities for node-plans. To avoid recomputing plan-activity-maps for participating node-plans, the PGPlanner stores the plan-activity-map that it generates for a node-plan with that node-plan. Thus, for plans that do not change, the PGPlanner can simply extract the plan-activity-map for the node-plan and use it to form a PGP's plan-activity-map. However, when a node changes its plans (new information arrives, actions fail, time predictions are modified), these modifications are propagated to its node-plans and the node-plans will be transmitted to any pertinent nodes (based on the meta-level organization). Included in the modifications to a node-plan is an updating of the plan-activity-map (if any has yet been formed). The PGPlanner removes from the plan-activity-map any *future* plan-activities which should be recomputed. Any past plan-activities are left in the plan-activity-map since the past cannot be changed.

Therefore, when building a plan-activity-map for a PGP, the PGPlanner may

have some plan-activities as part of a node-plan's plan-activity-map, and must compute the others. The past plan-activities act as historical information so that the PGPlanner knows which activities have already been completed and which remain. The plan-activity-map it produces therefore may include both past and future plan-activities. A node that forms a PGP and updates a node-plan for one of its own local plans can send out the node-plan's plan-activity-map with that node-plan. Although this increases communication costs (more information to transmit), it saves on computation costs (since the recipient node's PGPlanner need not derive the plan-activity-map itself) and it better reflects the actual times when plan-activities will occur (since the node that is actually performing the local plan knows more exactly when activities for the plan will be pursued).

The second important issue is dealing with several active PGPs. Since PGPs are rated, the PGPlanner considers them one at a time starting with the most highly rated. However, this does *not* mean that all nodes involved in the most highly-rated PGP should currently follow plans associated with that PGP. Some of the nodes may have completed their local plans that participate in the PGP, and so the PGPlanner must look at other PGPs to determine what these nodes should be doing. Furthermore, it may be advantageous to delay activities by some of the nodes working on the best PGP so that they can form important results for other PGPs (such as predictive results). The PGPlanner must determine whether such activities exist for a node and whether they can be pursued without affecting the more highly-rated PGPs.

For example, consider the situation in Figure 71. In this situation, $node_1$ participates in two PGPs: to form track d_1–d_{10} and to form d'_1–d'_{10}. Because the data for d_1–d_{10} is more strongly rated, the plans of $node_1$ and $node_3$ to develop their partial solutions are highly rated and the PGP to form the overall solution has a higher rating than the PGP to form d'_1–d'_{10}. The PGPlanner develops this PGP first, and identifies the partial solutions being formed by $node_1$ and $node_3$ (Figure 72). From this information, the PGPlanner recognizes that $node_1$'s partial solution is received by $node_3$ well before $node_3$ needs it: the partial solution provides no predictive information (since $node_3$'s data has little ambiguity), and therefore need not arrive at $node_3$ until $node_3$ has formed the other partial solution to integrate with it.

The PGPlanner treats the time between when $node_1$ begins to form its partial

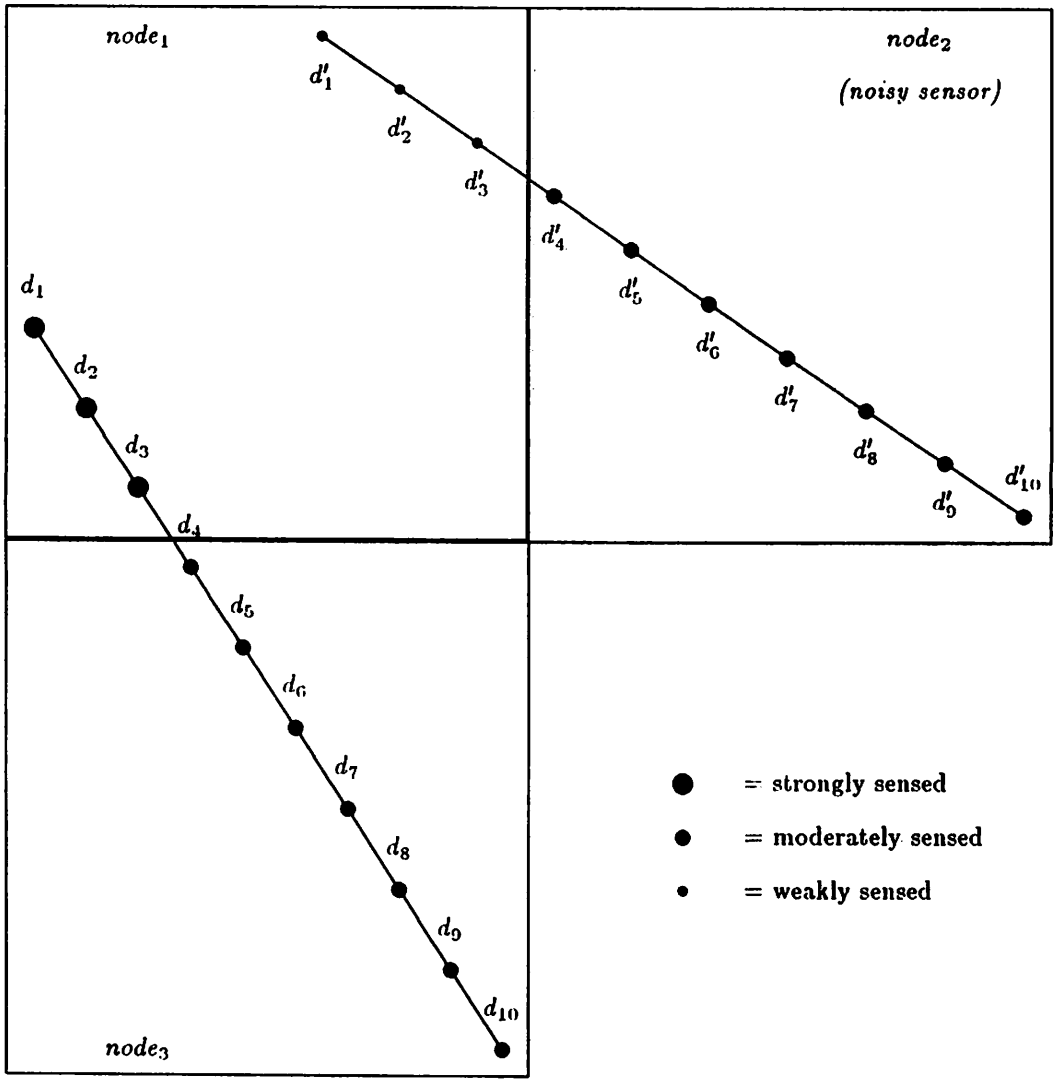


Figure 71: An Example With Alternative PGPs.

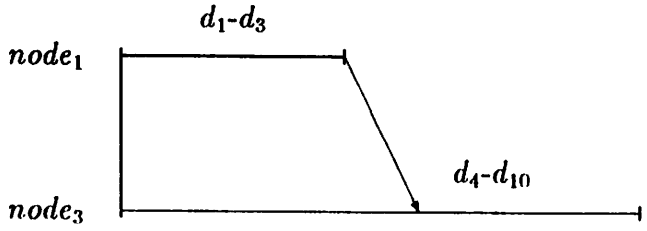


Figure 72: Proposed Solution Construction for Better PGP.

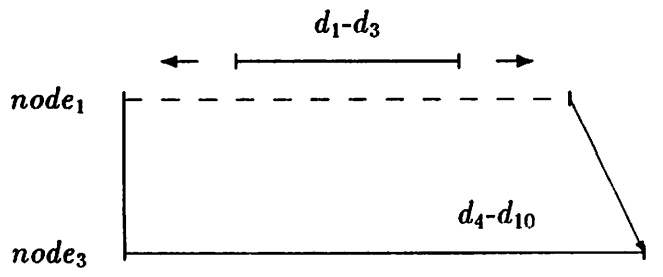


Figure 73: Time-Window for Node 1's Activities.

solution and when it must send this partial solution to $node_3$ as a time-window (Figure 73). Because the PGP is the most highly rated and should thus be achieved as quickly as possible, the PGPlanner must plan to pursue the activities that generate this partial solution within the time-window, but it can use any extra time to pursue other PGPs. When it examines its other PGP (to form d'_1-d'_{10}), it recognizes that $node_1$ may provide predictive information to $node_2$. As a result, the PGPlanner decides to postpone $node_1$'s activities on the more highly-rated PGP,[3] and will perform activities for the more lowly-rated PGP until it has used up the available time (Figure 74). Hopefully, it can perform enough activities to form and send predictive information to $node_2$ before it must work on its activities for the more highly-rated PGP. If not, it still has not lost anything since it still forms and transmits the partial solution for the more highly-rated PGP so that $node_3$ receives it by the time it is needed. Thus, a PGP's solution-construction-graph not only represents how nodes should exchange partial results but also represents the flexibility they have in their local activities while still interacting effectively with others. The PGPlanner exploits this flexibility when possible.

The third important issue to address is task-passing. The PGPlanner as outlined above determines how nodes should pursue their activities and exchange partial solutions to improve cooperation. However, it did not consider how cooperation might be further improved if nodes transferred activities to make better use of their computational resources and their expertise. The PGPlanner does

[3]Because the predicted durations of the activities might be somewhat off, the PGPlanner can add an extra time cushion to the expected time needs for forming the partial solution to avoid the situation where a node exceeds the window.

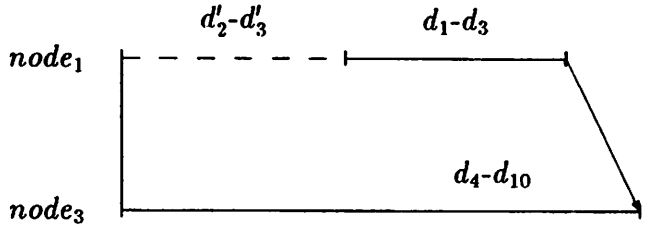


Figure 74: Modified Solution Construction for Better PGP.

make some simple decisions about whether tasks (groups of related activities) should be transferred for better load balancing, and uses the meta-level organization to find nodes with appropriate expertise for these tasks. Once again, the PGPlanner bases these decisions on a PGP's solution-construction-graph: it identifies whether there is a *bottleneck-node* that takes substantially longer to form its partial solution than the other cooperating nodes. If it finds such a node and can find nodes that are underutilized (based on its network-model) and that could potentially perform some of the bottleneck-node's activities, then it modifies the PGP to represent the possible transfer of tasks. This PGP is sent to the underutilized nodes, and they respond with node-plans that the original node compares to find which (if any) of the nodes should receive the tasks.

The task-passing mechanisms are more completely described in Section 7.3.8. It is important to note, however, that the simplified view of task-passing that they use is limited. They are unable recognize situations where an *exchange* of tasks between two nodes would improve network problem solving. Identifying and performing task swaps of this type is certainly possible within our framework, however, since nodes could use their network-models to determine whether one node has expertise that would allow it to better perform another node's tasks while at the same time the other node would be better able to perform the initial node's tasks. The current task-passing mechanisms move tasks to nodes with available resources or expertise, but do not yet have the capability to move resources or expertise to nodes with tasks needing them. In the future, we hope to extend the task-passing mechanisms to provide these capabilities (see Chapter 9).

7.3.3 Generating a Partial Global Plan Plan-Activity-Map

Given a set of active PGPs, the PGPlanner processes them one at a time, starting with the most highly-rated one.[4] In general then, when the PGPlanner processes a PGP, it may have already processed some set of more highly-rated PGPs, and these PGPs will affect the PGP currently under consideration. The mechanisms for generating a PGP's plan-activity-map therefore go through three phases. First, an initial plan-activity-map is formed by interleaving the plan-activities of the participating nodes. Next, this plan-activity-map is modified based on the more highly-rated PGPs: because a node that participates in this PGP may also participate (with some other set of nodes) in more highly-rated PGPs, the PGPlanner must check to see whether this PGP's plan-activity-map assumes a node will be pursuing tasks during times that it has committed to other PGPs. If so, the PGPlanner changes the expected begin-times and end-times of these plan-activities to the more distant future where there is no conflict. Finally, the PGPlanner determines whether the plan-activities could be reordered to improve group problem solving, and if so, generates a new, improved plan-activity-map.

Forming the Initial PGP Plan-Activity-Map

The first step in processing a PGP is to build its plan-activity-map. For each of the participating nodes, the PGPlanner finds a sequence of plan-activities for that node (whenever possible). These plan-activities can come from any of several sources. If the node-plan for the participating node is not in its node-model, then the PGP must have been received (this node could not have identified the PGG involving that node). In this case, the plan-activity-map for the node is extracted from the PGP's plan-activity-map: the node that sent the PGP could have sent the entire plan-activity-map with it. If the plan-activities for this node were not included in the received PGP message, then the PGPlanner cannot compute the PGP's plan-activity-map. In this case, the PGPlanner cannot locally modify the PGP but must follow it exactly as received. By holding back plan-

[4]The PGPlanner works from the best PGP down, reasoning about one before considering the next. Because it does not consider several PGPs together, it might not find an optimal arrangement of coordinated activities, but such an optimal view would be much more expensive to develop (especially in a dynamic domain where it must be recomputed when things change).

activity information, therefore, the node sending the PGP can enforce adherence to the PGP since a recipient lacks sufficient context to develop alternative ways of achieving the PGP.

If the node-plan for the participating node is in its node-model, then it is used to generate the plan-activity-map. The PGPlanner may have already formed a plan-activity-map for the node-plan and stored it with the node-plan, so it simply must retrieve this plan-activity-map. Similarly, the node that formed the node-plan (from a local plan) may have computed the plan-activity-map and included it in the node-plan message, so again the PGPlanner simply retrieves the plan-activity-map. If the plan-activity-map for the node-plan does not already exist, however, then the PGPlanner must compute it.

To compute a plan-activity-map, the PGPlanner uses three pieces of information: the past activities for the node-plan, the order in which it will pursue i-goals, and the time-predictions for these i-goals. If a node has already completed some activities, then these are permanently part of its plan-activity-map (since the past cannot be altered). Therefore, if a plan was pursued in the past but, before it was done, the node began working somewhere else, then with the plan's node-plan is associated any completed plan-activities: when the planner begins pursuing another plan, it removes any predicted future plan-activities from the initial node-plan's plan-activity-map (since these predictions were based on pursuing the node-plan without interruption). Thus, a record is kept of what has been achieved in the past. The long-term:time-order provides predictions about the order in which the remaining i-goals will be pursued, and the long-term:time-predictions roughly estimate how long each of those i-goals will take. The calculation is thus:

1. Examine the node-plan's plan-activity-map to determine what i-goals have been completed already (if any);

2. Set the time-to-begin-i-goal to the current-time;

3. For the remaining i-goals in order:

   (a) for the next i-goal, build a plan-activity and set:

      i. the begin-time to the time-to-begin-i-goal,

ii. the end-time to the begin-time plus the predicted time needed to achieve the i-goal,

iii. the area-of-focus to the i-goal's sensed time and corresponding region,

iv. the expected-result to the i-goal's new time-region combined with any previously pursued i-goals expected results that are compatible (at adjacent times);

(b) the time-to-begin-i-goal is set to the end-time of this plan-activity;

4. The new plan-activity-map is appended onto the past plan-activity-map (if any).

When a node calculates the plan-activity-map for other nodes' node-plans, therefore, it estimates that the plan will be pursued starting at the current time or at a slightly later time based on communication delays. If nodes have exchanged node-plans, this corresponds to having one node assume that others are currently seeing the same situation and will therefore form their plan-activity-map at the current time. If the node is coordinating others, the plan-activity-maps estimate when the actual activities will be pursued based on when the PGP could be received by others. In more general terms, whenever a node first recognizes some PGG and develops a PGP to achieve it, the node cannot be completely aware whether other nodes are simultaneously recognizing and forming equivalent PGPs. The plan-activity-map that one node forms for another thus represents an estimate of that node's activities, and inconsistencies between how nodes view coordination can result from differing estimates. Rather than building in synchronization mechanisms to prevent such inconsistencies, the PGPlanning mechanisms allow nodes to cooperate despite them and to exchange information to converge on more consistent PGPs.

The PGPlanner interleaves the plan-activity-maps for the participating node-plans into a single plan-activity-map for the PGP. It interleaves them based on the end-times for the plan-activities (and when plan-activities end at the same time, the name of the node performing the activity is used to order them). By ordering the plan-activity-map based on end-times, the PGPlanner can step through the plan-activity-map and develop a view of which partial solutions will be developed before others and of the relative times of their development.

Modifying a PGP Plan-Activity-Map Using Better PGPs

The PGPlanner gives priority to the most highly-rated PGPs: if a node can participate in several PGPs, then preference is given to node-plans that pursue the more highly-rated PGPs. After it forms the plan-activity-map for the PGP, therefore, the PGPlanner must check this plan-activity-map against the plan-activity-maps of the more highly-rated PGPs to recognize situations where the same node is expected to be pursuing two different node-plans at the same time. When such a conflict occurs, the more highly-rated PGP takes priority, and the plan-activities for the less highly-rated PGP are usually moved to non-conflicting times. However, the plan-activities for the more highly-rated PGP can be moved in certain circumstances.

Given a current PGP and a previously processed (more highly-rated) PGP whose plan-activity-maps conflict, the PGPlanner first determines whether the plan-activities of the better PGP can be moved without penalizing that PGP. Because the PGP represents how and approximately when nodes will cooperate by exchanging partial solutions, the PGPlanner examines the PGP to discover whether the node in question has available time in the PGP. It does this by expanding the PGP's solution-construction-graph to represent the greatest amount of flexibility (we describe how the solution-construction-graph is formed in Section 7.3.4). An example of this was shown in Figure 74, where, starting from the earliest possible integration of results, we worked backwards determining when the latest a part of the overall results could be formed without affecting the integration time. We represent the flexibility nodes have in when they develop partial solutions as *time-windows* [Vere, 1983]. A PGP-partial-solution structure stores the begin-time, end-time, and duration of the activities to generate the partial solution. The time-window for a PGP-partial-solution is represented by its begin-time and end-time: if its duration is less than the interval between the begin-time and end-time, then the activities can be moved around within that window to improve flexibility, as was shown in Figure 74. When a partial solution is not needed until later than initially planned, its begin-time and end-time are changed so that it will complete when it is needed. Once this has been done, then the end-times for the bottom ("leaf") PGP-partial-solutions can be set to a later time.

Determining an optimal arrangement of activities within windows is a com-

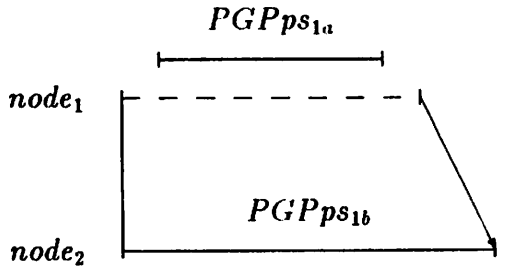


Figure 75: Flexibility in Better PGP.

plex (and costly) process. This complexity is compounded by the fact that windows change depending on the scheduling of PGP-partial-solutions, and the nodes where some activities (such as integrating partial solutions) are performed can vary depending on how PGP-partial-solutions are scheduled. For example, in Figure 75, the time-windows and activities for one PGP are shown and in Figure 76 the time-windows and activities for a less highly-rated PGP are shown. Each of these representations was developed independently, but now they must be combined to determine which PGP-partial-solution $node_1$ should pursue. If $node_1$ pursued $PGPps_{2a}$ first, then it could not complete $PGPps_{1a}$ within the bounds of its time-window. To guarantee that the results for the better PGP are formed as quickly as possible, therefore, $PGPps_{1a}$ should be pursued first. However, this delay for $PGPps_{2a}$ may change how other nodes should coordinate to complete the PGP. For example, in Figure 77, moving $PGPps_{2a}$ to a later time means that integration of partial solutions should be done at $node_1$, and now $node_3$ has available time. This change to $node_3$ may mean that other PGPs may have to be checked (and possibly changed), and propagating the effects of arranging activities on a node can therefore be extremely complicated and costly.

For these reasons, and for the underlying assumption that unexpected changes to the situation may occur that will make any optimal arrangement void, the PGPlanner uses a much simpler approach to scheduling activities within their windows. It begins with the premise that the only reason for moving a highly-rated PGP's activities to a later time is if the activities *do not* generate predictive results (so generating them earlier will not help other nodes) and if the activities of inferior PGPs do. The algorithm is thus:

if the most highly-rated PGP is generating predictive results or no more lowly-

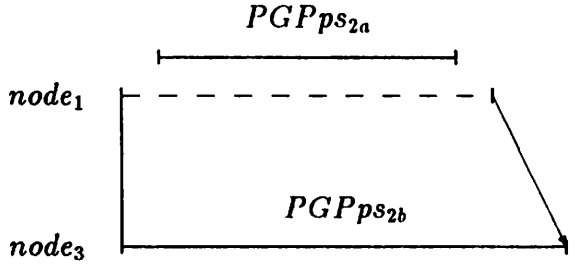


Figure 76: Flexibility in Other PGP.

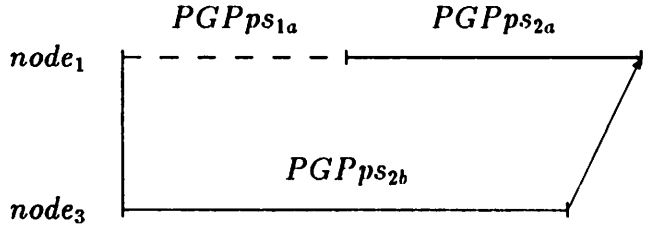


Figure 77: Modified Solution Construction For Other PGP.

rated PGPs are generating predictive results:

then the activities are scheduled to begin right away;

else the activities are scheduled to begin at the latest possible time (so that they end at end-time).

The activities for subsequent PGPs are fit in around assigned times for previous PGPs. Thus, given a conflict between two PGPs, the more lowly-rated PGP moves its activities to non-conflicting times. Since the previous PGPs have set particular times for their activities (and these times may not necessarily begin immediately), the later PGPs leave any non-conflicting activities alone (so that some may be pursued and perhaps predictive results may be generated before the more highly-rated PGPs take over). The end result is that a PGP's plan-activity-map is altered based on the previously pursued PGPs to reflect when nodes are likely to pursue the activities considering how they are already tentatively committed to cooperating with others. Because the plans, goals, and data of nodes change over time, however, this tentative postponing of some actions in

favor of others can be changed when new information arrives and the activities and ratings of PGPs change.

Reordering the PGP Plan-Activity-Map to Improve Cooperation

The PGP plan-activity-map represents the concurrent activities of nodes as they are currently perceived. However, the order in which nodes locally decide to pursue activities (intermediate-goals) may be inappropriate from the more global context perceived by the PGPlanner. For example, by pursuing i-goals in a particular order, a node might quickly generate a partial result that it can pass to another node to provide predictive information. Without the more global view represented in the PGP, the node would not recognize this possibility for improving network activity, and may therefore order i-goals in a less effective order. As another example, two nodes may share common data, and only one of them should process it to cover that data while the other processes other important data. Without the more global view from the PGP, the nodes could not recognize and avoid this potential duplication of effort. Once it has formed PGPs, therefore, a node must examine the order in which nodes locally decided to pursue their activities and rearrange these activities when by doing so network problem solving will be improved.

To reorder the PGP plan-activity-map, the PGPlanner needs to recognize which activities are more important (or less costly): it must find the cost of the various plan-activities in the plan-activity-map, and move less costly activities earlier in the plan-activity-map. The PGPlanner considers cost to be the opposite of rating. The reason that costs are found for activities rather than ratings is that, to simplify the mechanisms, there is no upper limit on costs (ratings). The lower limit is 0—corresponding to a cost of 0. This is particularly useful for plan-activities that have already been completed since any reordering should not move them: by giving completed plan-activities a cost of 0 (since once completed they cost nothing to the nodes), they are never moved to later times since there are no less costly plan-activities possible.

The PGPlanner computes a cost for each plan-activity. If the plan-activity is completed, its cost is 0. Otherwise, the PGPlanner performs the calculation:

$$c_{pa} = (rdf \times rd) + (rlf \times rl) + (drf \times dr) + (prf \times pr) + (lpf \times lp) + (inf \times in) + (dvf \times dv)$$

where *rdf* is the redundancy factor, *rlf* is the reliability factor, *drf* is the duration factor, *prf* is the prediction factor, *lpf* is the locally-predicted factor, *inf* is the independence factor, and *dvf* is the diversity factor. Each of these is specified in the cooperation-parameters (as part of the meta-level organization). The redundancy value *rd* is computed as the number of nodes that could perform the activity specified in the plan-activity (that have the data for the specified area). Thus, the redundancy cost increases as more nodes are able to perform the activity. In contrast, the reliability value *rl* is the total number of nodes minus the number that could perform the activity, and so the reliability cost decreases as more nodes are able to perform the activity. The duration value *dr* is simply the duration of the plan-activity (the difference between its end-time and begin-time), and the duration cost increases with duration.

The predictiveness value *pr* is computed by comparing the plan-activity with the uncompleted plan-activities for other nodes and determining whether the plan-activity might provide the others with predictive information (there is no point in providing predictive information for completed activities). This decision is based on the durations of the plan-activities: a plan-activity that has a shorter duration may provide predictive information for an plan-activity with a longer duration (assuming that the longer duration is because of ambiguity). For each such plan-activity, the PGPlanner determines the *activity-distance* between them. This distance is equal to the absolute value of the difference between the sensed times for the data they work on. For example, if one plan-activity works on data d_1 (sensed at time 1) and the other on data d_4 (sensed at time 4), then the activity-distance between them is 3. If either or both plan-activities process data for several sensed times, then the minimum distance between these sensed times is used. The minimum activity-distance between the plan-activity and the others that it can provide predictive information for is used as the predictiveness value. If it does not provide predictive information at all than the maximum possible activity-distance (difference between the first and last sensed time for the PGP's objective track) is used. Thus, the closer a plan-activity is to plan-activities that need its predictive information, the lower its predictiveness cost.

The locally-predicted value *lp* is computed by comparing the plan-activity with *earlier* plan-activities for the same node. If there are none, then the value is 0. Otherwise, the minimum activity-distance between these plan-activities and

the plan-activity whose cost is being computed is calculated. This minimum is used as the locally-predicted value: the closer the plan-activity is to previous plan-activities performed by the same node, the lower the locally-predicted cost. Note, furthermore, that when plan-activities are reordered this value can change because a plan-activity might have different previous plan-activities.

The independence value *in* is computed as how many plan-activities for the node are performed before this plan-activity *in the initial plan-activity-map*. For a given node, later plan-activities have a higher independence cost than earlier ones do, and these costs are constant whatever the reordering. The independence cost thus represents a form of inertia, preventing the PGPlanner from finding an ordering far from the original ordering and therefore minimizing the PGPlanner's effects on local plans.

Finally, the diversity value *dv* is computed by comparing the plan-activity with earlier plan-activities in the plan-activity-map and determining whether it will derive redundant results. If not, the diversity value is 0. Otherwise, the PGPlanner examines the later plan-activities for the same node and determines if any do not represent redundant activity. If all are redundant, then the diversity value is 0 (since nothing better can be found), but if some are not redundant, than the activity-distance between this plan-activity and those are computed and the minimum is used as the diversity value. A plan-activity thus has a higher diversity cost when it is redundant and farther away from non-redundant plan-activities for the same node. Note once again that the diversity cost of a plan-activity can change when the plan-activity-map is reordered because it is based on which plan-activities are performed earlier and which later.

The PGPlanner works its way down the plan-activity-map and computes the total cost for each plan-activity (c_{pa}). It computes the cost for the entire map as the sum of the plan-activities' costs. It then iteratively attempts to reorder the plan-activity map until it cannot find a better one, using the hill-climbing algorithm:

1. **for** the current (initial) plan-activity-map
   (a) compute the cost of each plan-activity
   (b) compute the current-plan-activity-map-cost as the sum of the plan-activity costs

2. **for** each plan-activity in the current plan-activity-map

   (a) **examine** the later plan-activities for the same node to find the one with the lowest cost

   (b) **if** the lowest cost later plan-activity has a lower cost than the current plan-activity, then swap them in the new plan-activity-map

3. **if** the new plan-activity-map is different from the current plan-activity-map then

   (a) compute the cost of each plan-activity in the new plan-activity-map

   (b) compute the new-plan-activity-cost as the sum of the plan-activity costs

4. **if** the new plan-activity-map is different from the current plan-activity-map and the new-plan-activity-cost is lower than the current-plan-activity-cost

   then set the current plan-activity-map equal to the new plan-activity-map and the current-plan-activity-cost to the new-plan-activity-cost and go to step 2

   else return the current plan-activity-map.

Since a new plan-activity-map must have a strictly lower cost than any earlier versions of the plan-activity-map, and because the same plan-activity-map will have the same overall cost if encountered twice in the algorithm, the algorithm cannot get into indefinite cycle. Also, because the number of possible orderings of a plan-activity-map with a set number of plan-activities is finite, the algorithm must terminate. In fact, the algorithm generally needs to iterate only a few times (less than 5 in our experiments) before it generates a plan-activity-map that is not an improvement over earlier ones. The number of iterations is small because each iteration can swap any number of plan-activities: if instead it were constrained to swap only one pair at each iteration, then many more iterations might be needed. By swapping several pairs in an iteration, the algorithm might miss a locally optimal ordering because it would "overshoot" that peak (if the good swaps are outweighed by the bad). The more conservative approach of swapping one pair per iteration is less likely to overshoot the optimal ordering, but is also much more

expensive because so many more iterations are needed. The computation of costs for the plan-activities can be expensive (since comparisons between each pair of plan-activities must be done) so reducing the number of iterations is important. The less conservative approach is thus used in the implementation.

Even though the algorithm attempts to minimize the number of times that the costs of plan-activities must be determined, the expense of this calculation must be reckoned with. Because the cost of a plan-activity depends on what plan-activities precede it and which succeed it, computing this cost means that, in the worst case, a plan-activity must be compared with every other plan activity. Thus, the complexity of computing the cost for a plan-activity-map increases quadratically with the number of plan-activities in that plan-activity-map. For a sizable plan-activity-map containing on the order of one hundred plan-activities, this complexity can make reordering the plan-activity-map very expensive. In the experiments studied in this dissertation, these costs are still acceptable, but future research will need to deal with this issue.

Summary of the Mechanisms for Forming a PGP Plan-Activity-Map

To summarize, the PGPlanner uses information from the PGP and from any local node-plans to generate an initial plan-activity-map that interleaves the plan-activities of the participating nodes. It alters the begin-times and end-times of certain plan-activities based on the other active PGPs to generate a plan-activity-map that does not interfere with other PGP's plan-activity-maps. It then iteratively reorders the plan-activity-map by computing costs for the plan-activities, returning the plan-activity-map with the lowest total cost. Whenever the plan-activity-map for a PGP is reordered, it indicates that some local plans should change: some nodes should pursue activities in a different order so that the network as a whole can cooperate more effectively. For any local plans associated with such a PGP reside at the node, the PGPlanner updates those local plans to reflect the reordering. It compares the plan's long-term:time-order with the order indicated in the PGP's plan-activity-map. If these differ, the plan's long-term:time-order is changed to match the PGP's order and, if the plan's next i-goal has changed, the plan's short-term actions are updated as well.

A PGP's plan-activity-map is used by the PGPlanner to compute the other attributes of the PGP, including its solution-construction-graph and its com-

munication actions. Furthermore, the PGPlanner modifies the node-plans that participate in the PGP: for each node-plan, it extracts the plan-activities for that node and stores them with the node-plan. Thus, the next time the PGPlanner must generate a plan-activity-map for the PGP it can simply extract the plan-activity-map for the node-plan and interleave it with the other plan-activity-maps for the other node-plans. The PGPlanner might also transmit this node-plan to inform other nodes of the changes, particularly if the node-plan corresponds to a local plan for the node that altered it.

More generally, these mechanisms reason about the activities being pursued concurrently at several nodes, and use knowledge about what types of activities should occur concurrently (or precede others) to improve cooperation. This view is used by subsequent mechanisms that determine how nodes can interact to achieve network goals.

7.3.4 Building a Solution-Construction-Graph

After it has determined what activities the cooperating nodes should perform and when, the PGPlanner next must determine how the nodes will interact to form the overall solution. By roughly estimating future interactions, the PGPlanner recognizes cases where some nodes have extra time before they need to form their piece of the overall solution, and the mechanisms described in the last section use this view to decide whether in fact a node should pursue an inferior PGP because it has extra time before it must generate its pieces of the overall solutions for better PGPs. The PGPlanner also uses information about interactions when deciding what partial solutions should be exchanged between nodes and what tasks should be transferred between nodes.

The potential future interactions between nodes are represented as a solution-construction-graph. There are two principal phases in building a solution-construction-graph for a PGP. First, the PGPlanner must identify the pieces of the overall solution that are being formed by the participating nodes, using the plan-activity-map to roughly predict when those pieces will be formed. In the second phase, the PGPlanner determines where and when these separate pieces should be joined into larger pieces, until the entire solution is formed. These two phases are addressed in the next two sections.

Finding the Initial PGP-Partial-Solutions

In the first phase of building the solution-construction-graph, the PGPlanner must decide what pieces of the overall solution built by nodes are relevant and when they will be formed. The relevance of a partial solution depends on the goals of cooperation, which are represented as parameters in the cooperation-parameters data structure. For example, if the redundancy cost is high, then the PGPlanner should avoid having nodes provide redundant partial solutions. In this case, an activity by a node is only relevant if it provides results that no other node has yet generated. On the other hand, if the reliability cost is high, then the PGPlanner should promote the derivation and exchange of duplicate results.

The PGPlanner begins by scanning down the plan-activity-map and removing irrelevant plan-activities. In the current implementation, a plan-activity is irrelevant if it generates results that a previous activity (at another node) has already generated *and* if the redundancy cost is greater than the reliability cost. This allows the PGPlanner to let nodes avoid informing each other of results that they have already derived or received from elsewhere. When rederivation is important, on the other hand, the PGPlanner does not remove these activities since nodes should exchange results covering the same areas in order to compare their results (and have higher confidence if their results agree). In addition, the PGPlanner does not remove a plan-activity that generates results that were formed earlier somewhere else if it determines that the earlier results could not arrive at the node performing the later activity before the later activity is finished. That is, communication delays might mean that a node could locally form a duplicate result sooner than if it waits for that result sent by another node.

With the possibly filtered plan-activity-map, the PGPlanner next groups together related activities being performed by the same node. For example, if a node performs a sequence of activities to form a single partial solution, then these activities are grouped together (see Figures 69 and 70 for and example). For each node, any number of such groups are possible, each corresponding to a separate partial solution that cannot be combined with any other local partial solutions (otherwise their combined activities would be grouped together). Since some plan-activities may be removed to avoid redundancy, some nodes may not have any groups at all.

Each group of activities is used to form a PGP-partial-solution. First, the

overall partial solution of the activities is represented as an expected result (the rough time-region specifications of the track that the activities as a whole are expected to form). Next, this expected result is compared to any PGP-partial-solutions that have already been developed for the PGP to determine if another node is already forming these results. When no such PGP-partial-solutions exist, then a new PGP-partial-solution is formed for the group of activities. If such PGP-partial-solutions exist, then the PGPlanner uses the solution-construction-redundancy to decide whether to form a PGP-partial-solution for this group of activities or whether they are already being performed elsewhere. If generating a new PGP-partial-solution will not exceed the solution-construction-redundancy (the total number of PGP-partial-solutions for the expected result, including the possible new PGP-partial-solution and any matching existing PGP-partial-solutions, is not greater than the solution-construction-redundancy), then a new PGP-partial-solution is formed. Alternatively, if the possible new PGP-partial-solution would represent too much redundancy, then its expected end-time (the end-time of the last plan-activity in the group) is compared to the existing PGP-partial-solutions' end-times. When the existing PGP-partial-solutions' end-times are much earlier than the new group's end-time, then no PGP-partial-solution is formed for the new group, while if the group's end-time is much earlier than any of the existing PGP-partial-solutions' end-times, then the existing PGP-partial-solution with the latest end-time is replaced by a new PGP-partial-solution for the new group. Finally, if the new group's and existing PGP-partial-solutions' end-times are within the time-cushion specified in the cooperation-parameters, then the new group replaces one of the PGP-partial-solutions if the new group of activities is being performed on a less congested node.

To decide whether one node is less congested than another, the PGPlanner examines the PGPs that it knows about. It first looks at any PGPs that are more highly rated than the current PGP, and uses their solution-construction-graphs to determine which of the nodes is less highly used by those PGPs. That is, it determines when each node is expected to be performing activities as part of those PGPs, and the node that completes its activities earlier is less highly used. If no more highly-rated PGPs exist, or if both nodes complete their activities at the same time, then the PGPlanner examines less highly-rated PGPs. Since these may not yet have solution-construction-graphs (and even if they do, their

solution-construction expectations can be voided if by doing so the PGPlanner can improve how nodes pursue the more highly-rated PGP), the PGPlanner scans the PGPs until it finds one where only one of the two nodes participates. If it finds such a PGP, the PGPlanner assumes that the other node is less highly utilized (pursues less highly-rated PGPs) and assigns responsibility for the partial solution to that node. Finally, if the PGPlanner cannot choose between the nodes on this basis (they participate in the same PGPs), then it arbitrarily chooses one of them (the one with the higher number in the current implementation).

Forming the Solution-Construction-Graph from Initial PGP-Partial-Solutions

By forming a set of PGP-partial-solutions for groups of activities, the PGPlanner generates the initial set of partial solutions. It next determines when and where these partial solutions should be combined to eventually form complete solutions. This is performed by an iterative procedure:

initialize the PGP-partial-solution-set to the initial set

while the PGP-partial-solution-set contains two or more elements

> **for** each pair of entries in the PGP-partial-solution-set, find the earliest time they can be combined and at what node
>
> **for** the pair with the earliest combination time
>
> > **form** a new PGP-partial-solution for their combination and add it to the PGP-partial-solution-set
> >
> > **remove** the PGP-partial-solutions that have been combined from the PGP-partial-solution-set depending on the solution-construction-redundancy

return the single entry in the PGP-partial-solution-set

Where the initial set of PGP-partial-solutions comes from was described in the previous section. The PGPlanner iterates through the above procedure attempting to combine pairs of PGP-partial-solutions as it forms larger and larger PGP-partial-solutions. It starts by considering each pair of PGP-partial-solutions in its set, and for each pair finding when and where that pair could be combined. To

make this calculation, the PGPlanner uses the end-times of the two PGP-partial-solutions, information about communication-delays between nodes, and models of other nodes' time needs for combining results (the expected-integration-costs associated with the node in its node-model). Using the communication topology, it determines which nodes could possibly receive the two partial solutions to combine. For each of these, the PGPlanner determines the earliest that the possible combining node could receive the partial solutions (based on the end-time of each PGP-partial-solution and communication-delays to send it to the possible node). It then determines when the combined partial solution could be formed as the time that the later partial solution arrives plus the node's time needs for combining results. If only one of the possible nodes is capable of combining the results, then this node and the combination time is returned. When several nodes could combine the results, all at about the same time (determined by the time-cushion), then the procedure outlined in the previous section is used to find the least utilized node and assign the integration activities to it. In this case, the procedure takes an additional step before choosing one of the nodes arbitrarily. If the nodes participate equally in the PGPs, then preference is given to nodes that are forming the partial solutions that must be combined: when one of these nodes can receive the other partial solution and form the combination, then less communication is needed. If both such nodes could combine the partial solutions, then one of these is chosen arbitrarily.

After the PGPlanner accumulates possible combination times and nodes for pairs of PGP-partial-solutions, it acts on the pair that can be combined earliest (or one of them in case of a tie). First, it forms a new PGP-partial-solution for the combination, where this new PGP-partial-solution gets its node and end-time (combination time) from the information for the pair. The duration is the node's time needs for combining results and the begin-time is the difference between the end-time and duration. The PGP-partial-solution's result is the combined time-regions of the pair of PGP-partial-solutions. The new PGP-partial-solution points to the two combined PGP-partial-solutions as supporting partial solutions, and they in turn point to this new PGP-partial-solution as a supported partial solution. The new PGP-partial-solution is then added to the set of PGP-partial-solutions.

The set of PGP-partial-solutions is further updated by possibly removing the PGP-partial-solutions that have been combined. If these are removed once they have been combined, then the solution-construction-graph becomes a binary tree: each PGP-partial-solution points to only one supported partial solution formed at a single node. This is the simplest representation of constructing a solution, and in a reliable system is a reasonable way of forming overall solutions. However, in less reliable networks, more redundancy may be desirable so that if a node that is to combine partial solutions fails, there are other node's that are also responsible for forming the desired partial solutions. To allow various degrees of reliability, the PGPlanner removes PGP-partial-solutions from the PGP-partial-solution set only after they have been used in some specified number of combinations. A PGP-partial-solution is removed once the number of PGP-partial-solutions it supports equals the solution-construction-redundancy. Because the algorithm ends when a single PGP-partial-solution remains in the set, a PGP-partial-solution is also removed if it is combined in a PGP-partial-solution that represents the same track: once the solution-construction-graph contains several complete solutions, these may be "combined" at nodes so that the solution-construction-graph has a single root (to simplify its representation and use).

To illustrate the formation of a solution-construction-graph, consider the examples graphically depicted in Figure 78. These are based on the three partial solutions that must be combined in the environment shown in Figure 53. In Figure 78a, the solution-construction-redundancy equals 1. Tracks d_1–d_6 and d_7–d_{11} can be combined slightly earlier at node 1 than they can be combined at other nodes, and node 1 forms d_1–d_{11}. Because node 2 lags behind, it should be sent this result which it combines with its own result into d_1–d_{15}. Alternatively, if the time-cushion is larger, the PGPlanner identifies that node 4 can combine results at an acceptably later time (since communication delays affect when it can start combining partial results). As shown in Figure 78b, nodes 1, 2, and 3 send their results to node 4 for integration in this case since node 4 does not otherwise participate in highly-rated PGPs.[5] Finally, if the solution-construction-redundancy equals 2, and the time-cushion is small, then the solution-construction-graph is

[5]The same behavior would occur if node 4 has better combination expertise (KSs), so that even though it gets the partial solutions to combine later it forms the combination substantially earlier. In this case, the solution-construction-graph would assign these integration tasks to node 4.

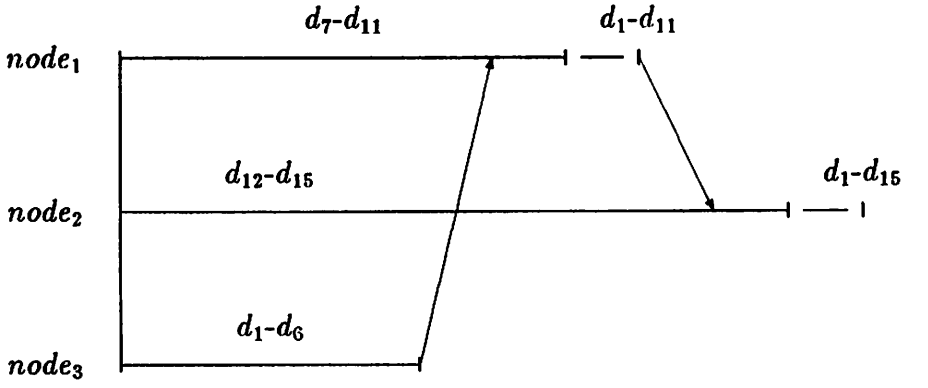
no longer a simple binary tree. Shown in Figure 78c, each of the initial partial solutions are combined at two different nodes. Tracks d_1–d_6 and d_7–d_{11} are combined at both nodes 1 and 3. Track d_1–d_{11} formed at node 1 is sent to node 2, which forms the complete track d_1–d_{15}, but node 2 also sends its partial track d_{12}–d_{15} to node 3 which also forms the complete track.

7.3.5 Determining Domain-Level Communication Actions

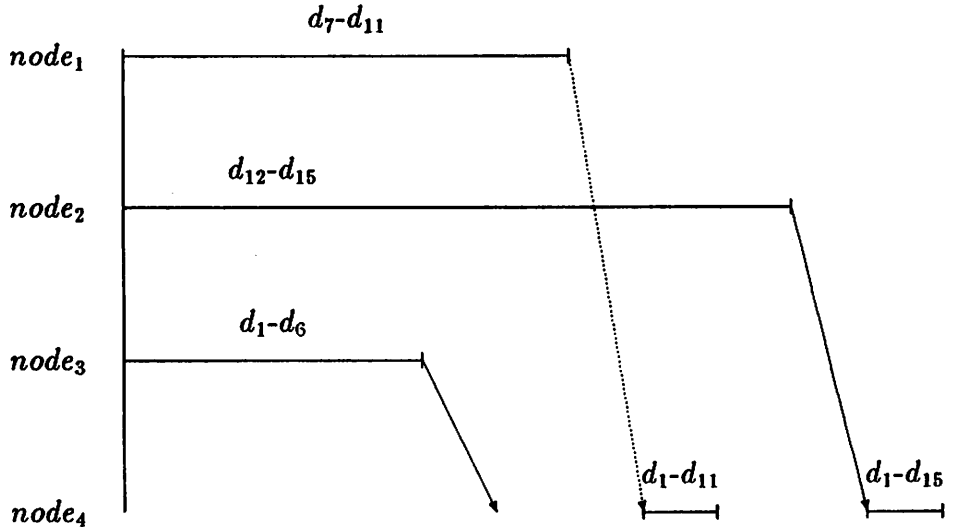
To decide what domain-level information (hypotheses) should be communicated among nodes for a given PGP, the PGPlanner considers the goals of cooperation and several attributes of the PGP. First of all, since the purpose is to integrate partial solutions into complete solutions, the PGPlanner uses the solution-construction-graph to determine what hypotheses should be transmitted, where they should be transmitted, and approximately when they should be transmitted. To form the overall solution, nodes must at least transmit sufficient hypotheses so that the separate pieces can be integrated, and the solution-construction-graph provides clear information about this exchange of hypotheses.

Although the solution-construction-graph indicates the necessary communication to generate complete results, nodes may communicate about hypotheses for other reasons. In particular, they may exchange information to help each other resolve uncertainty. If one node believes that another node could more effectively form its partial solution if it received a predictive hypothesis, then it should generate and transmit this predictive hypothesis. Moreover, if two nodes are attempting to verify each other's results by individually forming hypotheses covering the same data, then they should exchange these hypotheses to corroborate their results. The PGPlanner therefore must examine the plan-activity-map for the PGP and find opportunities for communication that promote the goals of cooperation, depending on the importance of avoiding redundancy versus verification, or of providing predictive results versus allowing nodes to work independently.

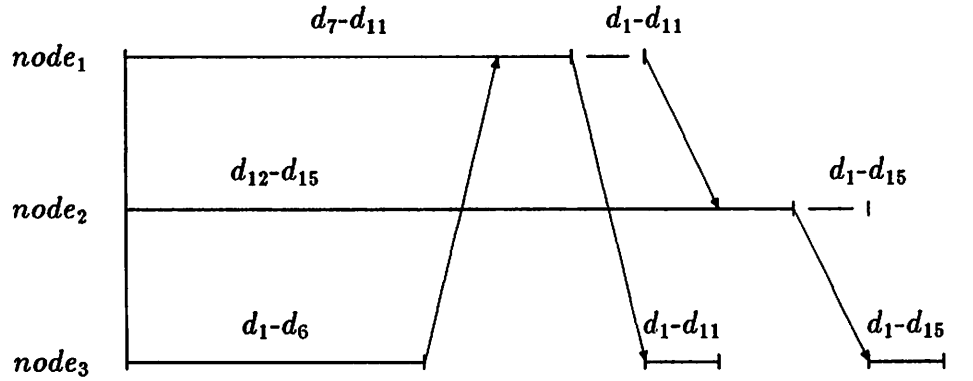
If the cooperation-parameter for reliability is greater than that for redundancy, then the PGPlanner scans the PGP's plan-activity-map to find verifying activities (plan-activities at different nodes forming results with common time-region characteristics). When it finds two matching plan-activities, the PGPlanner builds a communication expectation to send the results of the earlier plan-activity to the



(a)



(b)



(c)

Figure 78: Example Solution-Construction-Graphs.

other node (if possible), or to find a common integrating node where both results are sent and compared.[6]

If the cooperation-parameter for predictiveness is greater than the parameter for independence, then the PGPlanner should build communication expectations to send predictive results as well. To find predictive results, the PGPlanner scans the PGP's plan-activity-map to find plan-activities performed by one node that border on those being performed by another node. If these plan-activities are less costly (have shorter duration) than the other node's, then the other node may be facing ambiguity in its data. By sending the bordering result to the other node, this node may provide useful predictive information that helps the other node form results sooner.[7] For example, in the situation in Figure 53, the PGPlanner builds a communication expectation for node 1 to send the hypothesis it forms for d_8–d_9 to node 2, because this predictive information can help node 2 focus its attention on a smaller subset of its data.

The PGPlanner therefore builds a set of communication expectations for the PGP that it uses for making communication decisions. When it forms a hypothesis that it could potentially send (a KSI to send the hypothesis is instantiated), the node compares this hypothesis to the communication expectations for its PGPs, and only executes that KSI if the transmission corresponds to some communication expectation. Similarly, given a potential received hypothesis (a KSI to receive the hypothesis is instantiated), the node compares this hypothesis message to the communication expectations for its PGPs, and only executes that KSI if the reception corresponds to an expectation. By being more selective about what hypotheses are sent and received, the nodes reduce the amount domain-level communication in the network while still exchanging important hypotheses.

[6]Comparison for verification means that the two hypotheses are merged. Since they cover the same time-regions, the resulting hypothesis is not larger than the initial hypotheses, but it may have higher belief. Unfortunately, the belief system in the DVMT does not currently provide capabilities for increasing belief based on corroborating, independent results from several nodes.

[7]However, there is no guarantee that this information will actually help the other node, since it may be that the other node does not face more uncertainty but instead simply has inferior expertise (KSs).

7.3.6 Coordinating Current Activities

The PGPlanner must balance two opposing needs: it should avoid planning too far into the future since unpredictable events (failed actions, new data) may cause plans to change, but it should plan far enough so that it can make intelligent decisions about current actions. Planning future actions and interactions (in the form of the solution-construction-graph) is important for making good control decisions. For example, without making estimates about when nodes will form their partial solutions and when they are needed for integration, the PGPlanner cannot recognize situations where it can safely postpone actions for a more highly-rated PGP so that it can take useful actions for a more lowly-rated PGP. The results of these actions may provide predictive information to other nodes, and the earlier this predictive information is supplied, the less time those nodes will waste when forming their partial solutions.

To balance these needs, the PGPlanner may generate several PGPs, but discontinues its activities as soon as it has found current actions for each of the nodes that must be coordinated. It begins with a list of nodes to coordinate, as specified in the meta-level organization. It then steps through the PGPs, starting with the most highly-rated and ending when it has found current actions for each of these nodes. For each PGP, it checks to see whether any information about that PGP has changed since it was last reasoned about (whether any node-plans have more recent creation-times than the PGP's creation-time). If so, the PGPlanner updates the PGP's plan-activity-map, solution-construction-graph, and communication expectations, and resets its creation-time to the current time. It then examines the solution-construction-graph to identify which nodes should be pursuing plans associated with the PGP: if the PGP-partial-solutions associated with a node indicate that the node has activities that it must pursue, then the node is removed from the list of nodes to coordinate. A node has activities that it must pursue if there is a PGP-partial-solution associated with that node whose duration equals the difference between its begin-time and end-time (so the node has no time to spare), or if there is a PGP-partial-solution associated with that node which indicates there is time to spare but none of the less highly PGPs has possible actions that the node could take that would provide predictive information (the expected durations of the node's actions in that plan, indicated in the long-term time-predictions, are not significantly shorter than the expected

durations of any other node's actions, based on the long-term time-predictions from its node-plans).

After each PGP is successively processed, the PGPlanner may remove some nodes from the list of nodes to coordinate. The PGPlanner stops processing PGPs when the list of nodes becomes empty or the set of PGPs is exhausted, whichever comes first. The PGPlanner therefore generates views of actions and interactions for groups of nodes so that it roughly predicts the current activities of the nodes that it should coordinate. If the PGPlanner is coordinating nodes besides the current node, then it should inform these nodes about relevant PGPs. By sending them PGP information, the coordinating node can give other's a more global view of network activity. Moreover, a node may send PGPs to another node even if it is not coordinating that node: by exchanging PGPs, nodes communicate about how they view network cooperation, and thus can resolve inconsistent views.

7.3.7 Meta-Level Communication of Partial Global Plans

The meta-level organization specifies which nodes exchange PGPs. In fact, the communication of PGPs is very similar to the communication of node-plans as described in the previous chapter. First, the PGPlanner decides which PGPs could potentially be transmitted: it scans the queue of PGPs and any that involve other nodes are potential candidates. It then compares the creation-time of each of these PGPs with the record of what versions of the PGP other nodes already have (from the PGP's record:sent-to-nodes information). If any have obsolete versions, and the node is permitted to send PGPs to them (depending on the meta-level organization), then PGPs are transmitted.

As is outlined in the next section, a PGP might also be transmitted to elicit information about possible cooperation through the exchange of tasks. Such a PGP represents potential future interactions, and consequently has its future attributes set. How future PGPs are formed is described in the next section.

The PGPlanner constructs a PGP message out of a PGP's attributes, and this information is passed to recipient nodes. Depending on where the PGP is to be sent and how that node is related to the sending node, some information about the PGP may be eliminated. For example, if the sending node is supposed to coordinate the receiving node, then it might send a plan-activity-map whose plan-activities for other nodes have been removed. By doing so, the coordinating

node does not give the recipient the ability to form alternative plan-activity-maps (since it does not have all the relevant plan-activities). This decreases the recipient's computation costs (since it is not attempting to reorder network activities) and increases network consistency (since the recipient blindly follows the PGP). On the other hand, if two nodes are peers (with equal coordination authority), then they would exchange complete plan-activity-maps so that they could reason about each other's PGPs in an attempt to converge on consistent PGPs. They exchange PGPs to negotiate about network coordination, since they could adopt the most highly-rated PGP or might form a new compromised PGP.

When a node receives a PGP, it first determines whether the PGP was sent to elicit information or to supply coordination information or both. If the future information is set, then the node incorporates the PGP into its network-model and forms a future node-plan indicating its expected ability to perform the possible future tasks. If the future information is not set, then the node simply incorporates the PGP into its network-model.

When incorporating a received PGP into its network-model, the PGPlanner first checks to see if the PGP already exists: it compares PGPs' record information about the names of participating plans to see if any match the received PGP. If not, then a new PGP is formed. If a matching PGP already exists, then the PGPlanner must combine the received information with the existing information. First it determines which PGP is more highly rated (where the received PGP's rating is modified based on the recipient node's credibility in the sending node). It then steps through the PGPs' attributes and modifies the existing PGP based on their contents and the more highly-rated PGP. Many of the attributes must match if the PGPs match: the PGP's nodes and its record:node-and-plans, for example. However, for attributes that do not match, the values of the more highly-rated PGP are adopted, including its objectives and rating. If the PGPs have different plan-activity-maps, then the PGPlanner examines their record:plans-and-times attribute to determine, for each node, which PGP has the more current view of its activities. The PGPlanner combines the most current plan-activities from the two PGPs to potentially form a new plan-activity-map (of course, the received PGP might be from the same node (a coordinator node) as the existing PGP, and therefore its entire plan-activity-map is more recent and is adopted in its entirety). Thus, nodes might exchange PGPs to form compromise views on coordination.

Once the PGPlanner forms a new PGP or modifies an existing PGP from the message, it links this PGP to the local environment. This linking is simply done by using the new PGP's record information about the names of participating plans and, if any of these plans are modeled by node-plans at the recipient node, then these node-plans are updated to point to the PGP and the PGP to point to these node-plans as participants. In addition, the PGPlanner examines the PGP to determine whether any local plans for the current node must be modified. It extracts plan-activities for the current node from the PGP's plan-activity-map, and compares them with the participating local plan's long-term information (time-order and time-predictions). If the PGP represents a reordering of the local plans long-term:time-order, then the local plan is updated: its time-order is changed to reflect the reordering and new short-term actions may be found if the next i-goal to pursue has changed.

7.3.8 Coordinating Future Activities: Prediction and Task-Passing

The PGPlanner, as described so far, decides how nodes should process their data in an effective manner so that the network can most efficiently form complete solutions from that data. To plan actions and interactions better, the PGPlanner should anticipate possible future tasks and consider how these tasks could affect coordination. In addition, it should recognize situations where tasks currently at one node should be transferred to another node to make better use of network resources and expertise. The PGPlanner's ability to represent activities and goals of groups of nodes as PGPs allows extensions that permit it to anticipate future tasks and transfer tasks when appropriate.

Anticipating Future Activities

To anticipate future tasks, the PGPlanner extrapolates the objective tracks of PGPs to determine whether a vehicle that is being tracked by some nodes will likely pass through other nodes' sensed areas. This extrapolation is based on the assumption that the vehicle will approximately maintain its current course.[8] By

[8] Without this assumption, the vehicle could potentially move anywhere among the nodes, and the uncertainty about where it could be would make anticipating future tasks impractical. Given information about environmental characteristics that might trigger course changes (for example,

building a view of a vehicle's future course and identifying when that course intersects the sensed areas of other nodes, the PGPlanner can build future node-plans indicating that these nodes will likely be performing activities to track the vehicle in the future. Because these are future node-plans, the PGPlanner sets certain future attributes such as the future:start-time. This time can come from two sources. If the simulated environment has signals detected at known intervals, then the PGPlanner can simply compute when a node will begin receiving data. To enlarge the range of problems that the system can study, however, the start time can instead be retrieved directly from the environment file which specifies when data will be simulated to arrive. This flexibility allows the user to simulate data arriving in bursts or out of order, and the PGPlanner as currently implemented uses this information to determine the start time of a predicted future node-plan.

Future node-plans point to the PGPs that triggered their creation. When computing the PGP's plan-activity-map, the PGPlanner can form plan-activities for predicted activities, so that when reordering activities it does not give responsibility for some tasks to a node that will soon receive other important data to process. However, the cost of extrapolating tracks and forming future node-plans can be high and possibly unnecessary, since other unexpected changes to the nodes' problem solving may cause PGPs to be changed over time: rather than anticipating future node-plans, the PGPlanner may be better off simply reacting to these node-plans when the data actually arrives. In the current implementation, therefore, the PGPlanner does not typically generate future node-plans in the process of coordinating nodes' current activities.

When nodes exchange tasks, however, this investment of time is more worthwhile. Because exchanging tasks can be a costly process (both in computation and communication), the PGPlanner should determine whether a node that is currently underutilized (not participating in important PGPs) is likely to remain underutilized long enough to perform tasks for other nodes. Tasks should not be passed to a node that will soon receive data from its sensors that it will have to process, because either it will not process this data in a timely manner or else it will postpone the received task (possibly making the exchange of information

---

the terrain it is heading for), the PGPlanner's uncertainty could be reduced so that anticipating the vehicle's future path is feasible.

unnecessary).

Task-Passing

To make better use of network resources and expertise, the PGPlanner is capable of moving tasks (data to process) from one node to another. The task-passing process can basically be broken down into these steps:

- Decide whether any nodes are overburdened;
- Find any tasks that could be passed from these nodes;
- Find any nodes that are underutilized and could perform these tasks;
- Build a PGP to represent the possible cooperation between nodes;
- Send the PGP to possible nodes that could perform tasks;
- Wait for node-plan messages in response to the PGP;
- Choose among the responses for the best (if any);
- Transfer the tasks (data) to the chosen node.

The PGPlanner steps through the highly-rated PGPs, and for each first checks to see if there is a bottleneck node. By examining the PGP's solution-construction-graph, the PGPlanner determines whether there is a node whose partial solution is formed significantly later than the partial solutions with which it is combined (if any). The decision about when a result is significantly later is based on the min-task-duration information contained in the cooperation-parameters: if the time the network waits for the partial solution to be formed exceeds the min-task-duration, then the delay is substantial enough for the nodes to attempt to assign some of the tasks elsewhere.

Given a bottleneck node, the PGPlanner next finds a task to pass, where a task is represented as sequence of plan-activities. By having another node perform these plan-activities, the PGPlanner hopes to better balance load. It finds all of the plan-activities that go into the partial solution that the bottleneck node is generating, and roughly divides the pending plan-activities (some might already be complete) in half. The plan-activities that were to be performed by the node

at later times are grouped together into a task to pass, while the node keeps the activities that it expects to perform sooner.[9]

Next, the PGPlanner scans its network-model to identify underutilized nodes that could potentially perform this task. A node is underutilized if it participates only in unimportant PGPs or is idle. The PGPlanner does *not* assume that a node is underutilized if it has no information about the node. Because of communication delays, such an assumption would lead to a flurry of attempted task-passing activity early during problem solving since nodes have not had time to exchange enough information to build adequate network-models. Instead, the PGPlanner works from knowledge instead of ignorance, and determines that a node is underutilized based on explicit information about that node: that it currently participates in only lowly-rated PGPs (it has completed its activities for any highly-rated PGPs), or that it has transmitted an idle node-plan indicating that it does not participate in any PGPs at all.

The PGPlanner further scrutinizes these possible nodes by using the mechanisms for future predictions previously described to determine whether any of these nodes are likely to be receiving sensor data in the near future. The PGPlanner computes the time between the earliest time the node could get the task (based on communication delays) and the start time of any future node-plans. It also computes an expected duration for the task at that node: the expected duration of the task at the source node is multiplied by the ratio of the potential recipient node's synthesis capabilities and the source node's capabilities (from the node-models' expected-processing-costs). If the expected duration exceeds the time available before any future node-plans start, the PGPlanner removes the possible recipient node from consideration.

If it has found a task to send and one or more potential recipients, then the PGPlanner modifies the PGP that it used to find the task. Into the plan-activity-map of this PGP, the PGPlanner inserts new plan-activities. It first copies the plan-activities that comprise the task, and then modifies the copies. It alters their begin and end times based on expectations about when a recipient node could

[9]To simplify the current implementation, the task-passing mechanisms attempt to roughly divide the bottleneck activities in half to split the load between the bottleneck node and another node. To make use of any other available nodes, the passed tasks must be subdivided again if the recipient node becomes a bottleneck node. In the future, we hope to generalize the mechanisms so that the bottleneck activities can be divided into an arbitrary number of roughly equal portions to increase parallelism.

pursue them. It builds these expectations by considering communication delays between nodes: since the PGP must be sent to the nodes, node-plans received, and then tasks sent out, the PGPlanner uses communication delays to estimate the earliest time that a recipient node could receive the tasks. The PGPlanner also changes the name of the node performing these plan-activities from the name of the bottleneck node to a special marker indicating that the activities are currently unassigned. These plan-activities are interleaved with other plan-activities in the PGP's plan-activity-map. The future attributes of the PGP are also modified: the to-nodes are set to the possible recipients, the respond-to attribute is set to the current node (that is forming the PGP), and the response-label is set to the local name for the PGP. This PGP is then sent to the to-nodes.

When it receives the PGP, a recipient node first checks to see if it is in fact a request for information. If the node is listed in the PGP's future:to-nodes attribute, then the node develops a future node-plan in response to the PGP. It begins by extracting from the plan-activity-map the unassigned plan-activities. It builds a node-plan whose plan-activity-map contains these plan-activities, with its own name replacing the special marker. The node then examines its PGPs and node-plans for its own activities to determine the earliest that it could begin working on the task if it were to receive it. As part of this computation, the PGPlanner determines whether any future node-plans can be developed from its PGPs, and if so when data from its sensor is likely to start arriving. If future node-plans are possible and the received task may not be completed before the start-time of these future node-plans, then the task's plan-activities would have their begin and end times pushed into the future to avoid interference. Even if no future node-plans are found, the PGPlanner may still push the task's plan-activities into the future if they may interfere with expected actions based on currently available data. Moreover, it can use its model of its own expertise (cost of KS actions) to increase or decrease the duration of the plan-activities (difference between begin-time and end-time) indicating an intention to complete the task faster or slower.

The recipient node sets other fields in the future node-plan as well. It uses its confidence in information from the other node (from the meta-level organization) to modify the rating of the future node-plan. It also sets the node-plan's future attributes: the start-time as the begin time of the first plan-activity, planned-by

as the received PGP's future:respond-to node, and the triggering-PGP is set to the received PGP's response-label. The resultant future node-plan that conveys when the node could do the task and how highly it rates it is then transmitted back to the node that sent out the PGP.

As it receives node-plan messages, the node that formed and sent the PGP examines them to decide whether they are in response to the PGP. If the received node-plan has, as its future:triggering-PGP attribute, the name of a local PGP, then this node stores the message in that PGP's future:responses attribute. After sufficient time has elapsed between when it sent the PGP (stored in the future:sent-time attribute) and the current time, the PGPlanner scans the responses and attempts to find the best one. It compares the responses' plan-activity-maps to find the nodes that could complete the task earliest, and looks at the PGP to find out whether this is substantially earlier than when the node that already has the task could complete it. If the responses indicate that some node could complete the task faster than the node that currently has it, then the PGPlanner decides to transfer the task.

Transferring a task means sending the data needed to form the desired results of the plan-activities. The PGPlanner modifies the PGP's future attributes by first of all setting the source-node information: the source node has the data for the task being transferred. Since some nodes may coordinate others, the node with tasks to pass and the node that coordinates the task-passing need not be the same—a coordinator node may decide which of the nodes it controls will send out tasks and which will receive them. When it has decided what task (activities) should be passed, which node should send them, and which node should receive them, the PGPlanner sets the communication attribute to indicate this transfer of data. The PGPlanner also removes the unassigned plan-activities from the PGP's plan-activity-map and interleaves the plan-activities from the node that will receive the task. It then sends this PGP to all of the previous recipients and also to the source-node.

When a node that responded to the PGP receives the modified PGP, it checks the PGP's future:communication attribute to see if it should be receiving the task. If not, then it removes the future node-plan that it had formed for the PGP from its network-model and might adjust other future node-plans as well. For example, if it had responded to a number of PGPs and had generated plan-activity-maps

based on possibly receiving tasks that it had already responded to, then once the task is awarded elsewhere the PGPlanner changes the plan-activity-maps of other future node-plans to indicate that it could pursue the tasks earlier, and it sends these modified future node-plans to the node that requested them. In this way, nodes can respond to multiple requests and can update their responses when tasks are assigned to particular nodes, although because of communication delays it is possible that the updated information does not reach the other node before the task is assigned and a less than optimal assignment might be made. In a network with communication delays, potentially errorful channels, and asynchronous activities at the different nodes, such incoherence is unavoidable.

When the recipient node examines the PGP and discovers that it has been awarded the task, it saves the PGP information and allows itself to receive the data. Similarly, if the recipient node sees that it is the source-node for the task, then it invokes the communication mechanisms to send the data out. The node that receives this data will form a local plan to process it, and develop a node-plan for this plan. By communicating about this node-plan, the node updates its own network-model and the network-models of other nodes so that the appropriate PGPs are updated to reflect how the node is now cooperating with others.

To add flexibility in the task-passing capabilities of nodes, the PGPlanner has several options not mentioned above. First, if it believes that the network-model it uses for deciding likely nodes for a task might be incomplete or obsolete, the PGPlanner could more fully broadcast the PGP to elicit a broader range of responses. Thus, the PGPlanner can work in a focused addressing mode as described above, or can work in a more broadcast mode if desired. The type of mode it works in is specified in the cooperation-parameters. A second option it has is to improve reliability by assigning a task to multiple nodes. If nodes are likely to fail or encounter unexpected future activities that may interfere with received tasks, then important tasks should be multiply assigned to increase the likelihood that some node will complete them. The PGPlanner, when it scans the node-plan messages that it receives in response to a PGP, could choose the n best responses (instead of the single best) and assign the task to each of those nodes. It integrates plan-activities for each recipient node into the PGP's plan-activity-map and plans communication to pass the appropriate data to each of the recipients.

The task-passing mechanisms have several limitations, however. Chief among these is the inability to swap tasks: tasks can be passed from one node to another to better balance load, but two nodes that each have important tasks do not recognize, in the current implementation, when they should swap their tasks to make better use of network resources and expertise. This is a common difficulty in task-passing formalisms. To identify the best distribution of tasks among nodes requires a "world" view of tasks and nodes, which is difficult (and usually impossible) to achieve in a dynamically changing, distributed environment. When nodes only communicate about the subset of their tasks that they cannot complete locally and should pass to others, then they may miss possible swaps of tasks. In the future, our task-passing mechanisms might be extended to recognize some possible swaps (based on the PGPlanner's network-model), even though finding optimal assignments of tasks among nodes will be impractical in environments where tasks change dynamically.

A related issue that must also be addressed concerns interference between possible future tasks: when responding to received PGPs, a node builds the responding future node-plan based on both its current plans and on the future node-plans that it has already made. When tasks that it had built future node-plans for are awarded elsewhere, the PGPlanner may change its responses to other PGPs and send these out. Because at any time nodes may be updating their responses (bids) for tasks, the PGPlanner's decision about where to award a task may be based on inaccurate, out-of-date information. Tasks therefore may not be assigned to the best nodes. This lack of coherence is a common problem in loosely-coupled systems. Given delays and uncertain communication channels, it is impossible to guarantee that the PGPlanner can ever have common knowledge about what nodes are best for tasks. Our mechanisms only allow it to make the best decisions it can with its limited view, and the network pursues solutions despite non-optimal task assignments.

In summary, the task-passing mechanisms essentially go through basic contracting steps: a node builds a task that could be passed elsewhere,[10] it sends out bid-requests (in the form of PGPs) to those nodes that it believes could potentially do the tasks, it receives bids (future node-plans) from those nodes, it

[10]Note, moreover, that task decomposition depends on the particular network situation rather than being done in a prespecified manner.

chooses the best bid(s) (if any), and it stimulates the passing of tasks between nodes (by transmitting the PGP to the source and recipient nodes). The important thing to recognize is that contracting is really a type of partial global planning because it is a PGP shared by two (or more) nodes indicating how they can work together by exchanging tasks. Since contracting is a form of partial global planning, the PGPlanner and the PGP representations can be used for it as well: the same representation that allows nodes to coordinate their current activities serves to allow them to identify and coordinate future activities caused by passing tasks.

7.3.9 Putting It All Together: Partial Global Planning, Local Planning, and Problem Solving

Now that we have examined the important computations made by the PGPlanner—how it finds PGGs, forms PGPs, builds and reorders PGPs' plan-activity-maps, generates solution-construction-graphs, develops communication expectations, finds current activities for nodes, hypothesizes future activities for nodes, identifies possible transfers of tasks, and communicates about PGPs—this section describes in more detail how all of these activities fit together and how the PGPlanner interacts with the local planner and with problem solving. To understand how these mechanisms work together, we examine the activities of a node through a node execution. The basic activities of the node can be broken down into four major phases: incorporating new information, reasoning about local and network activity to find a local action, transmitting information that affects coordination, and pursuing the local action.

Incorporating Information

The first phase of node activity is incorporating any new received information. This information might be data from other nodes or from sensors, or it might be meta-level (coordination) information from other nodes. The node starts off by processing new domain-level information: the local planner updates the clustering hierarchy and changes the local plans based on any new or modified alternative-goals. The process of updating the local plans was described in detail in Chapter 4.

The PGPlanner is then invoked to update the node's model of network activities. Changes to the network-model can stem from received coordination information and from changes to local plans. Let us address how changed local plans affect the network-model first. If a new local plan is formed, a node-plan summarizing that plan is generated and added to the node's model of itself. If highly-rated enough, this node-plan may later be transmitted to other nodes. An existing local plan may be updated either because new data arrives or because the short-term actions of the plan are modified (actions to repair the plan or to work on the next i-goal). In either case, the plan's corresponding node-plan may or may not be altered, depending on the significance of the change to the local plan. Significance of change is determined from a more global perspective: the PGPlanner examines the PGPs that the node-plan participates in and determines whether the change to the local plan will substantially affect the activities of nodes and their expected interactions. A change to a local plan should be propagated to that plan's node-plan if:

- the objectives of the plan have changed in *unexpected* ways (ways that were not predicted by a PGP);
- the order of activities being pursued by the plan have changed (so that expected interactions may be affected);
- the predicted time when partial solutions will be formed deviates substantially from what is expected in the PGP's solution-construction-graph.

The first reason for propagating changes handles the situation where new data, especially new sensor data, arrives at the node. If the possible solution that the plan is working toward changes unexpectedly, then it is important that network-models reflect this change. However, sometimes a plan's objectives change in expected ways. A node that is expected to integrate two partial solutions (based on a PGP's solution-construction-graph) should not update its node-plans and network-model when it receives the expected partial solutions and modifies its local plans to find actions to integrate them. Only when the changes to its local plans are not predicted by PGPs should the node update its node-plans and transmit these updated node-plans to others.

The second reason for propagating changes is when the local planner has changed the order in which activities will be pursued. Since PGPs are based

on expectations about the order in which nodes will generate partial results, it is important that network-models reflect any changes to such an ordering. However, if the order is simply extended with expected new information (such as a partial solution to integrate), then the change is expected and the network-model need not be updated.

The third and last reason for propagating changes in local plans to node-plans is generally the most common. Node-plans and their resulting PGPs involve expectations about when activities will be performed and when nodes will interact. However, these expectations are based on rough predictions of likely durations of activities (see Chapter 4). As a plan is pursued, the actual durations of activities often deviate from the expected durations, and the predictions for remaining activities might be updated. When a local plan is updated, therefore, the predictions might change and the PGPlanner must decide whether the changes could affect network activity. The PGPlanner compares the local plan's new predictions against the predictions recorded in the node-plan. By using the time-cushion as the definition of "significant" time, the PGPlanner determines whether the old and new predictions differ by more than this measurement. If so, then the node-plan is updated; otherwise, the minor differences are not considered important from the more global perspective.

The time-cushion thus determines how and whether nodes will respond to the feedback [Arbib, 1972] provided by the node's actual performance. Feedback is useful so that nodes can react to changes and modify how they coordinate appropriately. However, overcompensating for minor variations can lead to oscillating behavior where nodes bounce from one way of coordinating to another instead of converging on a stable view. Since planning and execution are interleaved in our system, such oscillations degrade performance but eventually stop, because with each oscillation some problem solving occurs so that eventually nodes complete their activities. A smaller time-cushion leads to more changes to node-plans and network-models, possibly causing oscillations and increasing communication and computation overhead as nodes recompute how they should coordinate. A larger time-cushion, on the other hand, reduces oscillations and costs but at the possible price of having the network generate solutions in a less effective manner because nodes were less responsive to changes. The time-cushion thus represents how nodes balance predictability and responsiveness.

The other source of changes to the network-model is received node-plans and PGPs. Received node-plans are used to update the node-models of appropriate nodes, as was described in the previous chapter. Received PGPs are introduced into the set of PGPs as described above. Furthermore, any received PGPs that represent requests for future node-plans (bids) are processed at this time: the future node-plans are generated and marked for transmission. Finally, any received PGPs that indicate that this node is a source node for transferred tasks is processed, where the data associated with the tasks are sent to the appropriate recipients.

Reasoning About Local Plans and PGPs

After received domain-level and meta-level information has been incorporated appropriately, then the next step is to update PGPs if necessary. When the network-model has not changed, the PGPlanner does not need to update any PGPs (since the information it would use is the same that it last used). If the network-model has changed, then the PGPlanner can step through its PGPs until it has found current activities for each of the nodes that it should plan for based on the meta-level organization. The way the PGPlanner does this has already been outlined.

As it develops PGPs that involve the appropriate nodes, the PGPlanner associates with each of these nodes the PGP that it currently is pursuing. When it has finished developing these PGPs, therefore, the PGPlanner can return the PGP that involves the local node. Through the pointers between PGPs, node-plans, and local plans, the PGPlanner finds the local plan associated with the PGP.[11] The local planner then works with this plan: it finds the next action to pursue, possibly updating, repairing, or aborting the plan in the process. If it successfully finds an action (with a KSI), the local planner triggers the KSI for execution. When local planning must further alter the plan, then once the local plan is changed the PGPlanner once again goes through the steps of updating network-models and PGPs if necessary. The control mechanisms thus cycle through local planning and PGPlanning (although depending on the changes caused by local planning, PGPlanning may do very little) until the next action can be found. As

[11]Recall that if initially several local plans participate in the same PGP, they are merged into a single local plan.

a failsoft mechanism, moreover, if after trying all the PGPs the node still cannot find an action, the PGPlanner drops out of this loop and the node simply pursues its best local plan. Similarly, if unable to find a local plan to pursue, the local planner drops out of the loop and the node simply pursues the most highly-rated KSI.

Meta-level Communication

Once the next action for the node has been found, then the PGPlanner must determine what information, if any, it should pass on to other nodes. It scans the network-model for new and freshly modified PGPs and node-plans, including node-plans that it formed in response to PGPs that request task-passing information.

The PGPlanner also invokes the task-passing mechanisms at this time. These mechanisms scan the set of highly-rated PGPs and determines whether any of them could potentially be improved by the transfer of some tasks, and if so, to which nodes. The mechanisms modify the PGP as described earlier and the PGP is transmitted to the potential task receiving nodes. In addition, these mechanisms scan the set of PGPs to find any that had been sent out previously to elicit responses. If any exist and if sufficient time has elapsed[12], then the mechanisms scan the set of responses stored in the PGP to find the best one(s). If the mechanisms find a suitable transfer of tasks, the PGP is updated (with communication information filled in) and sent to the appropriate nodes, including the source node for the tasks. When the current node is also the source node, it triggers the communication mechanisms to transmit the data associated with the tasks.

Problem Solving

Finally, the KS associated with the next action is invoked and the node takes the next step toward solving the problem.

[12]Sufficient time for the PGP to have arrived at the nodes and for them to respond is computed from the communication delays associated with the network.

Summary

The sequence of control activities that a node invokes to find another action is substantial and possibly costly. At times the overhead of local and partial global planning is simply not worth the cost when it steals too much computation time from actual problem solving. The mechanisms have been developed so that they can be scaled down: the node can bypass the partial global planning or both the local and partial global planning mechanisms, and fall back on the simpler mechanisms which could possibly be more cost effective (achieve acceptable control with much less overhead). In the next chapter, we evaluate the mechanisms and explore some cases where less sophistication is actually better.

Even though the description considers these control mechanisms to be applied sequentially, they are also amenable to parallel execution: the local problem solving, local planning, and PGPlanning can all go on in parallel. This is discussed as a future research direction in Chapter 9, but the basic idea is that the PGPlanner could be running constantly, using any information it has to (asynchronously) update local plans and possibly alter their position on the plan queue. At the same time, the local planner could asynchronously be updating, repairing, and modifying local plans, and changing the KSI queue to reflect its view. Problem solving occurs asynchronously, where the node simply invokes the next KSI on the queue. Thus, the components work asynchronously, but affect each other by manipulating the contents of each other's queues.

7.3.10 Examples of Coordination of Cooperation

In this section, we illustrate how the mechanisms work through some detailed examples. The purpose of these examples is to describe the internal reasoning performed by the local planner and PGPlanner in particular situations, and to do this we examine several sample experiments in detail. The first example illustrates how a single coordinating node identifies a PGP for an important overall solution, recognizes how nodes should pursue their local activities and exchange results to work as an effective team, and distributes these PGPs to coordinate nodes. The second example shows how each node can individually form PGPs to coordinate activities, how a node can safely postpone important activities so that it helps another node work on its plans, and how nodes can assign tasks to

nodes with suitable expertise. Finally, the third example examines task-passing and prediction in more detail: how nodes can pass tasks when appropriate and can recognize when task-passing is not advisable.

These examples are not intended to cover every issue that could possibly arise in a network of problem solvers but instead to describe in detail the behavior of cooperating nodes in certain situations. This is in preparation for the next chapter, where we discuss a wider variety of experimental results but where we do not go into as much detail.

Example 1: Centralized Organization and Predictiveness

The first example has the environment of Figure 53 with four cooperating nodes that track a vehicle passing among them. We assume in this environment that: each node can communicate with any other node with a delay of two time units; each KS execution takes one time unit; the data arrives over the first 15 time intervals so that problem solving begins at time 16; and that nodes have the same knowledge sources (equivalent expertise). The meta-level organization assigns the principal coordination responsibilities to node 4 (since it has no data of its own). Nodes 1, 2, and 3 send node-plans to node 4, which forms PGPs and sends them back to the other nodes. Moreover, when sending a PGP to a node, node 4 only supplies plan-activities for that node so that it cannot locally develop alternative plan-activity-maps. This enforces consistency between nodes and reduces overhead (since the receiving node must blindly follow the PGP). The time-cushion in this experiment is 0 time units (although this is changed in later variations), so that nodes update and transmit their node-plans any time their local plans deviate from the PGP expectations in any way. Finally, the min-task-duration is set very high (to 100) so that nodes do not pass tasks in this environment.

Over the first 15 time units, the nodes incorporate data for the 15 sensed times, so that problem solving begins at time 16. At this point, each node's local planner generates plans to process the data local to that node: node 1 has one plan to form the track d'_1–d'_5 and another plan to form the track d_4–d_{12}; node 2 has the plan to form the track d_{10}–d_{15}; node 3 has the plan to form track d_1–d_6; and node 4 has no local plans.[13] The PGPlanner at each node summarizes these

[13]This analysis ignores the lowly-rated plans for unlikely vehicle types. See Chapter 4 for more information.

local plans into node-plans and adds them to the network-model. It forms PGPs based solely on the local node-plans (so that initially each node's view of group activity is equivalent to its own activity), and pursues the most highly-rated PGP which is equal to the most highly-rated local plan. The PGPlanner at each node except node 4 transmits its node-plans to node 4.

Two time units later, at time 18, node 4 receives these node-plans and constructs larger PGPs. First, it identifies that there are two PGPs in the network: to form the overall track d_1–d_{15}, and for node 1 to form the track d'_1–d'_5. The first of these PGPs is more highly rated (since three nodes participate in it), and so the PGPlanner starts with that PGP. It constructs the plan-activity-map by interleaving the plan-activity-maps of each of the participating node-plans. For example, from node 1 it received a node-plan without a plan-activity-map (because locally node 1 is pursuing its other plan based on the more strongly sensed data d'_1–d'_5). However, the node-plan does indicate that node 1 intends to start at sensed time 4 and work across to sensed time 12. It also has rough predictions about how long node 1 expects to take for each of these activities. With this information, the PGPlanner builds a plan-activity-map for node 1, where it assumes node 1 will start the first activity when it becomes aware of the PGP (which will be two time-units from now since the PGP must be sent back to node 1). By assuming the next plan-activity will start after the previous one ends, the PGPlanner constructs the plan-activity-map for node 1. On the other hand, node 4 receives plan-activity-maps for the node-plans from nodes 2 and 3: since these plans are currently being pursued at those nodes, the PGPlanner at those nodes constructed and sent the plan-activity-maps with those node-plans.

The PGPlanner interleaves the plan-activities in order of their end-times, so that it can identify the order in which results will be completed. Since this is the most highly-rated PGP, it need not check against other PGPs to find conflicts. The PGPlanner therefore next applies the mechanisms for reordering plan-activities to find better coordination. The cooperation-parameters are such that nodes assign higher costs to actions that are redundant or that generate less predictive results. The reordering sequence is shown in Figure 79. The initial ordering of actions for each node and their costs is shown in Figure 79a. The PGPlanner swaps less costly plan-activities for more costly plan-activities for a node, as shown in Figure 79b, and modifies the begin and end times of the plan-

Node 1	actions	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	
	costs	286	281	276	171	166	161	276	261	266	total = 2144
Node 2	actions	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}				
	costs	493	538	533	408	413	418				total = 2803
Node 3	actions	d_1	d_2	d_3	d_4	d_5	d_6				
	costs	267	272	277	412	427	442				total = 2097
											plan total = 7044

(a)

Node 1	actions	d_9	d_8	d_7	d_{11}	d_{12}	d_{10}	d_6	d_5	d_4	
	costs	115	130	145	215	220	210	320	345	370	total = 2070
Node 2	actions	d_{13}	d_{14}	d_{15}	d_{10}	d_{12}	d_{11}				
	costs	372	377	382	537	507	523				total = 2698
Node 3	actions	d_1	d_2	d_3	d_4	d_5	d_6				
	costs	246	251	256	361	366	371				total = 1851
											plan total = 6619

(b)

Node 1	actions	d_9	d_8	d_7	d_{10}	d_{11}	d_{12}	d_6	d_5	d_4	
	costs	114	129	144	209	214	219	319	344	369	total = 2061
Node 2	actions	d_{13}	d_{14}	d_{15}	d_{12}	d_{11}	d_{10}				
	costs	371	376	381	506	522	536				total = 2692
Node 3	actions	d_1	d_2	d_3	d_4	d_5	d_6				
	costs	245	250	255	360	365	370				total = 1845
											plan total = 6598

(c)

Figure 79: Example of Plan-Activity-Map Reordering.

activities to reflect their change of position. The PGPlanner computes the costs of each plan-activity in this new ordering, and the total cost of the new sequence is less than the initial sequence. The process is then repeated, resulting in the ordering shown in Figure 79c, which is better (has lower cost) than the previous ordering. In this ordering, no more costly actions precede less costly actions at the same node, so the algorithm terminates and the ordering is returned as the chosen reordering of node activities.

With the reordered PGP plan-activity-map, the PGPlanner next forms the solution-construction-graph. Because the cooperation-parameters lead the PGPlanner to avoid redundancy, it steps through the plan-activity-map and groups together plan-activities for each node that have not yet been developed at other nodes.[14] In this case, three different groups are found (one for each node), and

[14]When several nodes expect to finish working on the same data at almost the same time

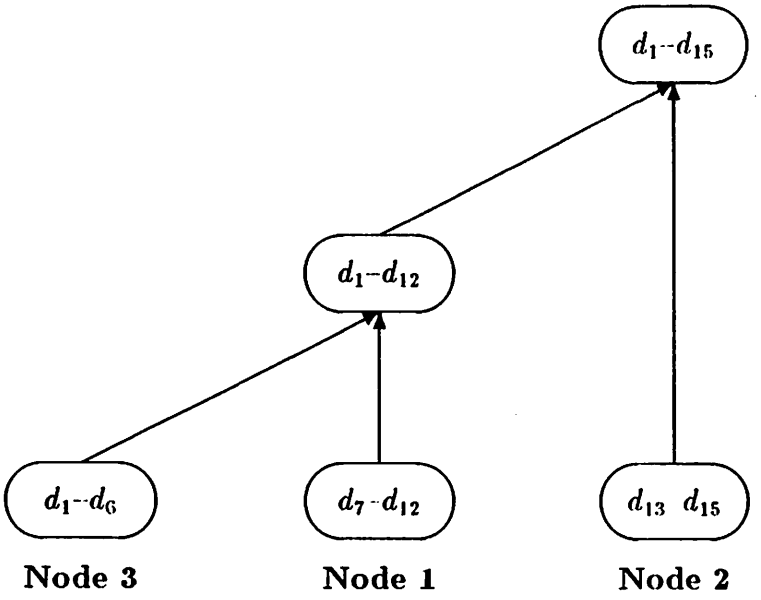


Figure 80: Example Solution-Construction-Graph.

a PGP-partial-solution is formed for each group. These are the leaves of the solution-construction-graph shown in Figure 80. The PGPlanner then considers each pair of these and finds that the PGP-partial-solutions from nodes 1 and 3 can be combined earliest, and they should be combined at node 1. This new PGP-partial-solution could then be combined at node 2 with node 2's PGP-partial-solution. The resulting solution-construction-graph is shown in Figure 80. The PGPlanner also determines that, in the only other PGP there is no possibility for predictive results to be formed earlier (since only one node participates in it). Thus, it decides that each of the participating nodes should pursue activities for the shared PGP first.

Next, the PGPlanner builds the set of communication expectations. First it uses the solution-construction-graph to recognize that the result d_1–d_6 should be passed from node 3 to node 1, and that the result d_1–d_{12} should be passed from node 1 to node 2. The PGPlanner also scans the plan-activity-map to recognize any additional communication that could help network problem solving. Because the cooperation-parameters indicate that providing predictive information is more important than refraining (to improve independence), the PGPlanner scans down

(within the time-cushion), then the relevant plan-activity for each is grouped with the others.

the plan-activity-map and recognizes that node 1 will develop results in an area bordering on node 2 (since their sensors overlap) and that the duration of its activities in these areas is substantially less than node 2's (indicating that perhaps node 2 could use predictive information). The PGPlanner finds the earliest partial track that node 1 will develop that borders on node 2's information: since node 1 will work on d_9 and d_8 and form d_8–d_9, the PGPlanner builds an expectation for node 1 to transmit this result to node 2.[15] If the cooperation-parameters had indicated that verification was desirable (instead of avoiding redundancy), then the PGPlanner would also scan the plan-activity-map to find the earliest cases where duplicate tracks are formed and would build communication expectations to transmit these (so that the recipient node(s) could verify results as soon as possible, and if the results are incompatible the plans can be abandoned as soon as possible).

At this point, the PGPlanner has completed the PGP, and must do two more things. First, it should modify any relevant local plans based on the reordered plan-activity-map. Since node 4 has no participating plans for the PGP, it does not need to change anything. Second, the PGPlanner must transmit the PGP to the relevant nodes, which in this meta-level organization are nodes 1, 2, and 3. Instead of transmitting the PGP as is, node 4's PGPlanner modifies the plan-activity-map differently for each message so that it only sends a node the plan-activities for that node. As a result, the recipient node cannot reason about alternative orderings. This helps enforce consistent views, and reduces communication overhead (a smaller plan-activity-map is sent) and computation overhead (since the recipient does not explore alternatives).

Finally, node 4's PGPlanner examines any other PGPs since it has not yet found current activities for itself. Since it has no local plans, it does not find such a PGP. Once it exhausts the set of PGPs, it completes its activities. No local plan (or KSI) is invoked at node 4.

The PGP messages incur communication delays and do not arrive until time 20. Before they arrive, some of the nodes have already changed their node-plans. Node 1 has finished developing d_1' and begins working on d_2', but finds that its predictions about how long each i-goal (other than d_1') will take were off by

[15]The PGPlanner uses knowledge about the communication KSs to recognize that the earliest result that can be transmitted must be a track with at least two data points.

one time unit because they did not include time for integrating new data with previously formed results (d'_1 was formed before any other results and so did not have to be integrated). Similarly, the predictions made by node 3 were off by one time unit.[16] Node 2 has not changed its plans yet since each of its i-goals takes so much longer (due to the ambiguity in its data). Because the time-cushion is 0, the changes to local plans made by nodes 1 and 3 cause them to propagate these changes to their node-plans, and these node-plans are sent at time 19 to node 4, where they will arrive at time 21.

At time 20, the PGP messages from node 4 arrive at nodes 1–3. Each node incorporates the received PGP into its queue of PGPs, and it links the PGP with the local environment. In particular, each finds out which local node-plans and local plans participate in the PGP, and makes sure to link them with the PGP. When the PGPlanner for each of these nodes begins its activities, it finds the new PGP information. Because it only receives plan-activities for itself and has no node-plans for other nodes, the PGPlanner does not attempt to change the received PGP's attributes, but instead follows the received plan-activity-map, solution-construction-graph, and communication expectations. However, because its local plans participate in the PGP, each node does update its local plans if necessary. The PGPlanner at node 1 alters the plan to form d_4–d_{12} so that its long-term time-order is now (9 8 7 10 11 12 6 5 4), as indicated by the PGP. Similarly, the PGPlanner at node 2 alters its plan so that it works on data at times 13, 14, and 15 first. Node 3 does not change its local plan since its activities are not reordered by the PGP. Both node 1 and node 2 propagate the changes to their node-plans and transmit these updated node-plans to node 4.

At time 21, node 4 receives the node-plans that nodes 1 and 3 sent at time 19. The node-plan from node 1 is not part of the highly-rated PGP and so does not affect it. However, the node-plan from node 3 indicates that node 3 will take longer than previously expected to form its results. The PGPlanner checks to see if the plan-activity-map should be reordered and finds that the nodes should not change the order of their activities but that the relative times they will complete activities has changed. These changes affect the solution-construction-graph be-

[16] Although the planning mechanisms could have been changed so that this difference between the first piece of a track and later pieces could have been anticipated, we chose to let the planning mechanisms make this minor mistake in predictions so that we could better see the effects of incorrect predictions.

cause now node 1 is expected to form its PGP-partial-solution earlier than node 3. The solution-construction-graph and the communication expectations are altered so that now node 1 sends its result d_7–d_{12} to node 3 which then sends d_1–d_{12} to node 2. This modified PGP is then sent by node 4 to nodes 1, 2, and 3, which receive it at time 23.

Node 4 receives node-plans from nodes 1 and 2 at time 22, indicating that they have changed the order of their i-goals. These updated node-plans correspond the node 4's expectations, and node 4's PGPlanner does not alter the highly-rated PGP based on them.

At time 23, several things happen. First, nodes 1–3 receive the modified PGP from node 4. They incorporate this PGP, which does not alter their local plans since their plan-activities are not reordered. The other thing that happens, however, is that node 1 has completed developing d_9 and, when it plans for d_8, it discovers that its predictions were off by one time unit for the same reasons as discussed above. It therefore updates the corresponding node-plan and sends this off to node 4.

Node 4 receives this node-plan at time 25, and once again changes the PGP's plan-activity-map because the relative times that results will be formed have changed. The solution-construction-graph and communication expectations are changed back again to their initial values: node 3 sends its partial result to node 1, which sends their combined results to node 2. This updated PGP is sent to nodes 1–3, which receive it at time 27 and incorporate it into their network-models.

At time 29, the predictive result d_8–d_9 is received by node 2. It had been formed by node 1 at time 27 and transmitted because it met the communication expectations. With this new information, node 2 makes major revisions to its local plans. It divides its original plan that involved a large amount of ambiguity into two plans: it modifies the existing plan to only develop results that are consistent with the received result (track the same type of vehicle) and forms a new plan to develop the other possible results. It updates its existing node-plan to reflect the change and forms a new node-plan for the new local plan. Both of these are sent to node 4.

At time 31, node 4 receives this information from node 2 and changes the PGP appropriately. Because node 2 has changed its plan to process a more selective subset of the data at each time frame, the predicted durations for its

plan-activities are substantially reduced. Its plan-activities are expected to complete earlier and are moved to earlier parts of the PGP's plan-activity-map. The solution-construction-graph is altered as a result: now node 2 is expected to process data in d_{12} before node 1 does, so node 1 should form track d_7–d_{11} while node 2 forms d_{12}–d_{15}. In fact, node 2 should form its PGP-partial-solution before node 1, so the solution-construction-graph and the communication expectations now have node 3 sending d_1–d_6 to node 1 and then node 2 sending d_{12}–d_{15} also to node 1, so that node 1 is now responsible for forming the overall result. The modified PGP is sent to the nodes which receive it at time 33.

The nodes follow this PGP until they form the solution at time 45. For most of this time, the local plans do not change. However, when first node 3 and then node 2 finish their partial solutions, they begin exploring their other local plans (to develop tracks for other vehicle types). Because the better plans have been completed, these local plans now become the most highly-rated local plans and these nodes begin sending node-plans to node 4. Node 4 cannot combine them into a larger PGP (since the node-plan that would join them would come from node 1 which is still working on the more highly-rated plans). As a result, the communication of node-plans and PGPs seems to peak in the early stages of problem solving (when nodes are developing an initial cooperative strategy) and at the late stages of problem solving (when some nodes are looking for alternative ways to cooperate since they have fulfilled their initial activities). Because plans do not change much during the middle stages of problem solving (in this environment), the amount of coordination activity drops off during this period.

As a variation on this environment, the time-cushion can be set to 1. Several things happen in this case. First of all, the amount of meta-level communication decreases: where before the nodes updated node-plans when the predicted duration of i-goals was off by 1 time unit, in this variation the nodes will consider that deviation acceptable and will not inform each other of it. The result is that fewer updates to node-plans are made and this reduces the coordination overhead, but that the nodes might not interact as crisply as before. That is, if some nodes take a little longer to form results and others take less time, then they might form overall solutions a little earlier if they had exchanged information and altered the PGPs. The time-cushion specifies how far off things can get before it is worth recomputing interactions.

Another thing that happens when the time-cushion is enlarged is that node 4 begins to come into play. When integrating results, it is generally faster to have one node pass one partial result to the node forming the other partial result: by having the result formed earlier sent to the node forming the later result, the time spent waiting for both results to be present at the same node is minimized. When the time-cushion is small, therefore, results will be integrated at nodes that form them. On the other hand, with a larger time-cushion, the difference between when these nodes can form the result and when unrelated nodes can form the result becomes less important. For example, in this environment node 4 can combine the results from nodes 1 and 3, but because it must wait for both it generally integrates the results 1 or 2 time units later. With the time-cushion equal to 0, node 4 is not called on to do the integration. However, when the time-cushion is set to 2, the PGPlanner recognizes that node 4 can form results soon enough and that it is underutilized (does not otherwise participate in important PGPs). In this case, node 4 is chosen to integrate results so that its computational resources can be used more effectively.

When the example environment with this variation is run, the same activities occur at times 16 and 18, except now the PGP calls for nodes 1–3 to all send their partial results to node 4 for integration. At time 19, the changes to local plan predictions are not propagated to the node-plans, so these are not sent to node 4. Nodes 1–3 receive the PGP at time 20, nodes 1 and 2 change their local plans accordingly, and the modified node-plans are sent to node 4. The PGP is *not* changed at time 21 since node 4 does not get updated node-plans from 1 or 3. Node 4 receives node-plans from 1 and 2 at time 22 but as before these do not alter the PGP. The nodes do not exchange PGPs and node-plans until node 2 alters its local plans based on the predictive information from node 1. The altered node-plan is sent to node 4, which updates the PGP and sends it back to the other nodes, where it still expects results to be integrated at node 4 but now node 2 provides results including d_{12} instead of node 1. The nodes follow this PGP, and the overall solution is formed at node 4 at time 46—slightly later than in the earlier version, but in this case the communication and computation overhead was reduced because less PGPlanning was needed. The costs and benefits of changing the time-cushion are more completely explored in the experiments of the next chapter.

Example 2: Broadcast Organization and Flexibility

This example explores how different groups of nodes can cooperate on more than one solution. The experimental environment is shown in Figure 81. Two vehicles pass among the nodes, where both pass through node 1's area. The strength of the sensed data varies, however, so that some portions of vehicle tracks may look more promising than others. Node 2 once again detects noisy data and so must resolve much ambiguity. We assume once again that: each node can communicate with any other node with a delay of two time units; each KS execution takes one time unit; and the data arrives over the first time intervals so that problem solving begins at time 12 when all data is present. However, in this case we assume that nodes do *not* have the same KSs and that this is reflected in the network-model: node 4 can integrate results better (faster) than the other nodes.

Many of the same issues as in the previous example arise, but some new issues are also explored. First of all, the meta-level organization allows nodes to broadcast node-plans and individually form PGPs. Because they individually have more autonomy, the nodes can be more responsive to unexpected events such as finding that their predictions about when they would form results are somewhat off. However, to prevent them from incurring large amounts of communication and computation overhead when updating their views of cooperation, the cooperation-parameters have a time-cushion of 1. With this time-cushion, nodes are less responsive to changes in their expectations, and they are more predictable since they are less prone to changing their views of how they will interact. We also once again set the min-task-duration to a large value (100) so that nodes do not pass tasks. Another important new issue is that now node 1 can cooperate with node 2 and with node 3, while nodes 2 and 3 do not cooperate. Thus, node 1 must plan its activities to be an effective partner with each of the other two nodes. Finally, the last difference is that, unlike the previous example where nodes were equally capable of combining partial solutions, in this case node 4 has particular expertise that make it the best node for integration tasks.

The data is incorporated over the first 11 time units, and problem solving begins at time 12. At this time, the each node forms its local plans: node 1 has two local plans, to form track d_9–d_{11} and track d'_6–d'_9; node 2 has the local plan to form track d'_1–d'_6; node 3 has the local plan to form track d_1–d_9; and node 4 has no local plans. Each node builds node-plans for its plans, generates PGPs

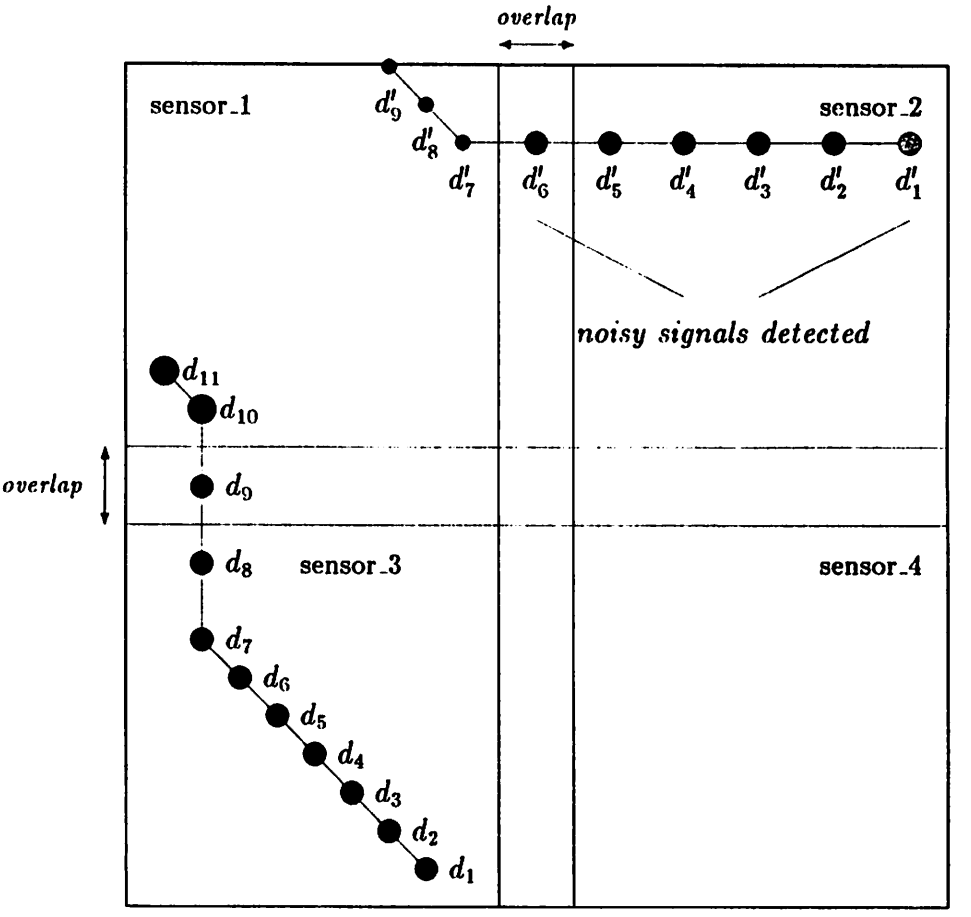


Figure 81: Four-Sensor Configuration with Multiple Vehicles Sensed.

based on its local node-plans, and broadcasts its node-plans to the other nodes. Each node then follows its most highly-rated PGP (currently corresponding to the most highly-rated local plan) for the next two time units. Node 1 works at d_{10} because its local plan to work on d_9–d_{11} is more highly rated and the data at d_{10} is more strongly sensed than the data at d_9. Node 2 works at d'_1, node 3 works at d_1, and node 4 is idle.

At time 14, nodes receive each other's node-plans and individually form PGPs. Each node first identifies two PGPs corresponding to the two vehicles moving through the area. Moreover, the rating of the PGPs is such that the PGP to form d_1–d_{11} is more highly rated since it consists of the more strongly sensed data. Consider the activities of the PGPlanner at node 1 as it develops this more highly-rated PGP. It forms the plan-activity-map by combining the plan-activity-maps for the participating node-plans of nodes 1 and 3. It applies the reordering mechanisms and discovers that the plan-activities do not need to be reordered: neither node is likely to provide predictive information to the other (they each have little ambiguity in their data) and each node was going to leave the possible redundant activities (in d_9) until later. Thus, node 1 has its i-goals ordered as (10 11 9) and node 3 has its ordered as (1 2 3 4 5 6 7 8 9).

When the PGPlanner forms the solution-construction-graph, however, it recognizes that the partial result formed by node 1 (d_9–d_{11}) will be sent to node 4 for integration well before node 3 has formed and sent its partial result (d_1–d_8). Node 1 therefore has some flexibility in when it pursues the plan. The PGPlanner recognizes that in the other PGP node 1 could potentially supply predictive information to node 2 (since node 2 has noisy data). As a result, the PGPlanner moves node 1's plan-activities in the more highly-rated PGP to a later time. It determines how much time node 1 should reserve for these activities (using the time-cushion to add some extra time to these predictions to insure that they will be performed in time even if the predictions are somewhat off). The PGPlanner then modifies the PGP's plan-activity-map so that node 1's plan-activities begin and end at later times but so that the last plan-activity still ends soon enough so that the partial solution can be passed to node 4 for integration. The communication expectations are formed for the PGP (to send the partial solutions to node 4), and, since the PGP is consistent with the local plans, the PGPlanner has finished processing this PGP.

Examining the more highly-rated PGP, the PGPlanner recognizes that it has not yet found the current activities for node 1 because it had postponed node 1's activities in that PGP until a later time. It therefore also processes the less highly-rated PGP. It constructs the plan-activity-map for this PGP out of the participating node-plans' information, and then it applies the reordering mechanisms. In this case, the plan-activities are reordered. Initially node 1 had planned on working on its i-goals in the order (6 7 8 9) because the data in d'_6 was more highly rated. When part of the larger PGP, the PGPlanner recognizes that d'_6 is possibly redundant, but that node 1 should supply predictive data adjacent to node 2's data. It therefore orders the activities so that node 1 can supply the predictive result d'_7–d'_8 as soon as possible by ordering the i-goals as (7 8 9 6). The PGPlanner builds a solution-construction-graph for this PGP as well and recognizes that once again node 1 will likely form and send its partial solution to node 4 before the cooperating node (this time node 2) will. However, since in this case node 1 is generating predictive information the PGPlanner does not consider moving its plan-activities to later times. The PGPlanner forms the communication expectations (to send the partial solutions to node 4 and for node 1 to send d'_7–d'_8 to node 2). It also updates node 1's participating local plan, changing the long-term time-order so that it will form the predictive result as soon as possible.

At this point, node 1's PGPlanner has found the current activities for node 1: to work in d'_7 to hopefully form predictive results for node 2. Since it has modified one of its local plans, it updates this plan's node-plan and broadcasts this information. Nodes 2, 3, and 4 each go through similar but not identical PGPlanning because they are trying to find their own current activities. Node 2 processes the more highly-rated PGP because it expects to cooperate with node 1 (in the more lowly-rated PGP) and node 1 participates in the more highly-rated PGP. Node 2 builds the same view of this PGP, including that node 1 will postpone activities on this PGP to provide predictive information to node 2 via the less highly-rated PGP. On the other hand, node 3 only processes the more highly-rated PGP because this PGP specifies node 3's current activities. Node 4 processes both PGPs to locally form their solution-construction-graphs so that it can identify whether it is expected to integrate results (which it is).

The nodes pursue their planned activities until time 21 (the node-plan node 1 sends at time 14 does not change the PGPs nodes are following since it agrees

with how nodes view what node 1 should do). At time 21, however, node 1 forms and transmits the predictive information d'_7 d'_8. Because it has formed the predictive results, it no longer needs to pursue the less highly-rated PGP. It resumes work on data in d_{10} to cooperate in the more highly-rated PGP. At time 23, node 2 receives this predictive information, divides its plan into two plans (as detailed more fully in the first example), sends out node-plans about these plans, and works purposefully on data that is compatible with the received information. These node-plans are received by the other nodes and incorporated into their PGPs, but do not affect how they pursue their activities.

At time 30, node 1 completes its partial track d_9–d_{11} and sends it to node 4. It then resumes work on the less highly-rated PGP, and forms and sends the partial track d'_6–d'_9 at time 38. Meanwhile, thanks to the predictive information from node 1, node 2 has also formed its partial track d'_1–d'_5 at time 38, and sends it to node 4. Node 4 receives these partial tracks at time 40 and combines them into a solution at time 41. Node 3 forms its partial solution d_1–d_8 at time 42 and sends it to node 4, which combines it with the result it has already gotten from node 1 into the other solution track d_1–d_{11} at time 45.

In contrast to this example, consider the minor variation where we do not allow the PGPlanner to postpone activities for more highly-rated PGPs in favor of generating predictive information for more lowly-rated PGPs. In this experiment, the nodes behave the same as before until time 14, when the PGPlanner forms somewhat different PGPs. In particular, node 1 continues working on the more highly-rated plan. In this case, node 1 forms the partial result d_9–d_{11} at time 22. Even though this is substantially earlier than when node 1 formed this result in the initial variation (time 30), this does not improve when the overall solution is found since node 3 still forms its piece at time 42 and node 4 does not combine them until time 45. In this variation, however, node 1 does not form and send the predictive result d'_7–d'_8 until time 31, so node 2 gets a later start at being more specific about the data it should process. Node 1 forms the partial solution d'_6–d'_9 at time 38 but, because the predictive information arrives later, node 2 forms the partial solution d'_1–d'_5 at time 47 and node 4 therefore forms the overall solution at time 50. Whereas with the ability to postpone activities in the first variation the network found both solutions by time 45, when this ability it taken away the network cannot form both solutions until time 50.

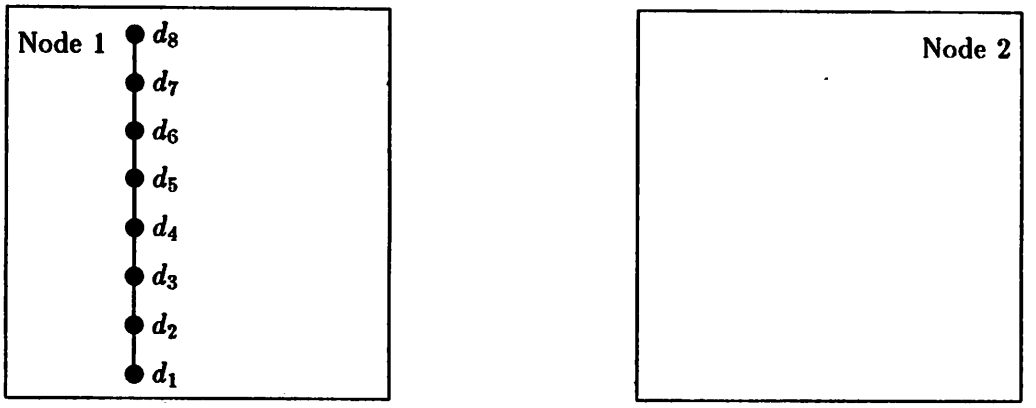


Figure 82: A Situation Needing Task Passing.

Finally, the ability of the PGPlanner to choose an appropriate node for integration is shown in these examples. Based simply on the communication delays, the pieces of overall solutions could soonest be together at nodes other than node 4. That is, if node 1 sent the relevant pieces it forms to nodes 2 and 3, then each of those nodes would have the two partial solutions to combine as soon as it had formed its local piece. Node 4 could have both pieces 2 time units later because of communication delays. However, the PGPlanner determines that node 4 can form the combined solutions earlier despite its later start because its integration expertise (KSs) are faster than those of nodes 2 and 3. Node 4 takes 1 time unit to combine pieces while nodes 2 and 3 would take 4 time units, and their head start is negated.

Example 3: Task-Passing

In this example, we consider a simple network with two nodes that have non-overlapping sensed areas. In the initial situation, a vehicle passes through node 1's area and node 2 gets no sensor data of its own, as shown in Figure 82. We assume that the nodes exchange information with a delay of two time units, that each KS execution takes one time unit, that the data arrives in the first time intervals, and that the nodes have equivalent KSs. The meta-level organization allows nodes to broadcast node-plans, and to transmit PGPs to each other that represent potential task-passing. The time-cushion is set to 0, and the min-task-duration is 8 time units (so sequences of activities that could take 8 or more time units could be passed).

In this situation, node 1 acquires its data over the first 8 time units, and problem solving begins at time 9. Node 1 forms its local plan to generate d_1–d_8, and orders the i-goals (1 2 3 4 5 6 7 8) to work from the earliest sensed time to the latest (since the data is equally rated throughout). It builds and transmits a node-plan for the plan, and develops a PGP corresponding only to this local plan. Meanwhile, node 2 can build no local plans (it has no data) and forms an idle node-plan that it transmits.

At time 11, node 1 receives the idle node-plan from node 2. Node 1's PGPlanner now can recognize that node 2 is underutilized (before it gets the idle node-plan it cannot be sure). It examines the (single) highly-rated PGP and recognizes that node 1 is the bottleneck node. It determines the earliest time that node 2 could possibly begin received tasks as the current time plus 3 times the communication delay between the nodes (corresponding to the time to send the PGP to node 2, then to get the node-plan back, and then to send the data to be processed). Therefore, node 2 could start tasks at time 17. Based on its current predictions, node 1 expects to complete its local plan at time 40. The PGPlanner recognizes from these computations that if it split the tasks between time 17 and time 40 between the two nodes, then the sent task would take approximately 11 time units which is large enough to be transferred.

Having determined that it can transfer a task, the PGPlanner next builds that task to transfer. It extracts the plan-activities for node 1 (the overutilized node) from the PGP's plan-activity-map. Starting with the last plan-activity (farthest in the future), it works its way backward until it has found a set of plan-activities that meets three criteria. First, the sum of their durations does not exceed the expected task time (in this case 11 time units). Second, adding one more plan-activity would cause the set to exceed the expected task time. Third, they are related in that their results can be combined into a single partial track.[17] The set of plan-activities that meet these criteria in this example are to work on d_8, d_7, and d_6, generating the partial result d_6–d_8.

With these plan-activities, the PGPlanner modifies the PGP's plan-activity-map. It makes copies of these plan-activities and indicates that each would be done by some as yet unassigned node. It also modifies their begin and end times:

[17]To achieve this third criterion, the PGPlanner may skip over some plan-activities if their results could not be combined with those of the plan-activities already chosen.

the first plan-activity is modified to begin at the time that the recipient node(s) could start working on the task (in this case, time 17), and the begin and end times of subsequent plan-activities are shifted appropriately. These plan-activities are then interleaved into the PGP's plan-activity-map. The PGP's future attributes are also updated: the to-nodes attribute is the list of nodes that could potentially do the task (in this case, node 2), the respond-to attribute is set to the current node (in this case, node 1), and the response-label is set to the name of the PGP. This PGP is then transmitted to the potential recipients (node 2).

Node 2 receives this PGP at time 13, and by examining the future attributes recognizes that the PGP represents a possible transfer of tasks. Because node 2 is named in the PGP's future:to-nodes, it generates a future node-plan to respond to the PGP. First it extracts from the PGP's plan-activity-map the plan-activities that are marked as unassigned. It steps though each of these and computes how long *it* expects to take to do each, where it takes the duration that came with the plan-activity (difference between begin and end times) and factors in the relative expertise in synthesizing data (from its node-models). Given a modified view of how long the task will take, the recipient node's PGPlanner then scans its node-model of itself to see what other node-plans it has for local activity. These node-plans include any other future node-plans that it has sent in response to other PGPs. In addition, the PGPlanner uses any PGPs to hypothesize future node-plans based on the trajectory of tracked vehicles, but in this case no vehicles appear to be moving its way. With all of these node-plans, the PGPlanner looks for an unassigned time-interval that is long enough to achieve the possible new task. Given such a time interval (after it expects to complete all its other plans, if not sooner), the PGPlanner shifts the begin and end times of the received plan-activities to represent when it believes it can pursue them and how long it expects each will take. In this example, node 2 has only an idle node-plan and has equivalent KSs to node 1, so the received plan-activity-map is not altered.

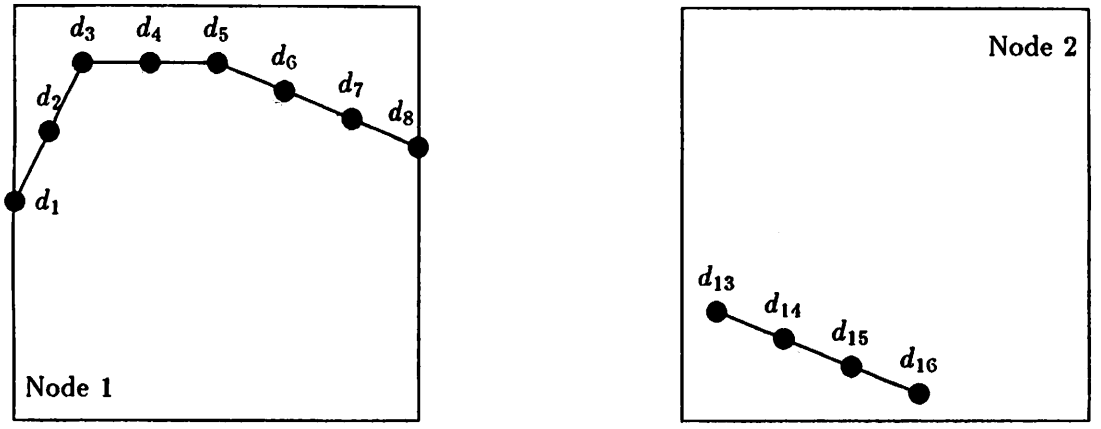
The PGPlanner forms a future node-plan for this modified plan-activity-map. It computes the objectives from this plan-activity-map (their combined results) and the PGP (the vehicle types). It rates the node-plan based on the PGP's rating and the credibility in the sending node. It also sets the node-plan's future attributes, using the begin-time of the first plan-activity as the future:start-time and setting the triggering-PGP attribute to the PGP's response-label. In this

case, for example, the future node-plan's objectives are to form d_6–d_8 and the start time is 17. This node-plan is sent to the respond-to node indicated in the PGP, in this example node 1.

Node 1 receives the node-plan message at time 15. By examining its future attributes—in particular, the triggering-PGP—node 1 recognizes that the node-plan is in response to the transmitted PGP. It inserts the received information in the PGP's future:responses attribute. The PGPlanner determines that it need not wait for any more responses (enough time has elapsed for all responses to have arrived) and examines the PGP to decide whether the task should be awarded and where. It compares the plan-activity-maps of the PGP with the received node-plan's activity-map. Since they match up (the recipient node expects to do the activities at the expected time), the PGPlanner decides to award the task. It replaces the marker in the PGP's unassigned plan-activities with the name of the node that will perform them (in this case, node 2). The PGP's future information is updated to indicate that data in d_6, d_7, and d_8 should be sent to node 2. Because node 1 has this data and is the overutilized node, it triggers communication actions that send the hypotheses in these regions. Node 1 also sends the modified PGP back to node 2.

At time 17, node 2 receives the modified PGP and recognizes that it should accept the data from node 1. It receives the hypotheses sent by node 1, and treats these as any other hypotheses: the local planner clusters the data and forms the local plan to generate d_6–d_8, and the PGPlanner incorporates this plan into its network-model. The PGP is updated: the solution-construction-graph builds the expectation that node 2 will form d_6–d_8, that node 1 will form d_1–d_5, and that these pieces will be integrated into an overall solution at node 1. The communication expectations represent these expected interactions.

From time 17 to time 28, node 2 builds its partial solution. Meanwhile, node 1 continues on its plan and builds its partial solution also at time 28. Node 2 sends this partial solution to node 1, which integrates them into the overall result at time 32. In comparison, if node 1 had formed the entire track by itself, it would have finished at time 41. By transferring tasks, the nodes make better use of network computational resources and solve problems faster. Note that the communication delays prevent the nodes from taking more advantage of these resources because they need to exchange a certain amount of information before



Data d_1–d_8 arrives at the start of problem solving, but data d_{13}–d_{16} does not arrive until simulated time 13.

Figure 83: Predicting Future Plans

they can transfer tasks.

As a second variation on this example, consider what happens when the vehicle moves across both node's sensed regions but where the sensors supply data in bursts (Figure 83). Node 1's data arrives at the outset of node activity, so it again forms a plan to generate d_1–d_8. Meanwhile, since node 2's data does not arrive until time 13, node 2 initially believes that it is idle. In this variation, we limit nodes so that they can only predict future node-plans for themselves rather than for any nodes based on PGPs. This saves in overhead, but, as we shall see, may cause nodes to have different views.

At time 9, node 1 forms its plan to generate d_1–d_8 and node 2 forms an idle node-plan. At time 11, node 1 receives the idle node-plan from node 2 and, as before, recognizes that perhaps a task should be passed. It builds up the task for node 2 to do d_6–d_8 and sends the modified PGP to node 2.

When node 2 receives this PGP at time 13, it builds the future node-plan for the possible activities as described before. However, before node 2 assigns begin and end times to the plan-activities, it first identifies any other node-plans that it is pursuing or may pursue. It extrapolates the track of the PGP to discover that, in fact, the vehicle will likely pass through its area at any time now. Node 2 forms a future node-plan whose start-time is 13 and whose objective is the extrapolated track. It expects the future plan-activities will take as long as the plan-activities

that it could receive from node 1, and from these expectations develops future expectations that from time 13 to time 29 it will be pursuing a local plan from sensed data. As a result, it shifts the begin and end times of the task's plan-activities so that the task starts at time 29 and is expected to end at time 41. It sends the future node-plan with this information back to node 1.

When node 1 receives the future node-plan in response to its PGP, it associates the information with the PGP in the same way as before. However, when the PGPlanner examines the received node-plan to see when the plan-activities can be carried out, it discovers that the result of the task if passed would be formed later at the recipient node (time 41) than at the node where they currently reside (time 40, locally). It therefore decides not to transfer the task at all: it deletes the responses from the PGP's future:responses attribute. Node 2 receives the data at time 13 and begins building track d_{13}-d_{16}, while node 1 builds track d_1-d_8 by itself.

This brings up questions about what happens when expected future node-plans in fact do not occur. In the current implementation, a node's PGPlanner will intermittently examine its node-plans to find any whose start-time has passed and who still do not correspond to an existing node-plan. The PGPlanner removes these node-plans, and scans the PGPs to see if there are any that had requested responses from this node. If some exist, the PGPlanner forms a new future node-plan for each of these (since the loss of future node-plans that did not pan out could affect when plan-activities can be done) and sends this future node-plan. Thus, a node can essentially change its bid when its circumstances change. Note that once a task has been assigned, the modified PGP is sent to the nodes that were asked for responses so that they can delete this PGP from the set of PGPs to which they have responded (so that a change to their node-plans does not cause them to respond to obsolete PGPs). Although these simple mechanisms for handling uncertainty in task-passing are sufficient for our experiments so far, they represent an area where more study and experimentation is needed.

In the third and final variation of this example, we give each node the ability to predict future plans for other nodes. Given exactly the same data as in the second variation, the nodes once again exchange node-plans at time 9. At time 11, node 1 again identifies that it is overburdened while node 2 is currently idle. However, in this case node 1 takes the additional step of using its PGP to recognize that

node 2 will likely get a future node-plan. From this view, node 1 realizes that node 2 does not have enough time to perform the task before it gets data of its own, and node 1 therefore does not modify and send the PGP in the first place. By incurring the extra overhead of recognizing future node-plans for other nodes, node 1 saves on communication and computation overhead in this case.

7.4 The Partial Global Planning Mechanisms in General

After all of this detailed information about how the PGPlanner works and what it does in particular examples, the reader has hopefully developed an understanding of how partial global planning works in the distributed vehicle monitoring domain. In more general terms, the basic concept of partial global planning—to summarize and exchange local plans, recognize when nodes should cooperate, and give them information so that they can change their local activities to improve network activity—is very straightforward. In fact, it is so conceptually simple that its more general applicability at the conceptual level should be obvious. However, as we have seen, developing an implementation in a particular domain can be a sizable task. It is the purpose of this section to give a more general view of these mechanisms and to suggest ways that they can be applied in other domains.

7.4.1 Generalizing the Partial Global Planner's Components

A general view of mechanisms can be misleading because, from a high-level, many crucial details might be missed—details that affect whether the mechanisms do, in fact, represent a viable approach in a practical system. By describing in detail how the concepts have been implemented in the distributed vehicle monitoring domain, we have attempted to make more clear what they do and how they do it, addressing issues not only in how the concept of partial global planning corresponds to particular mechanisms but also the issues we faced and the decisions we made while implementing these mechanisms. The details of the mechanisms are intended to provide guidance to others that hope to use similar mechanisms for other applications: although they will doubtless come upon issues that we have not faced in our implementation, they can make use of our experience when

facing analogous implementation questions in their systems. Because we cannot foresee all possible variations of domains and implementations, we have detailed our experience so that others can hopefully extract more general aspects of the mechanisms that they need. However, in this section we attempt to generalize the mechanisms and the approach as a first step in helping others implement partial global planning mechanisms in different systems.

Generalizing the PGPlanner's Data Structures

The whole point of representation in the planner and the PGPlanner is finding the right level of detail, and our particular decisions about what the right level is has been biased by our problem domain. In a closely-coupled environment where communication is fast and inexpensive, nodes might cooperate effectively by coordinating specific actions: they could essentially form a global schedule of actions to follow. In an environment with extremely loose-coupling, nodes might communicate only about overall local plans, so that they can decide how the final results of a plan might affect network activity. Our view rests between these extremes, because we want nodes to coordinate larger groups of actions (intermediate-goals) within plans, without coordinating any specific actions. No matter what the particular aspects of the environment are, however, the data structures for control need the ability to represent activities at different levels of detail: a node needs a lot of detail about its own activities (to choose a specific action to achieve its goals), and less detail about the basic activities of others (what their goals are and how could their actions to achieve these goals affect this node).

Our representation of PGPs allows nodes to represent information at different levels of detail, so that it has suitable views for the different decisions it must make. For example, to recognize how nodes should cover data that they both sense (whether to avoid duplication of effort or to verify each others results), they need to have a rough view of what data each will be working on and when. This view is provided by the plan-activity-map. On the other hand, to decide how they should interact to combine individual results, they do not need details about the order in which each constructs its local result, but instead simply need to know when the desired local result will be formed. This view is provided by the solution-construction-graph. Both of these data structures provide information

at the right level of detail for making some control decisions, and the purpose of the PGP representation is to give nodes these different views.

In more general terms, a PGP contains some representation of group objectives. In the vehicle monitoring case, this is where vehicles are being tracked and what types of vehicles they are. In other domains, the objective might be to cooperatively construct a bridge, to cooperatively explore a region, or to cooperatively diagnose a network fault. A PGP also contains a a plan-activity-map to represent the concurrent activities of nodes. Depending on the environment, these activities could be simple actions or they could each be complex sequences of actions.[18] Because the nodes may interact infrequently (rather than after each activity), the PGP represents their interactions in a separate structure—the solution-construction-graph. Finally, because nodes may want to explore possible future cooperative activities (such as transferring tasks so that new nodes can join in important group activities), the PGP allows nodes to represent possible future activities so that they can reason about how these could influence local and network activity.

The other important data structure used for partial global planning is the cooperation-parameters structure. The choice of important cooperation parameters is in many ways domain dependent, influenced by which ways of cooperating are desirable (or undesirable) in a domain. However, the choices made for our implementation also have a certain degree of task independence. In any domain, for example, cooperating agents need information about the importance of duplicating activities (redundancy and reliability), of helping each other out (predictiveness), of staying out of each other's way (independence and diversity), and of committing to group activity versus adapting to changed local views (time-cushion). *What* information is represented in the cooperation-parameters, though by no means exhaustive of all possible information, is therefore fairly general. *How* the information is represented—as numerical parameters—is less likely to be the best representation in other domains. In fact, the PGPlanner in the vehicle monitoring domain could make better decisions if this information was had a richer, symbolic representation, but for the purposes of developing an

[18] A PGP could in fact represent the combined intentions of nodes where each node is working on a group of plans. The plan-activity-map might therefore represent sequences of plans rather than intermediate-goals of a single plan, and the nodes could coordinate the order in which they pursue these plans.

initial implementation, the numeric view has been satisfactory.

7.4.2 Generalizing the Plan-Activity-Map Generation and Manipulation

Representations similar to the plan-activity-map have been used to represent concurrent activities in other domains [Allen, 1983; Allen, 1984; Cheeseman, 1984; Dean, 1986; Vere, 1983], and using begin and end-times for activities to coordinate nodes is a generally useful mechanism (so long as time n means about the same thing to one node as it does to another). Moreover, given a representation of local plans that allows nodes to summarize and selectively exchange information (as discussed in Chapter 6), our method of generating the plan-activity-map using expected durations of activities is very straightforward.

The mechanisms for manipulating plan-activity-maps to avoid interference can also be generalized. Scanning through a pair of plan-activity-maps to find cases where they expect a node to be doing two things at once means simply comparing intervals for activities of the same node. When such a situation occurs, then activities are shifted to other times by altering their begin and end times. The techniques for moving activities of particular duration around within a window of time is a common problem in scheduling [Stankovic *et al.*, 1985] and in planning [Vere, 1983]. However, using plans about likely interactions between cooperating nodes to determine the extents of time-windows is still a little understood topic that our mechanisms contribute to exploring.

Finally, our mechanisms for reordering activities find a cost for each and iteratively move them around to minimize total cost. Cooperating nodes must have a basis for considering some activities better (less costly) than others based on how the activities fit into the overall network activities. Our mechanisms represent this preference numerically and simply move better activities to earlier times, postponing worse activities until later (if ever). Assuming that a suitable rating function (combining all important considerations into a single value) can be found in other domains, the hill-climbing algorithm used in the current implementation could be more generally applied. In our future research, we hope to improve on these techniques to find an alternative way to reorder activities (based on symbolic representations of their utility) that is more intelligent and less costly. It is important to note that the choice of algorithms used to reorder the activities

is irrelevant to the rest of the PGPlanning mechanisms which simply use whatever plan-activity-map is generated, so different techniques can be incorporated modularly into the PGPlanner.

7.4.3 Generalizing the Solution-Construction-Graph and Domain-Level Communication

The mechanisms for generating the solution-construction-graph assume that the preferences for network redundancy, that communication delays, and that estimates of nodes' integration expertise are known so that it can formulate a reasonable (though not necessarily optimal) prediction about how results should be combined into the complete solution. It begins by grouping together sequences of related activities to identify the combinable results made locally by the nodes. In more general terms, it finds groups of activities, where each group triggers some form of interaction. For example, in an assembly domain the group of activities may lead to a complete sub-assembly, and the next step would be to combine this with a sub-assembly made by another agent. Similarly, in a diagnosis task the group of activities might lead to a local hypothetical diagnosis which must then be sent to other nodes for corroboration. The criteria for grouping activities depends on how the nodes expect to cooperate and the relative time that nodes are performing activities, and is domain dependent.

Once these groups have been found, the simple mechanisms for determining when and where each pair of groups could be joined and for building up larger and larger combinations are applied. These mechanisms are more generally applicable in other domains. In an assembly domain, for example, the mechanisms would identify when and where sub-assemblies could be united into a larger sub-assembly, and through iteration would determine how all of the pieces should be combined into the overall result. However, the mechanisms may need to be extended in some domains since they only consider pairwise combinations of pieces—some domains may need a group of more than two pieces at the same place and time before they can be combined. Such extensions are not needed in the vehicle monitoring domain (since the KSs combine two pieces at a time). They would be fairly easy to add to the mechanisms, but may raise issues in additional complexity: when considering pairs of partial results, the number of combinations to consider rises quadratically with the number of pieces; but when

considering all combinations of two or more, the number of combinations rises exponentially with the number of pieces.

7.4.4 Generalizing the Mechanisms for Task-Passing and Prediction

An important strength of the partial global planning mechanisms is their ability to allow contracting out of tasks (task-passing) within the same framework as the coordination of tasks already assigned to suitable nodes. Since they transmit PGPs, the task-passing mechanisms allow the PGPlanner to communicate not only about the characteristics of a task that could be transferred, but also about how that task fits into a more global view of cooperative activity. With this additional context, a node that responds to such a request can respond based both on its local view and on its view of the PGP's importance. In essence, a PGP message that represents a request for task-passing not only tells the possible recipient node what the task is, but also why it is important. The task-passing mechanisms also promote flexibility in how larger tasks are decomposed, since nodes use the PGP as context to decide how many and which activities need to be transferred in a given situation.

Because they allow nodes to form contracts (shared PGPs), these mechanisms for task-passing are generalizable to other domains where task-passing is important for effective use of network resources. When the tasks at different nodes are interdependent, a PGP indicates how the potentially cooperating nodes should coordinate their tasks. However, the mechanisms are also useful in domains where tasks are independent, since a PGP for task-passing would simply reduce to information about the specific task to pass. The types of tasks to be represented and exchanged would have to be extended for other domains so that nodes can move more diverse tasks around, but, in any domain, nodes would want to represent possible transfers of tasks using PGPs. Before it can propose a task transfer, a node must have enough knowledge of the task to recognize how it fits into the more global perspective (whether it is relevant and not being done elsewhere) and what other nodes need to know to make intelligent decisions about whether they can perform the task. PGPs provide a useful representation for indicating the relevance of a task and how the task could be performed.

The mechanisms for predicting future tasks are very domain dependent, as well

as being rudimentary in the current implementation. How our mechanisms in the vehicle monitoring domain predict tasks is less important than the PGPlanner's more general ability to represent and use these future plans. The ability to represent possible future tasks based on current tasks can be useful in domains where tasks arrive over time and where what those tasks might be can be predicted from the current tasks. By representing and reasoning about predicted future activity, the PGPlanner can make better decisions about current actions (and about receiving tasks from others) so that the node is better prepared to pursue those future tasks.

Finally, note that the mechanisms represent the possible reception of tasks from other nodes the same way that they represent the possible local development of tasks. By viewing plans for future activities in a uniform way despite their different sources, the PGPlanner can more effectively reason about relationships between current problem solving and possible future activities.

7.4.5 Generalizing Meta-Level Communication

The meta-level communication mechanisms transfer information about node-plans and PGPs. Viewed very simply, these mechanisms extract the attributes of a local data structure, build a message structure from them, and send this message structure to another node which builds a local copy of the data structure with the attributes provided. The two basic issues that complicate these mechanisms is the need to keep track of what information is received when so that nodes can update this information, and the desire to be selective (make intelligent decisions) about what information is sent.

To keep track of the information exchanged, nodes timestamp their node-plans and PGPs (giving each a creation-time) and maintain information about local names of the plans involved. The names of local plans associated with node-plans and PGPs allow nodes to recognize when they get information about the same plans from different sources. By timestamping the information, nodes can compare different views of the same information (node-plans for the same local plan, PGPs for the same group of local plans) to decide which views are more current. These techniques are more generally used in networks (for example, distributed database systems) where time-dependent information must be propagated among nodes.

The mechanisms for making selective communication decisions identify when node-plans and PGPs have been updated, what versions were sent previously and where, and what nodes should receive the information (based on the meta-level organization). Because the decisions about updating node-plans are based on the time-cushion and PGPs are only updated when node-plans change, nodes can reduce meta-level communication by tolerating minor deviations in plans and only communicating when plans exceed the tolerance specified by the time-cushion. This technique is generally applicable to other systems where nodes should coordinate plans but can still achieve network goals when their plans get somewhat out of phase.

7.4.6 Generalizing the Connection Between PGPlanning, Local Planning, and Problem Solving

The PGPlanner, local planner, and problem solver are distinct parts of an overall node and, although they interact, the interface between them allows the implementation of one of these components to be changed without major revisions to the others. The purpose of the PGPlanner is to alter the local plans on the plan queue so that the best plan *from a global perspective* rises to the top of the queue. How this manipulation is done is irrelevant to the local planner, since it simply takes the top local plan from the queue and pursues it. Similarly, the local planner alters the KSI queue so that the best KSI *to work toward longer term goals* is invoked next. The implementation of the local planner is irrelevant to the problem solver itself when it invokes this KSI. The local planner needs to know about the problem solver's representation of hypotheses, goals, and KSIs, and the PGPlanner must know about the local planner's representation of plans. However, the different components reason about these representations independently.[19]

This view of control in different but interacting levels is more generally applicable to other cooperating systems. The more a node needs to reason about how nodes interact, the more it needs to reason about its own actions. On top

[19]Each component also needs to know how decisions at the lower level are made so that it can influence them. The local planner needs to know that the problem solver invokes the most highly-rated KSI, so that it can influence problem solving by changing these ratings. Similarly, the PGPlanner needs to know that the local planner pursues the most highly-rated plan, so that it can influence the planner's decisions by changing plan ratings. A more complete discussion of the relationships between these levels of control is given in Chapter 1.

of its mechanisms to control basic activities, therefore, a node needs more levels of control to represent and reason about its own behavior and the behavior of others. Our mechanisms provide an initial pass at these levels of control: a level for controlling local actions to achieve immediate goals, a level for controlling sequences of local actions to achieve long-term goals, and a level for controlling concurrent sequences of actions at different nodes to achieve network goals. Although it may well be that other applications need a larger (or smaller) number of levels, the levels we have developed are likely to be some part of the control mechanisms in cooperative systems.

7.4.7 Other Applications of Partial Global Planning

The mechanisms for partial global planning have been implemented in the DVMT, and their utility in this domain is evaluated in the next chapter. The general characteristics of the domain that make these mechanisms useful were summarized in Chapter 1. In this section, we very briefly explore the generality of partial global planning by considering how it might be used in other application domains.

One possible application domain is in a cooperative assembly task, where several robots can work together to build products. Each robot may have a store of parts and knowledge about ways that it could combine those parts into larger assemblies. Each can therefore recognize local long-term goals—possible sub-assemblies—and can locally plan actions to achieve these goals. The plans can be ranked based on how important each sub-assembly is (perhaps giving preference to sub-assemblies that use the most parts or a particular type of part). By communicating about these local plans, the robots can recognize larger combined assemblies that they could cooperatively form, and they generate PGPs to represent these. Moreover, they could use the local plans to determine their concurrent activities and to decide when and where to combine their sub-assemblies, considering the time it takes for one robot to transport a sub-assembly to another robot. As they pursue their plans and their expectations change (tasks take longer than expected, parts do not fit together properly), they can update and exchange their plans to reconsider how (or whether) to cooperate. They could also recognize how, by providing some parts of a sub-assemblies ahead of time, they can constrain each other's activities (since the recipient would have to build a compatible sub-assembly). Finally, by identifying important PGPs, they

can also identify underutilized robots and develop modified PGPs that reflect how these robots could participate in the PGPs. Partial global planning among potentially cooperating robots therefore allows them to reason about when and how they should cooperate.

As a second example, consider cooperating diagnostic systems that are monitoring a communication network. Each has a view of part of the network and can locally recognize aberrant behavior within its view. To diagnose network faults that span the views of several nodes, however, the nodes must cooperatively develop a more global view of faults. The nodes begin by locally forming plans to diagnose errors they locally witness, and then exchange summaries of these plans. Given information about several nodes plans, a node can recognize when they are individually diagnosing errors that could be part of a more global problem, and would form PGPs to represent the cooperative diagnostic activity. By reasoning about their concurrent activities, nodes can avoid duplicating each other's efforts (such as testing a communication link), can verify each other's partial diagnoses (such as hypothesizing common faults), and can provide predictive information (such as a symptom that any diagnoses must account for). Nodes that are cooperatively tracking down some possible diagnoses might pass responsibility for tracking others to nodes that otherwise would not be helping network diagnosis. When some potential diagnoses are proven to be unpromising, when certain diagnostic tests take more or less time than expected, and when new symptoms develop that must be accounted for, the nodes must update their local views and exchange appropriate information to modify PGPs to work together effectively. In an uncertain and dynamic environment like large communication networks, partial global planning represents a promising approach for diagnosing (and possibly repairing) failures.

As a final sample application, a network operating system needs to coordinate the processing and resource use of a group of nodes. Tasks arrive at the various nodes asynchronously and possibly unpredictably (although repetitive tasks might also be part of the domain). Each task has certain resource requirements (including computation time), may be related to other tasks (for example, one may have to be done before another or only one of two tasks might have to be done), and may have a deadline. A node could therefore plan out how it will perform tasks. In the simplest case, the plan might simply be a schedule of what

tasks it will perform when. Alternatively, the node might consider groups of tasks (with similar deadlines or that are related in some other way) and plan roughly when it will pursue each group. Since it may not have to perform every task (some tasks may be redundant), a node faces uncertainty about which tasks it should perform and when, and resolves this uncertainty over time: it performs some tasks and uses their results to determine which of the pending tasks are worth pursuing. A node therefore plans its activity incrementally. If nodes exchange summaries of their plans and update these summaries as their plans change, they can form a PGP to represent their concurrent activities. Using this view, they could reorder local activities, considering deadlines and precedence as they do, and could move tasks to underutilized nodes. They could also predict interactions, such as deciding that rather than redundantly performing the same task at several nodes, one node would perform the task and then send any needed results to the others. Because of communication delays and the dynamics of the domain, the nodes may not be able to ever develop completely consistent and up-to-date views of network activity, and so their local scheduling decisions might deviate from the best choice had they more knowledge. However, by partial global planning (at an appropriate level of detail), the nodes could better coordinate their local activities to work as a more coherent team.

Many hands make light work. -Heywood

Too many cooks spoil the broth. -Anonymous

Chapter 8

Partial Global Planning: Experiments and Evaluation

In the first part of this chapter we briefly summarize experimental results that illustrate the benefits (better coordination of control decisions), costs (computation, communication, and storage overhead), and capabilities (meeting goals of cooperation, permitting different styles of cooperation, and balancing predictability with responsiveness) of the mechanisms. The second part of the chapter uses these experimental results to evaluate the mechanisms.

8.1 Partial Global Planning Experiments

The purpose of this section is to explore partial global planning in a variety of distributed problem solving situations. The experimental situations examine different issues in how partial global planning affects the quality and cost of coordination. These experiments are thus illustrative, not exhaustive, and are intended to show how well or poorly the mechanisms work in sample situations.

Most of the experiments examine various aspects of the new mechanisms, concentrating on the effects of different degrees of planning, different meta-level

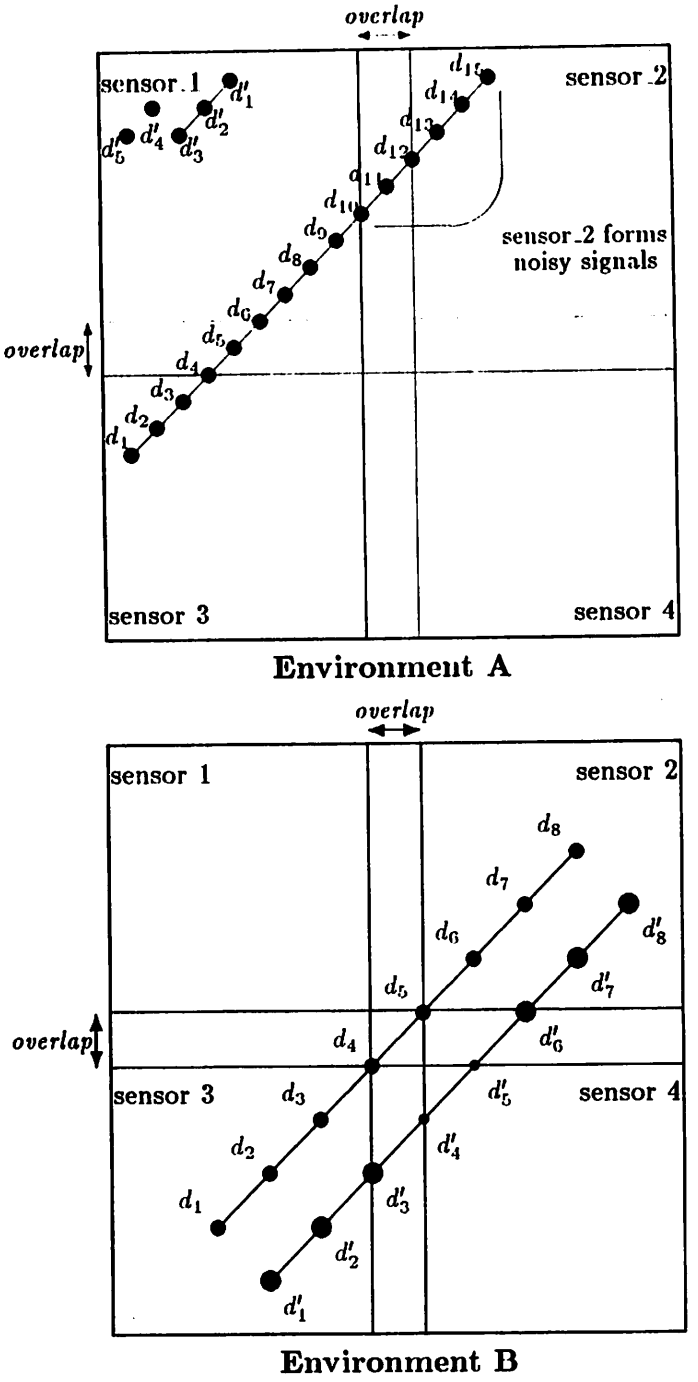
organizations, different degrees of dynamic change (data arriving over time), different balances of predictability and responsiveness, different node capabilities, and different degrees of node reliability. In addition, the experiments also explore larger networks (to get a feel for how well the mechanisms scale-up), the benefits and costs of task-passing through PGPs, and other miscellaneous issues such as postponing activities for one PGP to help in another PGP and dealing with more ambiguous environments.

Experimental Environments

The majority of the experiments are conducted on four-node networks, although experiments involving both larger and smaller networks are also discussed. The two principal environments are shown in Figure 84. Environment A should hopefully be familiar by now, since it was used for examples in the previous two chapters. Its important features are the track shared by several nodes (d_1–d_{15}), the much less important (but moderately strongly sensed) data of node 1 (d'_1–d'_5), and the ambiguity introduced into node 2's data by its sensor.

Environment A focuses on several issues: giving preference to more globally important plans, providing predictive information, and avoiding redundant processing. We have built the environment so that the track d'_1–d'_5, though it locally looks promising to node 1, is unimportant because it is not globally relevant and because the sharp turns that it involves cause the track to be lowly rated. Hence, we want the network to form the track d_1–d_{15} as quickly as possible and do not care whether it forms d'_1–d'_5. Providing predictive information is important so that node 2 can better deal with its ambiguous data. Recall that when node 1 sends a predictive hypothesis (such as d_8–d_9), node 2 can use this information to break its single plan into two plans: one to form partial tracks that can be joined with the received information (of the suitable vehicle type) and one to form the other partial tracks. Avoiding redundancy is important so the nodes do not generate duplicate hypotheses for data in their overlapping areas.

Environment B, shown in Figure 84, has two vehicles passing among the nodes. To better emphasize issues in coordination (rather than in resolving local uncertainty about what data to combine), we assume that the vehicles are far enough



The four overlapping sensors detect signal data at discrete sensed times (the dots with associated times). In environment A, Sensor 2 is faulty and not only generates signal data at the correct frequencies but also forms noisy signals at spurious frequencies. In environment B, two vehicles move in parallel with each other.

Figure 84: Four-Sensor Configuration with Sensed Data.

apart so that there is no confusion about which data belongs together.[1] The data for one of the vehicles is moderately sensed at each sensed time, while the data for the other has strongly and weakly sensed sections. When the node combines the data into overall tracks, the track made of moderately sensed data (d_1–d_8) is a little more highly believed than the track with strong and weak parts (d'_1–d'_8). We therefore consider d_1–d_8 to be the better solution and d'_1–d'_8 to be the worse solution. The network should find both solutions, but should find the better solution first.

Control of coordination is difficult in environment B for several reasons. The overlap between nodes causes them to receive much of the same data, so nodes must allocate tasks so they perform tasks that generate new (rather than redundant) results and they should combine these results in an efficient manner. The fact that there are two solutions also complicates matters because different groups of nodes cooperate on different solutions: all the nodes have data for d_1–d_8 but node 1 does not have data for the other solution. The nodes need to coordinate to promote work on each of the solutions while avoiding redundancy and giving preference to the better solution.

Assumptions and Default Values in the Experiments

There are many assumptions and default values for environmental attributes that hold throughout the experiments except where changes are explicitly noted. First, we assume that domain-level communication is limited to vehicle-track hypotheses (to reduce communication), the only exception being during task-passing when nodes can also exchange signal-location hypotheses. Domain-level and meta-level communication between nodes is broadcast, message delay is 2 time units (for all messages, no matter what their size), and messages are not lost. Although broadcast communication means that messages sent by one node arrive at all other nodes, a node does not necessarily incorporate a message's information. For example, a node compares a message about a hypothesis against some criteria when deciding whether to actually build the local version of that hypothesis. Without the PGPlanner, these criteria are specified in the organizational struc-

[1] Unlike similar environments described in other papers [Corkill, 1983; Corkill and Lesser, 1983; Lesser and Corkill, 1983], the nodes *cannot* form cross-tracks that connect data in different solution tracks together.

ture, while with the PGPlanner the node checks PGPs to see if it is supposed to incorporate the data.[2] Similarly, when a node receives a coordination message (a node-plan or PGP), it checks the meta-level organization to decide whether to actually incorporate that message. Thus, although messages are broadcast, nodes are selective about whether to incorporate the message information, since incorporating it can be a costly activity.

In local problem solving, each KS takes 1 time unit to execute, so that each time unit corresponds to a local control decision. An important consequence of this is that the simulated time does *not* reflect the time spent on control—modifying the DVMT to simulate time cost for all control activities (forming goals and plans, running KS preconditions, etc.) would be a major effort in itself.

Network problem solving terminates after every solution has been found by at least one node. Because our experiments are complicated enough already, we use the simple termination criterion of specifying the desired solutions beforehand. Although they do not use this information to guide control decisions, the nodes compare the hypotheses they form against this information to determine when they have found the answer(s) and can stop problem solving. Thus, in environment A the network ends problem solving as soon as one node has formed the track d_1–d_{15} at the pattern blackboard-level, while in environment B, problem solving ends as soon as some node has formed d_1–d_8 and some (possibly different) node has formed d'_1–d'_8, both at the pattern blackboard-level. Using this termination criterion also lets us compare networks with and without the local and partial global planning mechanisms, since the more advanced termination techniques are only possible with planning.

In experiments without local and partial global planning, the nodes are permitted to use all other control mechanisms. In particular, the nodes can use all of the goal processing mechanisms to form subgoals so that they can make better control decisions.

The default goals of cooperation cause the PGPlanner to avoid redundancy and to promote the formation and exchange of predictive information (the redundancy and predictiveness cooperation-parameters are greater than the reliability

[2]Messages that are not incorporated are saved and checked again if PGPs change just in case later on the node should incorporate the data. To reduce the storage, these saved messages are periodically scanned and any subsumed messages (the hypotheses they represent are subsumed by hypotheses of other messages) are deleted.

and independence parameters). The time-cushion is 1, so nodes tolerate deviations from predicted times of interactions that do not exceed 1 time unit. The solution-construction-redundancy is also 1 (no redundancy). To avoid complications of task-passing (except in the specific experiments designed to study task-passing), the min-task-duration defaults to a large value (100) where possible tasks in these environments always fall short of this value. The default meta-level organization has nodes broadcast node-plans and form their own PGPs. Finally, in the experiments that do not use the new PGPlanning mechanisms, the domain-level organization constrains communication decisions by having nodes *choose* to receive hypotheses only when they can extend those hypotheses. Thus, the communication channels are broadcast but nodes are selective about what communicated hypotheses they actually incorporate locally, and this saves them the expense of processing hypotheses that they are less likely to find useful.[3] With the PGPlanning mechanisms, these constraints are removed so that an idle node can receive results to integrate (when specified in its PGPs) regardless of whether it can locally contribute to the overall result. The PGPs, rather than the organization, guide communication decisions.

Experimental Measurements

To summarize and evaluate network problem solving, we measure several aspects of network activity: the simulated time to generate solutions, the actual time to generate solutions, the amount of communication (information exchange) in the network, and the amount of storage used by the network as a whole. We discuss each of these in turn.

Simulated Solution Time. As described in Chapter 2, the DVMT simulates time by assigning time costs to each of the KSs. Whenever a KSI is invoked by a node, the node's simulated clock is incremented by the simulated time needs for that KSI. In these experiments, we assign a simulated time cost of 1 time unit to each KS, so simulated time corresponds to the number of KSs a node has invoked. As a result, we can equate simulated time with control decisions: since each KS takes 1 time unit, a node that works for t time units has made t control

[3] Without these constraints, nodes in networks of ten or more nodes are overwhelmed by the amount of received information to process.

decisions. Because each node maintains a separate clock, the nodes are simulated to perform KSs in parallel. If the network finds the solution at time t, therefore, it means that *each node* has invoked as many as t KSs (although some nodes may have invoked fewer KSs if they ran out of data to process). In a network that finds the solution in less simulated time than other networks working on the same problem, the nodes have made better control decisions in parallel and have coordinated these decisions so that overall network control is better and the total number of KSs invoked in the network is reduced.

We measure simulated network performance as how quickly the network finds the solution(s) in simulated time. With better mechanisms for control (local and partial global planning), network performance should hopefully improve since nodes should invoke fewer unnecessary KSs and should exchange partial solutions in a more timely manner. In environments with a single solution (such as environment A), the network performance is based on how quickly the network forms that solution. When there are several solutions (such as in environment B), network performance is based both on how quickly the network forms all of the solutions *and* how quickly it forms the better solutions. That is, we view the network as supplying solutions to some consumer of solutions whenever it forms them, so the network should generate and send better solutions as soon as possible rather than timing its activities to form and send all solutions at once. In the experiments with two solutions (environment B), we therefore compare the simulated times that different networks need to form each of the solutions separately, and the maximum of a network's simulated solution times is its overall network performance.

The simulated network solution time(s) thus represent a measure of the quality of control in the nodes and in the network as a whole. Because control decisions are simulated to have uniform costs (since KSs have identical time needs), however, this measure does not adequately represent the costs of control in the network. Since we wish to understand the costs of our new control mechanisms (local and partial global planning), we must measure something besides simulated network solution time(s).

Computation Costs. While simulated solution time lets us measure the benefits of our new mechanisms (in improving control and coordination), we also

must measure the computation costs of these mechanisms since the time a node spends on control activities is time lost to actual problem solving. To measure the computation costs in the network, we use the actual computation time needed to simulate network problem solving. The time needed by the simulator, running in CLisp on a VAX 11/750 (see Appendix A), represents how much computation the network needed as a whole to find the solutions. This computation includes time spent on both problem solving and on control.[4] By measuring the overall computation time of DVMT experiments, we can compare the computation needed by networks using the various control mechanisms to determine whether the additional computation overhead introduced by more sophisticated control (local and partial global planning) is compensated for by a reduction in computation spent on other aspects of problem solving (since fewer KSs are invoked and less goal processing is done).

There are advantages and disadvantages to using this measurement of computation costs. A principal advantage is that it reflects costs in a real implementation of the mechanisms. Although the DVMT and the new mechanisms have been implemented with an emphasis on flexibility rather than efficiency (to facilitate modifying and improving them), the computation costs of experiments do represent the costs in a possible implementation. When costs of different activities are simulated, on the other hand, the experimenter can assign costs to elicit desired phenomena.

Another advantage of measuring computation cost as actual runtime is that this measurement can be taken without complicating the simulation. As mentioned above, by equating the simulated runtime with the number of control decisions, it is easy to recognize when control is better or worse. If the costs of control were to affect simulated runtime as well, then this measurement would be influenced both by improvements in control and by costs of control, making it much more difficult to understand benefits and costs separately. It would also make description of concurrent activity in nodes a much harder task. In a first pass at understanding the costs and benefits of our new mechanisms, simplifying the measurements allows us to understand phenomena without being overwhelmed.

There are several disadvantages of measuring computation cost as actual run-

[4] Because of the large granularity of node activities, the time spent context switching between nodes is negligible relative to the time spent on control and problem solving.

time rather than simulating the costs of control and problem solving. First, it only allows us to measure overall computation in the network, and does not reflect possible parallelism in control. For example, consider the costs of coordination when one node builds PGPs for the network versus when each node builds its own PGPs. In overall network computation, the former approach is much less costly since only one node is incurring the overhead of forming PGPs, as opposed to the latter approach where all nodes form PGPs. However, in the latter approach the simulated nodes would actually be forming PGPs in parallel, so although the network is doing more *computation*, it is not necessarily spending more *time* on control. When we discuss computation costs, therefore, it is important to remember that a higher computation cost implies more resources spent on computation but not necessarily more time spent in the simulated network.

Another disadvantage is that the computation spent on control cannot affect coordination. Since coordination among the nodes is simulated and the timing of their activities is based on simulated time, doing more or less control activity does not affect when nodes interact. More realistically, a node that spends time on control will pursue its problem solving actions later, and so may be delayed in when it sends expected partial results to other nodes. The PGPlanner can anticipate extra time spent on control by increasing its time-cushion. Since the time-cushion specifies a certain amount of flexibility in when results are formed are exchanged, a larger time-cushion builds extra "slop" time into the planned interactions of nodes so deviations caused by control activities can be tolerated.

Without the ability to simulate different costs for the various control activities, an experimenter cannot explore the ramifications of having more or less costly control activities. The experimenter cannot see what would happen if, for example, planning required half as much computation. By measuring actual computation we can determine the tradeoffs in control versus problem solving given the current implementation but cannot explore what would happen if the implementation were significantly changed for better or worse.

Clearly, in the long run we would like to improve the simulation of time in the DVMT to account for control activities. This is a sizable research task in itself, however, and is beyond the scope of the research we present here. For our purposes, therefore, it is enough to use actual computation time as a preliminary indication as to whether the improvements to control are cost effective in our

implemented system (whether the extra computation spent on local and partial global planning is offset by computation savings in problem solving since fewer KSs need to be run).

Communication Costs. Communication involves two types of costs. First, there is the cost for accessing communication channels. Second, there is the cost for processing communicated information—for building messages from local information at one end and for building and incorporating local information out of messages at the other. Incorporating received information can be especially costly because it can trigger substantial amounts of control activity so that a node knows what to do with that information.

In our experimental networks, we simulate a broadcast communication channel. A message sent by a node arrives at all other nodes. The other nodes need not incorporate the received information, however. Nodes consult their PGPs (or domain-level organization information if there is no PGPlanning) when deciding whether to incorporate received hypotheses. When exchanging meta-level information, nodes consult the meta-level organization to make such decisions. Simply measuring the number of messages transmitted therefore may not adequately reflect all of the costs of communication. To get a better measure of communication as information that has actually been transferred, we count the number of messages that are received *and incorporated* by nodes in the network. A single message sent over the broadcast channels might correspond to several reception events if more than one node incorporates the information. On the other hand, some messages may be incorporated by no nodes.

Communication is thus measured as the number of messages received and incorporated. Our measure of communication neglects other aspects, such as the size of the messages exchanged and how this affects contention for the channel as a function of time. As noted in the discussion of storage costs below, hypotheses, node-plans, and PGPs can have varying size, and thus so can their messages. Moreover, the size of the messages depends on how the information is encoded. Since encoding issues are not a focus of this research, we disregard actual message size and concentrate instead on the number of times nodes need to exchange information to coordinate and solve problems. This measure provides an initial basis for comparing the costs of communication in different networks.

Storage Costs. When nodes form and exchange plans to build PGPs, they have to keep track of more information. The costs of storing this information should not be overlooked. In the experiments, we measure storage costs by counting the number of major data structures that the nodes build both for planning and for problem solving. For problem solving, these structures are hypotheses, goals, and KSIs, while for planning these structures are clusters (in the clustering hierarchy), plans, node-plans, and PGPs. The actual size of the different structures vary with their contents and encoding: some hypotheses are bigger than other hypotheses; some hypotheses are bigger than some plans; some plans are bigger than some hypotheses; etc. Since we do not focus on encoding issues, we get a preliminary view of storage costs by simply counting the total number of structures in the network. This rough measure allows us to see how the new planning mechanisms incur some storage overhead (for clusters, plans, node-plans, and PGPs) but may also reduce overall storage costs by helping nodes avoid building unnecessary hypotheses, goals, and KSIs.

8.1.1 Experiment Set 8.1: Different Levels of Partial Global Planning

In the first set of experiments, we explore how the addition of the new mechanisms affect the quality and costs of cooperation. The environments we examine were described above and are shown in Figure 84. For each of these environments we run four experiments:

1. where neither the local planner nor the PGPlanner are incorporated in the nodes (only the original DVMT control techniques are used);
2. where local planning but not PGPlanning is incorporated;
3. where local planning and the ability exchange node-plans to recognize partial global goals are incorporated (so nodes can prefer to pursue more globally relevant plans but do not try to coordinate them);
4. and where all of the local and PGPlanning mechanisms are added to the nodes.

The experimental results are summarized in Table 7. In environment A, the lack of any planning mechanisms causes very poor performance (E8.1.1). The

Table 7: Experiment Summary for Experiment Set 8.1.

Expt	En	Mec	STime	Rt	H-r	N-r	T-r	Hyp	Goal	KSI	Cl	Pl	NP	PGP	Store
E8.1.1	A	none	171	465	44	-	44	1124	1771	698	-	-	-	-	3593
E8.1.2	A	local	81	76	17	-	17	761	268	373	222	13	23	28	1688
E8.1.3	A	pgg	62	124	23	90	113	706	236	384	224	9	45	42	1646
E8.1.4	A	pgp	46	64	5	54	59	593	202	296	186	9	32	34	1352
E8.1.5	B	none	84/44	221	117	-	117	749	1962	545	-	-	-	-	3256
E8.1.6	B	local	30/44	42	24	-	24	504	175	236	193	15	20	30	1173
E8.1.7	B	pgg	30/49	100	57	204	261	591	195	294	252	15	73	74	1494
E8.1.8	B	pgp	25/34	37	5	54	59	411	144	202	154	14	35	46	1006

Abbreviations

En: The problem solving environment
Mec: Which of the new planning mechanisms are used: none, local (no exchange of plans), pgg (exchange, no reordering), pgp (all)
Stime: The simulated time (number of KSIs invoked) to find solution(s); If two solutions, earliest time for each is given, in form *time for better / time for worse*
Rt: The total runtime (computation time) to find solution(s) (in minutes).
H-r: The total number of hypotheses messages received and incorporated by nodes.
N-r: The total number of node-plans messages received and incorporated by nodes.
T-r: The total number of messages received and incorporated by nodes.
Hyp: The number of hypotheses formed by nodes.
Goal: The number of goals formed and processed by nodes.
KSI: The number of KSIs formed by nodes.
Cl: The number of clusters in nodes' clustering-hierarchies.
Pl: The number of plans formed by nodes.
NP: The number of node-plans in nodes' network-models.
PGP: The number of PGPs formed by nodes.
Store: The total number of structures stored (storage costs).

control decisions are based on local views and the need for node 1 to provide predictive information to node 2 is not recognized. Node 2 receives predictive information only after it has already spent a lot of time on its ambiguous data, and as a result, the nodes generate and exchange many partial results before they converge on a solution. Not only does the network take longer (more KSs) to generate the solution, but the actual runtime, the amount of communication, and the amount of storage also indicate that the costs of finding the solution are very high.

When local planning capabilities are added to nodes, the network's ability to form solutions improves: although they as yet do not coordinate their activities with each other, the nodes are individually better problem solvers and so can

form their local partial solutions more efficiently. In the experiment (E8.1.2), the network finds the solution more than twice as quickly (in simulated time) as it did without local planning because local choices of KSs are made more judiciously. Moreover, the actual runtime in this experiment was dramatically lowered when local planning was introduced—the savings in problem solving (fewer KSs executed and less goal processing) much more than outweigh the overhead costs of planning (including forming and maintaining the clustering hierarchy). Local planning also reduces communication, because nodes form fewer partial solutions that could be exchanged.[5] The local planning mechanisms reduce the number of hypotheses, goals, and KSIs, and the storage savings more than offset the additional storage needs for clusters, plans, node-plans, and PGPs.[6] The costs of local planning (overhead in computation and storage) are more than justified by their benefits in improving network problem solving and in reducing overall computation, communication, and storage.

When nodes can exchange node-plans to recognize partial global goals, then they can pursue plans that work toward the best partial global goals. In experiment E8.1.3, nodes have the ability to do just this. As a result, the nodes (and in particular node 1 which locally prefers a plan that does *not* contribute to the best PGG) pursue better plans and the network finds the solution in less simulated time (fewer KSs are needed). However, the overhead for this improvement is very steep. The computation overhead of forming PGGs makes the overall runtime greater than what was needed when only local planning is used, even though local planning ran more KSs. The amount of communication is also dramatically increased by the exchange of node-plans. To a much lesser extent, communication is increased by the exchange of more hypotheses: without the more global view, the nodes only send the end result of their local plans; with the more global view, the nodes send several short partial tracks to each other for integration because since they cannot reorder activities to better divide the data so they each build one larger partial result. Overall storage needs are slightly reduced despite the fact

[5]Since the planning mechanisms lead the node to extend tracks by adding new data one step at a time, the node typically starts at one place and works outward from there. Without the planning mechanisms, a node often starts in several places and builds many more short tracks which it eventually merges into longer tracks.

[6]Local planning was caused by having nodes not send any node-plans and PGPs, but they still form these from their local plans. The fact that there are more node-plans and PGPs than local plans is due to idle node-plans and the PGPs they trigger.

that nodes maintain node-plans for each other and more PGPs, because fewer hypotheses and goals are formed. Thus, the ability to form PGGs introduces significant amounts of computation and communication overhead, while reducing storage and improving control decisions. The decision as to whether this level of partial global planning is worthwhile depends on the relative importance of these benefits and costs in a given application.

Finally, when all of the new mechanisms are added (E8.1.4), the control decisions improve still further, so even fewer KSs need to be run. In particular, nodes coordinate their activities to more quickly provide predictive results (from node 1 to 2) and to avoid redundant integration of results. Computationally, the overhead of PGPlanning is justified by this improvement in problem solving, since the runtime is less than in any other combination of mechanisms. Nodes also make much more intelligent communication decisions about hypotheses than before; however, the communication of node-plans adds overhead so that, on the whole, the nodes communicate more than if they had just planned locally (E8.1.2) or had not planned at all (E8.1.1). The overall storage needs of nodes are reduced despite the need to store node-plans and PGPs, thanks to the reduction in hypotheses and goals. Thus, with all of the PGPlanning and local planning mechanisms, the network is substantially more coordinated, uses less computation, and uses less storage, but the cost of these improvements is in the additional communication to provide nodes with the information they need to better control their actions and interactions.

Turning to environment B, we see the same basic trends when the mechanisms are added to the nodes (E8.1.5–E8.1.8), although their are some differences caused by characteristics of the environment. In particular, note that this environment contains much less ambiguity, since each of the data points is involved in one or the other of the solutions. The experiment summaries in Table 7 give the simulated time when each of these solutions is found, with the time for the more highly-rated solution (d_1–d_8) given first. Note that without the planner, the nodes form the poorer solution first because they initially focused on its strongly sensed data. Also, nodes exchange very many hypotheses in this environment because it is often the case that several nodes could potentially extend a given hypothesis (for example, d_1–d_3 formed by node 3 could be extended by any of the nodes).

As in environment A, local planning substantially reduces the simulated time (number of KSs run), and especially lets the nodes form the overall better solution earlier thanks to the ability of each to roughly predict beliefs of future results (E8.1.6). As a consequence, the solution time for forming both solutions is nearly half that of the network without any planning. Despite the computation overhead for planning, overall computation needs are significantly reduced. So is the communication in the network, since nodes form and exchange fewer short partial solutions. Finally, storage is also dramatically reduced.

With the additional ability to exchange node-plans, network performance actually degrades (E8.1.7). Not only do PGGs not help local decisions (since nodes do not need to be directed to plans that they should pursue), but they lead nodes into performing redundant tasks on the better PGG. For example, node 2 forms d'_7–d'_8 later than with only local planning because it is part of the more lowly-rated PGG, and so, node 2 exhausts its activities in d_4–d_8 first. The computation overhead is increased because of the additional redundancy and because of the costs of correlating node-plans into PGGs. The communication overhead is increased because of the exchange of node-plans and because the poorly coordinated PGPs (since nodes' activities are not reordered) cause nodes to form and exchange more short partial solutions. For example, in this environment the exchange of node-plans let the nodes recognize that they should prefer to work on data for d_1–d_8. Because they cannot reorder activities, node's 1, 2, and 4 all redundantly work on d_4 and d_5 first, and each sends d_4–d_5 to the others. Storage costs are also increased, so this level of partial global planning is not justified in any way for this case. In short, there is no advantage to forming PGGs without using all the PGPlanning mechanisms in this environment.

When nodes have all of the PGPlanning mechanisms (E8.1.8), the control decisions are improved: the PGPlanning coordinates nodes so that they avoid redundant work in their overlapping areas, and so that they avoid redundant integration of results. Simulated runtime to find both solutions is thus reduced and nodes still find the better solution earlier. The overall computation and the storage needs are also reduced, even with the added overhead of forming and saving PGPlanning information. However, although PGPlanning once again makes the exchange of hypotheses more selective, the communication of node-plans increases overall communication resource use.

In summary then, we can draw the following conclusions. The addition of a local planner to the nodes lets them as a group find solutions quicker not because they are a more coordinated network (team) but because each is a better problem solver (player). Giving nodes the ability to exchange information to recognize PGGs is only useful when some nodes would otherwise not pursue important plans in a timely manner—when they would otherwise focus on activities that look attractive locally but do not fit into more global goals—and even then the utility of finding PGGs depends on the available communication and computation resources. PGPlanning improves coordination, and can both improve network performance (reduce solution time) and decrease computation and storage resource needs. It does, however, add to communication resource use. Communication is inevitable when nodes must dynamically coordinate for specific situations; in situations that demand dynamic control of coordination to achieve better network performance, the additional communication overhead may well be worth it.[7] The local planning and PGPlanning mechanisms thus have costs as well as benefits, and the decision about which mechanisms to employ in a given time must consider these factors. In Chapter 9, we address issues in selective application of the mechanisms.

8.1.2 Experiment Set 8.2: Different Organizations and Their Overhead

Broadcasting coordination information and individually forming PGPs introduces considerable overhead into the nodes' activities. Altering the meta-level organization to change nodes' coordination responsibilities can reduce this overhead, but usually at some cost (such as reliability and node autonomy). In this experiment set, we analyze several static meta-level organizations to determine their costs and benefits.

Once again, the experiments are based on environments A and B (Figure 84).

[7] Recall that the initial approach that we have taken has nodes only consider how new information (hypotheses, node-plans, PGPs) can affect their own plans, so that others must be notified of changes. If the nodes were willing to perform more computations to infer how others' plans may change as well, then they would not have to communicate as much. Moreover, in the current implementation, control decisions about exchanging node-plans and PGPs are based on fairly simple criteria. Our future research directions therefore include exploring issues in the balance between computation and communication, and improving the mechanisms to be more selective about what information is exchanged.

For each environment, we explore four meta-level organizations. The first is the broadcast organization used in experiment set 8.1, where nodes broadcast their node-plans and individually form PGPs. The second is a centralized organization, where nodes send node-plans to a coordinating node (the least busy node is used—in environment A this is node 4 while in environment B this is node 1), which in turn forms the PGPs and sends these back to the other nodes. Since communication is more focused, this organization can reduce communication overhead, and since only one node maintains a full network-model and generates PGPs, the computation and storage needs of the overall network can be decreased. However, centralization decreases reliability (since the network depends on the single coordinating node) and the coordinating node may become a bottleneck. In addition, the extra lag caused by communication delays (node-plans must get to the coordinator and then PGPs must come back) can slow down the rate at which nodes respond to new or changing situations.

The third meta-level organization allows each node to pass node-plans only to its clockwise neighbor, so we refer to this as a "ring" organization. Nodes pass on these node-plans, but the extra delay as messages get passed around the ring can cause nodes to have less consistent network-models than in the broadcast organization, and poorer coordination can result. Thus, this organization explores what happens when communication patterns are limited and when nodes may have more inconsistent views of each other. Finally, in the fourth organization, nodes still communicate in a ring but with two important differences: they pass PGPs, and the organization limits when they can send these PGPs through the use of a predicate.[8] In this "pgp-ring" organization, the predicate causes nodes to pass a single group of PGPs around, where each modifies the group before passing it on: node 1 initially send its PGPs to node 2, which then adds to these PGPs and sends them to node 4 (its clockwise neighbor), and so on. The point of this organization is to see how passing a single set of PGPs can reduce communication but can also slow down the rate at which coordination information

[8] We define a predicate that limits when a given node can send PGPs, and it is checked whenever the node determines that it has PGPs to send. The predicate in these experiments allows each node to send PGPs once every 8 time units, and the allowed sending times are offset for the different nodes to achieve the receive-modify-send behavior desired. In essence, the predicate is part of the meta-level organization, but to maintain flexibility during evolution of the mechanisms we did not define a particular attribute of the organization that contains this information. An important area for future research is to develop a more encompassing, less *ad hoc* representation for the meta-level organization.

becomes available to nodes.

The experimental results are summarized in Table 8. In environment A, the broadcast organization promotes the best control decisions, allowing the network to find the overall solution faster than in other organizations (E8.2.1). However, the overhead costs (computation, communication, and storage) are lower in a centralized network (E8.2.2) than in any other: only node 4 maintains a full network-model (reducing network storage costs); only node 4 generates multi-node PGPs (reducing network computation costs); and communication is more directed, node-plans to node 4 and PGPs from node 4 (reducing communication costs). The ring network finds the overall solution later than the broadcast but earlier than the centralized organization, despite the fact that node 1 sends predictive results to node 2 later than in the previous organizations. This is because the more asynchronous formation and modification of PGPs (because of different delays in getting node-plans from each other) causes the nodes to update their node-plans and PGPs more often, and since this triggers nodes to communicate more often about their node-plans, the nodes develop a more current view of network activity. As the results indicate, the cost of this additional exchange of coordination information is an increase in the amount of communication and a dramatic rise in the computation overhead of coordination (since nodes are updating their PGPs more often). Finally, in the pgp-ring organization, the delays in propagating PGPs substantially degrades coordination, and the additional problem solving adds to computation and storage overhead. However, the restriction in how often nodes pass coordination information (a single set of PGPs is circulated, where each node receives them, updates them based on its local plans, and passes them on to the next) significantly reduces the number of messages exchanged in the network.

To illustrate how the different organizations affect network problem solving, we summarize the concurrent problem solving activities of the nodes. In Figure 85, the network with environment A and a broadcast organization is shown. Note that initially (time 16) the nodes pursue their best local plans, but at time 18 they receive each other's node-plans and begin to coordinate: node 1 starts to work on d_8-d_9 (more globally relevant than d'_1-d'_5), and node 2 focuses on data beyond the overlapping area to avoid redundancy (d_{13}). Node 1's predictive result d_8-d_9 causes node 2 to divide its single plan to process all its data in d_{10}-d_{15} into

Table 8: Experiment Summary for Experiment Set 8.2.

Expt	Env	Org	STime	Rtime	H-r	M-r	T-r	Store
E8.2.1	A	broadcast	46	64	5	54	59	1352
E8.2.2	A	central	48	52	4	48	52	1331
E8.2.3	A	ring	47	77	8	58	66	1339
E8.2.4	A	pgp-ring	62	77	4	27	31	1447
E8.2.5	B	broadcast	25/34	37	5	54	59	1006
E8.2.6	B	central	26/35	32	7	49	56	985
E8.2.7	B	ring	27/38	41	8	59	67	1089
E8.2.8	B	pgp-ring	27/44	44	9	33	42	1063

Abbreviations

Env: The problem solving environment
Org: Which of the meta-level organizations are used: broadcast, centralized, ring, or pgp-ring
Stime: The simulated time to find solution(s); if more than one, earliest time for each is given (better-sol/worse-sol).
Rtime: The total runtime (computation time) to find solution(s) (in minutes).
H-r: The total number of hypothesis messages received and incorporated by nodes.
M-r: The total number of meta-level messages (node-plan and PGP) received and incorporated by nodes.
T-r: The total number of messages received and incorporated by nodes.
Store: The total number of structures stored (storage costs).

two plans: one to form a track that can be combined with d_8–d_9 (of the same vehicle type) and the other to form tracks out of the remaining data. Since only the first of these plans fits into the larger PGP, it is pursued so node 2 concentrates on relevant data in d_{14} and d_{15} to combine with the results it already formed in d_{13}. Meanwhile, nodes 1 and 3 continue forming their partial tracks. Since the predicted and actual times of interactions are somewhat different (recall that the time-cushion is 1 so nodes can deviate slightly), node 2 believes that nodes 1 and 3 are further along than they actually are, and that is why it sends d_{13}–d_{15} rather than waiting until it forms a larger result. In addition, node 4 was initially intending to do the integration, so it receives this result. When node 3 completes its plan, and when node 1 modifies its plans because of received information, the PGP changes so that node 1 is seen as the node that could integrate the results soonest. The lack of the best possible coordination (because of the time-cushion) causes node 1 to attempt to duplicate the efforts of nodes 3 and 2, and node 2 recognizes too late (time 44) that it should send d_{12}–d_{15} to node 1. Node 1 nonetheless accumulates the results and builds the overall solution.

With the centralized organization (Figure 86), the node-plans sent at time 16 get to node 4 at time 18, and the PGPs arrive back at the nodes at time 20, which is when they begin to coordinate better (recall that in the broadcast organization they began to coordinate better at time 18). Once again, node 4 initially determines that it should integrate results, and this time it does this integration. Although its network-model is not quite current (for example, it believes that node 1 will receive d_1–d_6 sooner than it actually does), it does not learn about deviations from node-plans until after it has already integrated results—it identifies after the fact that the nodes could have coordinated better if node 1 had done the integration. With the ring organization (Figure 87), the nodes are initially much less coordinated, so the predictive results from node 1 to node 2 are not received until time 35. However, the different delays in propagating node-plans cause the nodes to update their network-models and change their local node-plans more often. As a result, the nodes expend more energy (computation and communication) coordinating, and can thus still make intelligent control decisions despite the delays in communication. However, note that the inconsistent views of the nodes do cause them to have different expectations about integrating results, so both node 1 and node 4 redundantly generate the

overall solution. Finally, with the pgp-ring organization (Figure 88), the delays in propagating PGPs means that node 1 does not get a more global view until after it has made significant progress on its locally superior (but globally inferior) plan to form d'_1–d'_5. Because this plan's rating increases as it becomes more complete (the fraction-complete rating factor increases, as discussed in Chapter 4), its PGP has a higher rating than the PGP to form d_1–d_{15} when node 1 receives the PGPs from node 3. Node 1 thus forms d'_1–d'_5 first, before it begins cooperating with the other nodes at time 36. Once it does cooperate, however, it knows to form and send predictive results to node 2 first, and thus the nodes coordinate reasonably from that point on.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-16		sensors			sensors			sensors			sensors	
16-17		d_1'			d_{10}			d_1				
17-18		d_1'			d_{10}			d_1				
18-19		d_9			d_{13}			d_1				
19-20		d_9			d_{13}			d_2				
20-21		d_9			d_{13}			d_2				
21-22		d_8			d_{13}			d_2				
22-23		d_8			d_{13}			d_1-d_2				
23-24		d_8			d_{13}			d_3				
24-25		d_8-d_9	d_8-d_9		d_{13}			d_3				
25-26		d_7			d_{13}			d_3				
26-27		d_7			d_{13}			d_1-d_3				
27-28		d_7		d_8-d_9	d_{14}			d_4				
28-29		d_7-d_9			d_{14}			d_4				
29-30		d_{10}			d_{14}			d_4				
30-31		d_{10}			d_{13}-d_{14}			d_1-d_4				
31-32		d_{10}			d_{15}			d_5				
32-33		d_7-d_{10}			d_{15}			d_5				
33-34		d_{11}			d_{15}			d_5				
34-35		d_{11}			d_{13}-d_{15}	d_{13}-d_{15}		d_1-d_5				
35-36		d_{11}			d_{12}			d_6				
36-37		d_7-d_{11}			d_{12}			d_6				
37-38		d_{12}			d_{12}			d_6			d_{13}-d_{15}	d_{13}-d_{15}
38-39		d_{12}			d_{12}-d_{15}			d_1-d_6	d_1-d_6			
39-40	d_{13}-d_{15}	d_6			d_{11}			d_1^2				
40-41		d_6			d_{11}			d_2^2				
41-42	d_1-d_6	d_1-d_{11}			d_{11}			d_1^2-d_2^2				
42-43		d_{12}			d_{11}-d_{15}			d_3^2				
43-44		d_1-d_{12}			d_{10}^2	d_{12}-d_{15}		d_1^2-d_3^2				
44-45		d_1-d_{15}			d_{10}^2			d_4^2				
45-46	d_{12}-d_{15}	*d_1-d_{15}*			d_{10}^2			d_1^2-d_4^2				

The concurrent activities of nodes over the time intervals are shown. For a given node and time interval, the node receives any data indicated at the beginning of the interval, performs local activity over the interval, and sends any data indicated at the end of the interval. Data being sent, received, or worked on locally is indicated as d_i. A prime (d_i') indicates work on another track (particularly in environments with two solutions). A superscript 2 (d_i^2) represents forming results for the data with an inferior vehicle type (for vehicle event-classes without much support, so results are lowly rated). Finally, asterisks surrounding data (*d_i-d_j*) indicates that the data is a solution.

Figure 85: Experiment E8.2.1--Environment A, Broadcast Organization.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-16		sensors			sensors			sensors			sensors	
16-17		d'_1			d_{10}			d_1				
17-18		d'_1			d_{10}			d_1				
18-19		d'_1			d_{10}			d_1				
19-20		d'_2			d_{10}			d_2				
20-21		d_9			d_{13}			d_2				
21-22		d_9			d_{13}			d_2				
22-23		d_9			d_{13}			d_1-d_2				
23-24		d_8			d_{13}			d_3				
24-25		d_8			d_{13}			d_3				
25-26		d_8			d_{13}			d_3				
26-27		d_8-d_9	d_8-d_9		d_{13}			d_1-d_3				
27-28		d_7			d_{13}			d_4				
28-29		d_7			d_{13}			d_4				
29-30		d_7		d_8-d_9	d_{14}			d_4				
30-31		d_7-d_9			d_{14}			d_1-d_4				
31-32		d_{10}			d_{14}			d_5				
32-33		d_{10}			d_{13}-d_{14}			d_5				
33-34		d_{10}			d_{15}			d_5				
34-35		d_7-d_{10}			d_{15}			d_1-d_5				
35-36		d_{11}			d_{15}			d_6				
36-37		d_{11}			d_{13}-d_{15}			d_6				
37-38		d_{11}			d_{12}			d_6				
38-39		d_7-d_{11}			d_{12}			d_1-d_6	d_1-d_6			
39-40		d_{12}			d_{12}			d_1^2				
40-41		d_{12}			d_{12}-d_{15}	d_{12}-d_{15}		d_2^2				
41-42	d_1-d_6	d_{11}			d_{11}			d_1^2-d_2^2				
42-43		d_7-d_{12}			d_{10}^2			d_3^2				
43-44		d_1-d_{11}	d_1-d_{11}		d_{10}^2			d_1^2-d_3^2			d_{12}-d_{15}	d_{12}-d_{15}
44-45		d'_2			d_{10}^2			d_4^2				
45-46		d'_2			d_{10}^2			d_1^2-d_4^2				
46-47		d'_1-d'_2			d_{10}^2			d_5^2			d_1-d_{11}	d_1-d_{15}
47-48		d'_3			d_{10}^2			d_1^2-d_5^2				*d_1-d_{15}*

See Figure 85 for explanation.

Figure 86: Experiment E8.2.2—Environment A, Centralized Organization.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-16		sensors			sensors			sensors			sensors	
16-17		d'_1			d_{10}			d_1				
17-18		d'_1			d_{10}			d_1				
18-19		d_7			d_{13}			d_1				
19-20		d_7			d_{13}			d_2				
20-21		d_7			d_{13}			d_2				
21-22		d_8			d_{13}			d_2				
22-23		d_9			d_{13}			d_1-d_2				
23-24		d_9			d_{13}			d_3				
24-25		d_9			d_{13}			d_3				
25-26		d_8			d_{13}			d_3				
26-27		d_8			d_{13}			d_1-d_3				
27-28		d_8-d_9			d_{13}			d_4				
28-29		d_7-d_9			d_{13}			d_4				
29-30		d_{10}			d_{13}			d_4				
30-31		d_{10}			d_{13}			d_1-d_4	d_1-d_3			
31-32		d_{10}			d_{13}			d_5				
32-33		d_7-d_{10}	d_7-d_{10}		d_{13}			d_5				
33-34	d_1-d_3	d_{11}			d_{13}			d_5				
34-35		d_{11}			d_{13}			d_1-d_5				
35-36		d_{11}		d_7-d_{10}	d_{14}			d_6				
36-37		d_7-d_{11}			d_{14}			d_6				
37-38		d_{12}			d_{14}			d_6				
38-39		d_{12}			d_{13}-d_{14}			d_1-d_6	d_1-d_6			
39-40		d_{12}			d_{15}			d_1^2				
40-41		d_7-d_{12}			d_{15}			d_2^2				
41-42	d_1-d_6	d_1-d_{12}	d_1-d_{12}		d_{15}			d_1^2-d_2^2				
42-43		d_1-d_{12}			d_{13}-d_{15}	d_{13}-d_{15}		d_3^2				
43-44		d'_1			d_{12}			d_4^2	d_1-d_5			
44-45		d_{11}^2			d_{10}^2		d_1-d_{12}	d_1^2-d_3		d_1-d_{12}		
45-46	d_{13}-d_{15}	d_1-d_{15}			d_{10}^2			d_3^2-d_4^2		d_{13}-d_{15}	d_1-d_{15}	
46-47		*d_1-d_{15}*			d_{10}^2			d_1^2-d_4^2			*d_1-d_{15}*	

See Figure 85 for explanation.

Figure 87: Experiment E8.2.3 Environment A, Ring Organization.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-16		sensors			sensors			sensors			sensors	
16-17		d'_1			d_{10}			d_1				
17-18		d'_1			d_{10}			d_1				
18-19		d'_1			d_{13}			d_1				
19-20		d'_2			d_{13}			d_2				
20-21		d'_2			d_{13}			d_2				
21-22		d'_2			d_{13}			d_2				
22-23		d'_1-d'_2			d_{13}			d_1-d_2				
23-24		d'_3			d_{13}			d_3				
24-25		d'_3			d_{13}			d_3				
25-26		d'_3			d_{13}			d_3				
26-27		d'_1-d'_3			d_{13}			d_1-d_3				
27-28		d'_4			d_{13}			d_4				
28-29		d'_4			d_{13}			d_4				
29-30		d'_4			d_{13}			d_4				
30-31		d'_1-d'_4			d_{13}			d_1-d_4				
31-32		d'_5			d_{13}			d_5				
32-33		d'_5			d_{13}			d_5				
33-34		d'_5			d_{13}			d_5				
34-35		d'_1-d'_5			d_{13}			d_1-d_5				
35-36		d'_1-d'_5			d_{13}			d_6				
36-37		d_9			d_{13}			d_6				
37-38		d_9			d_{13}			d_6				
38-39		d_9			d_{14}			d_1-d_6	d_1-d_6			
39-40		d_8			d_{14}			d_1^2				
40-41		d_8			d_{14}			d_2^2				
41-42	d_1-d_6	d_8			d_{13}-d_{14}			d_1^2-d_2^2				
42-43		d_8-d_9	d_8-d_9		d_{14}			d_3^2				
43-44		d_7			d_{14}			d_1^2-d_3^2				
44-45		d_7			d_{13}-d_{14}			d_4^2				
45-46		d_7		d_8-d_9	d_{15}			d_1^2-d_4^2				
46-47		d_7-d_9			d_{15}			d_5^2				
47-48		d_{10}			d_{15}			d_1^2-d_5^2				
48-49		d_{10}			d_{13}-d_{15}			d_6^2				
49-50		d_{10}			d_{12}			d_1^2-d_6^2	d_1^2-d_6^2			
50-51		d_7-d_{10}			d_{12}			d_1^2-d_6^2				
51-52		d_{11}			d_{12}							
52-53	d_1^2-d_6^2	d_1-d_{10}			d_{12}-d_{15}							
53-54		d_{11}			d_{11}							
54-55		d_{11}			d_{11}							
55-56		d_1-d_{11}	d_1-d_{10}		d_{11}							
56-57		d_{12}			d_{11}-d_{15}	d_{11}-d_{15}						
57-58		d_{12}			d_{10}							
58-59		d'_1			d_{10}-d_{15}							
59-60		d'_2		d_1-d_{10}	d_8-d_{15}							
60-61		d'_1-d'_2			d_1-d_{15}							
61-62		d_7^2			*d_1-d_{15}*							

See Figure 85 for explanation.

Figure 88: Experiment E8.2.4—Environment A, PGP-Ring Organization.

In environment B, the same types of issues arise. The broadcast organization promotes the best control decisions and the solutions are found in the least amount of (simulated) time, but the centralized organization incurs less overhead (computation, communication, and storage). The ring organization once again performs fairly poorly relative to the others. Finally, as in environment A, and the pgp-ring organization has inferior coordination and incurs higher computation and storage costs because of the additional problem solving, but the communication overhead is reduced because nodes send coordination information less often.

Looking more closely at the network activity in these experiments, we start with the broadcast organization (Figure 89). Initially at time 9, the nodes pursue their best local plans, but at time 11 they form and pursue larger PGPs. For the track d_1–d_8, they recognize that node 1 should form d_4–d_5, node 2 should form d_6–d_8, and node 3 should form d_1–d_3. Since node 2 gets a later start (it was initially doing redundant work on d_4), the nodes determine that node 3 integrates its result d_1–d_3 with node 1's result d_4–d_5, and sends this to node 2 for integration with its result d_6–d_8, and this is what occurs at time 24. However, some uncoordinated activity occurs at time 18 when node 3 changes its plans when it receives d_4–d_5, because it updates its predictions and believes that it will lag behind node 2 and so node 2 should integrate with node 1's results. However, when node 2 receives the updated node-plan from node 3 at time 20 and changes its plans to do the integration, it updates its own predictions and discovers that it also is taking longer than expected and that the initial plan to have node 3 generate d_1–d_5 was correct. The inconsistent views are thus eventually resolved and the overall solution is found. Meanwhile, node 4 initially could also participate in this PGP to form d_4–d_5, but since node 1 is already forming this result, and since node 4 has another PGP it could pursue while node 1 does not, node 4 turns to this other PGP and lets node 1 form d_4–d_5 (see Chapter 7, 7.3.4 for how node 4 decides this). Node 4 begins building d'_3–d'_6 while the other nodes are busy on the other PGP. They eventually complete their parts of the other PGP and begin cooperating with node 4: first, node 3 forms d'_1–d'_2 at time 27 which it combines at time 28 with d'_3–d'_6 received from node 4. It passes the combination on to node 2 which, since it generated the overall solution for the other PGP, is lagging behind in forming d'_7–d'_8. It does form this result at time 31, however, and at time

33 generates the overall solution for this PGP as well, completing this action at time 34.

In the centralized organization (Figure 90), the extra delay in coordinating caused by delays to and from node 1 both hurt and help network problem solving. They hurt network problem solving because the nodes get a later start working on data that is not redundant (nodes 2 and 4 work longer in d_4 and d_5). However, the delay helps because now node 2 is lagging sufficiently behind so that the indecision about who should integrate the overall solution that occurred in the broadcast organization does not happen here. Thus, although node 2 is somewhat more of a bottleneck in this case, this allows nodes to avoid lapses in coordination and find overall solutions almost as quickly as in the broadcast organization. In the ring organization (Figure 91), the delays cause node 4 to duplicate the work of node 1 and to delay pursuit of the other PGP. However, the delays do allow the nodes to avoid the lapse of coordination seen in the broadcast organization, because before they hear about changed plans they have already sent partial solutions to node 1 for integration.[9] Finally, in the pgp-ring organization (Figure 92), the delays in propagating PGPs cause inconsistencies that result in both node 2 and node 3 redundantly forming d_1–d_8. Inconsistent views degrade coordination in forming the other solution as well, since nodes differ in how they view solution construction and thus do not exchange partial results in the most timely manner.

In summary, this experiment set illustrated only a few of the styles in which nodes can cooperate in the partial global planning framework based on the meta-level organization. Many other organizations are also possible, particularly when there are more nodes in the network (see Section 8.1.6). The choice of which meta-level organization is best in a given situation depends on the desired network characteristics: broadcast can result in the best control decisions but at high overhead; centralized reduces the overhead but also the responsiveness and reliability suffer; given that communication channels between nodes have an underlying ring topology, a ring organization may be acceptable; given the need to reduce communication, the pgp-ring may be a suitable choice. There is no "best" meta-level organization, and the PGPlanning approach is thus geared to

[9]Node 1 is chosen because the partial solutions (d_1–d_5 and d_6–d_8) are formed at almost the same time. If integrated at node 2, the solution would be formed only 1 time unit earlier than if integrated at node 1. Since the difference is acceptable (with the time-cushion of 1) and since node 2 has other PGPs to work on, node 1 does the integration.

providing flexibility in how nodes coordinate.

These experiments also show the dynamic aspects of coordination, such as how predictive information causes nodes to change plans, and how nodes' inaccurate predictions about when interactions will take place may cause them to redevelop PGPs based on more up-to-date predictions. Therefore, even though the nodes essentially began problem solving with all the data, they are uncertain about how and when problem solving activities will take place. As time goes by, they change their plans and must dynamically recompute PGPs. The partial global planning mechanisms thus provide a means for nodes to respond to dynamic situations by exchanging the information they need (in a way dictated by the meta-level organization) to improve their PGPs.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-9		sensors			sensors			sensors			sensors	
9-10		d_4			d_4			d_1			d_4	
10-11		d_4			d_4			d_1			d_4	
11-12		d_4			d_6			d_1			d'_3	
12-13		d_5			d_6			d_2			d'_3	
13-14		d_5			d_6			d_2			d'_3	
14-15		d_5			d_7			d_2			d'_4	
15-16		d_4-d_5	d_4-d_5		d_7			d_1-d_2			d'_4	
16-17		d_4^2			d_7			d_3			d'_4	
17-18		d_5^2			d_6-d_7			d_3			d'_3-d'_4	
18-19		d_4^2-d_5^2			d_8		d_4-d_5	d_3			d'_5	
19-20		d_4^2-d_5^2			d_8			d_1-d_3			d'_5	
20-21				d_4-d_5	d_8			d_1-d_5	d_1-d_5		d'_5	
21-22					d_6-d_8	d_6-d_8		d'_1			d'_3-d'_5	
22-23					d_4-d_8			d'_1			d'_6	
23-24				d_1-d_5	d_1-d_8			d'_1			d'_6	
24-25					*d_1-d_8*			d'_2			d'_6	
25-26					d'_7			d'_2			d'_3-d'_6	d'_3-d'_6
26-27					d'_7			d'_2			d_4^2	
27-28					d'_7			d'_1-d'_2	d'_1-d'_2		d_5^2	
28-29					d'_8		d'_3-d'_6	d'_1-d'_6	d'_1-d'_6		d_5^2	
29-30					d'_8			d'_1-d'_6			d_4^2-d_5^2	
30-31					d'_8			$d_3^{2'}$			d'_3-d'_6	
31-32				d'_1-d'_6	d'_7-d'_8			$d_3^{2'}$			$d_3^{2'}$	
32-33					d'_1-d'_8			$d_4^{2'}$			$d_4^{2'}$	
33-34					*d'_1-d'_8*			$d_4^{2'}$			$d_3^{2'}$-$d_4^{2'}$	

See Figure 85 for explanation.

Figure 89: Experiment E8.2.5--Environment B, Broadcast Organization.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-9		sensors			sensors			sensors			sensors	
9-10		d_4			d_4			d_1			d_4	
10-11		d_4			d_4			d_1			d_4	
11-12		d_4			d_4			d_1			d_4	
12-13		d_5			d_5			d_2			d_5	
13-14		d_5			d_6			d_2			d'_3	
14-15		d_5			d_6			d_2			d'_3	
15-16		d_4-d_5	d_4-d_5		d_6			d_1-d_2			d'_3	
16-17		d_4^2			d_7			d_3			d'_4	
17-18		d_5^2			d_7			d_3			d'_4	
18-19		d_4^2-d_5^2			d_7		d_4-d_5	d_3			d'_4	
19-20		d_4^2-d_5^2			d_6-d_7			d_1-d_3			d'_3-d'_4	
20-21					d_8			d_1-d_5	d_1-d_5		d'_5	
21-22					d_8			d'_1			d'_5	
22-23					d_8			d'_1			d'_5	
23-24	d_1-d_5	d_1-d_5		d_1-d_5	d_6-d_8			d'_1			d'_3-d'_5	
24-25					d_1-d_8			d'_2	d_1-d_2		d'_6	
25-26					*d_1-d_8*			d'_2			d'_6	
26-27					d'_7			d'_2			d'_6	
27-28	d_1-d_2				d'_7			d'_1-d'_2			d'_3-d'_6	d'_3-d'_6
28-29					d'_7			d'_3			d_4^2	
29-30					d'_8			d'_3			d_5^2	
30-31					d'_8		d'_3-d'_6	d'_1-d'_6	d'_1-d'_6		d_5^2	
31-32					d'_8			$d_3^{2'}$			$d_4^{2'}$-$d_5^{2'}$	
32-33					d'_7-d'_8			d'_1-d'_6			$d_4^{2'}$	
33-34				d'_1-d'_6	d'_1-d'_8			$d_4^{2'}$			d'_3-d'_6	
34-35					*d'_1-d'_8*			$d_4^{2'}$			$d_3^{2'}$	

See Figure 85 for explanation.

Figure 90: Experiment E8.2.6—Environment B, Centralized Organization.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-9		sensors			sensors			sensors			sensors	
9-10		d_4			d_4			d_1			d_4	
10-11		d_4			d_4			d_1			d_4	
11-12		d_4			d_6			d_1			d_4	
12-13		d_5			d_6			d_2			d_5	
13-14		d_5			d_6			d_2			d_5	
14-15		d_5			d_7			d_2			d_5	
15-16		d_4-d_5	d_4-d_5		d_7			d_1-d_2			d_4-d_5	d_4-d_5
16-17		d_4^2			d_7			d_3			d_3'	
17-18		d_5^2			d_6-d_7			d_3			d_3'	
18-19		d_4^2-d_5^2			d_8		d_4-d_5	d_3			d_3'	
19-20		d_4^2-d_5^2			d_8			d_1-d_3	d_1-d_3		d_4'	
20-21					d_8			d_1-d_5			d_4'	
21-22					d_6-d_8	d_6-d_8		d_1-d_5			d_4'	
22-23	d_1-d_3	d_1-d_5		d_4-d_5	d_4-d_8	d_4-d_8		d_1'			d_3'-d_4'	
23-24		d_1-d_5			d_7'			d_1'			d_5'	
24-25					d_7'			d_1'			d_5'	
25-26	d_4-d_8	d_1-d_8			d_7'			d_2'			d_5'	
26-27		*d_1-d_8*			d_8'			d_2'			d_3'-d_5'	
27-28					d_8'			d_2'			d_6'	
28-29					d_8'			d_1'-d_2'	d_1'-d_2'		d_6'	
29-30					d_7'-d_8'			$d_4^{2'}$			d_6'	
30-31					d_6'			$d_4^{2'}$			d_3'-d_6'	
31-32					d_6'			$d_5^{2'}$		d_1'-d_2'	d_1'-d_6'	
32-33					d_6'			$d_5^{2'}$			d_1'-d_6'	
33-34					d_6'-d_8'	d_6'-d_8'		$d_4^{2'}$-$d_5^{2'}$			$d_4^{2'}$	
34-35					$d_6^{\prime 2}$			$d_3^{2'}$			$d_5^{2'}$	
35-36					$d_7^{2'}$			$d_3^{2'}$-$d_5^{2'}$			$d_4^{2'}$-$d_5^{2'}$	
36-37					$d_6^{2'}$-$d_7^{2'}$			$d_2^{2'}$		d_6'-d_8'	d_1'-d_8'	
37-38					$d_8^{2'}$			$d_2^{2'}$-$d_5^{2'}$			*d_1'-d_8'*	

See Figure 85 for explanation.

Figure 91: Experiment E8.2.7—Environment B, Ring Organization.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-9		sensors			sensors			sensors			sensors	
9-10		d_4			d_4			d_1			d_4	
10-11		d_4			d_4			d_1			d_4	
11-12		d_4			d_6			d_1			d_4	
12-13		d_5			d_6			d_2			d_5	
13-14		d_5			d_6			d_2			d'_3	
14-15		d_5			d_7			d_2			d'_3	
15-16		d_4-d_5	d_4-d_5		d_7			d_1-d_2			d'_3	
16-17		d_4-d_5			d_7			d_3			d'_4	
17-18		d_4^2			d_6-d_7			d_3			d'_4	
18-19		d_5^2			d_8		d_4-d_5	d_3			d'_4	
19-20		d_4^2-d_5^2			d_8			d_1-d_3			d'_3-d'_4	
20-21		d_4^2-d_5^2			d_8			d_4			d'_5	
21-22					d_6-d_8			d_4			d'_5	
22-23					d_4-d_8	d_4-d_8		d_4			d'_5	
23-24					d_4-d_8		d_4-d_5	d_1-d_5	d_1-d_5		d'_3-d'_5	
24-25					d'_7			d'_1			d'_6	
25-26					d'_7		d_4-d_8	d_1-d_8		d_4-d_8	d_4-d_8	
26-27				d_1-d_5	d_1-d_8			*d_1-d_8*			d'_6	
27-28					*d_1-d_8*			d'_2			d'_6	
28-29					d'_7			d'_1			d'_3-d'_6	
29-30					d'_8			d'_2		d_1-d_5	d_1-d_5	
30-31					d'_8			d'_2			$d_3^{2'}$	
31-32					d'_8			d'_2			$d_4^{2'}$	
32-33					d'_7-d'_8			d'_1-d'_2			$d_3^{2'}$-$d_4^{2'}$	
33-34					d'_6			d'_3			$d_5^{2'}$	
34-35					d'_6			$d_1^{2'}$			$d_3^{2'}$-d_5^2	
35-36					d'_6			d'_3			$d_6^{2'}$	
36-37					d'_6-d'_8			d'_3			$d_3^{2'}$-$d_6^{2'}$	d'_3-d'_6
37-38					d'_5			d'_1-d'_3		d_4-d_5	d'_3-d'_6	
38-39					d'_5			d'_4				
39-40				d'_3-d'_6	d'_3-d'_8	d'_3-d'_8		d'_4				
40-41					d'_7-d'_8			d'_4				
41-42					d'_7-d'_8			d'_1-d'_4				
42-43					d'_7-d'_8		d'_3-d'_8	d'_1-d'_8				
43-44								*d'_1-d'_8*				

See Figure 85 for explanation.

Figure 92: Experiment E8.2.8—Environment B, PGP-Ring Organization.

8.1.3 Experiment Set 8.3: Different Organizations in More Dynamic Environments

Issues in dynamic coordination are compounded when data arrives over time. In this experiment set, we briefly explore these issues. Our purpose is to determine first of all how well the PGPlanning mechanisms can adapt to unexpected changes caused by new data, and second of all the costs and benefits of using PGPlanning in such a domain. To do this, we once again turn to environments A and B, but this time we alter these environments so that all data does not arrive before problem solving begins. We instead have data arrive at intervals of three time units: data for sensed time 1 arrives at simulated time 1, data for sensed time 2 arrives at simulated time 4, sensed time 3 at simulated time 7, and so on. This arrival rate is slow enough so that nodes can perform some problem solving between arrivals, but fast enough so that control of actions and interactions are still important and nodes cannot exhaustively explore the data (see Chapter 5).

The experimental results are summarized in Table 9. For each environment, we run three experiments: without any planning (no local or PGPlanning); with the new mechanisms and a broadcast meta-level organization; and with the new mechanisms and a centralized meta-level organization.

The incorporation of data over time has an effect on problem solving without the mechanisms (E8.3.1). In particular, the computation time needed by the network is nearly twice that needed when the data arrives all at once (E8.1.1 in Table 7). This can be attributed to the later start that node 2 gets: since it does not begin getting data until time 28, node 1 has more time to develop tracks of both the correct and incorrect vehicle types, which it provides to node 2 which spends time working with each. Moreover, the extra hypotheses communicated (12 more than in E8.1.1) trigger additional problem solving and control activities. Once again, the broadcast organization provides better coordination than the centralized control, but is more costly in network computation and communication (recall, however, that some of the additional computation could occur in parallel).

Comparing the broadcast and centralized organizations, note that this time they are more different than when data came in at once (E8.2.1 and E8.2.2 in Table 8). In particular, the control decisions are substantially worse in the centralized case. To better understand this, we first illustrate the concurrent activities

Table 9: Experiment Summary for Experiment Set 8.3.

Expt	Env	Org	STime	Rtime	H-r	M-r	T-r	Store
E8.3.1	A	none	187	957	56	0	56	3847
E8.3.2	A	broadcast	55	93	6	151	157	1348
E8.3.3	A	central	68	70	4	109	113	1368
E8.3.4	B	none	71/58	109	57	0	57	2495
E8.3.5	B	broadcast	33/42	42	4	141	145	1067
E8.3.6	B	central	39/49	43	6	99	105	1108

Abbreviations

Env: The problem solving environment
Org: Which of the meta-level organizations are used: broadcast, centralized, or none (if none, then no planning at all)
Stime: The simulated time to find solution(s); if more than one, earliest time for each is given (better-sol/worse-sol).
Rtime: The total runtime (computation time) to find solution(s) (in minutes).
H-r: The total number of hypothesis messages received an incorporated by nodes.
M-r: The total number of meta-level messages (node-plans and PGPs) received and incorporated by nodes.
T-r: The total number of messages (hypothesis and meta-level) received and incorporated by nodes.
Store: The total number of structures stored (storage costs).

of nodes with the broadcast organization in Figures 93 and 94. Initially, node 1 works on the data for the undesirable track d'_1–d'_5, node 2 has no data, and node 3 works on the early data for the important track. However, node 1 switches to working on the important track as soon as it starts getting data at d_4 (time 10) because it has received a node-plan from node 3 covering data d_1–d_3 and can form a PGP to cooperate with node 3. Since it rates this PGP more highly (a combination of the participating node plans), node 1 pursues this data. Moreover, note how as more data comes in, node 1 prefers the later data to avoid duplication of effort with node 3. Node 1 has formed d_7–d_8 by the time it starts to get node-plans from node 2 (at time 30). At this point, node 1 recognizes that it should provide predictive information to node 2; it extends d_7–d_8 into d_9, and sends the result at time 36 (time interval 35-36). It continues to extend to later sensed data, and receives d_1–d_6 from node 3 at time 38. Meanwhile, node 2 uses the received data to pursue more relevant actions and forms d_{13}–d_{15} at time 51, which it sends to node 1 just when node 1 has formed d_1–d_{12}. Node 1 thus completes the overall solution at time 55.

In contrast, consider the concurrent activities of nodes when the centralized organization is used, as shown in Figures 95 and 96. Note that when node 1 gets data for d_4 at time 10, it does not begin to work with it because it cannot locally form the PGP to cooperate with node 3. Instead, node 4 forms this PGP and sends it back. In the mean time, however, node 1 has further progressed with the plan to form d_1'–d_5', and the plan's rating rises as more of it becomes completed (the fraction-complete increases, as in Chapter 4, Section 4.3.2). This increase is reflected in the corresponding node-plan and PGP ratings, and so, this PGP is more highly rated than the cooperative PGP that is received from node 4 at time 15. It is for this reason that node 1 gets a late start on forming predictive information for node 2. It gets the PGP specifying how it can assist node 2 after it already has all of its data, and thus forms d_8–d_9 at time 46 before extending it to other sensed times. Cooperation then proceeds much as in other organizations, including some minor coordination problems caused by actions taking longer than predicted: node 2 was supposed to have formed and sent d_{12}–d_{15}, and the delay caused node 1 to perform redundant and unimportant work while waiting.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-2		sens(1)			sens(1)			sens(1)			sens(1)	
2-3		d'_1						d_1				
3-4		d'_1						d_1				
4-5		sens(2)			sens(2)			sens(2)			sens(2)	
5-6		d'_1						d_1				
6-7		d'_2						d_2				
7-8		sens(3)			sens(3)			sens(3)			sens(3)	
8-9		d'_2						d_2				
9-10		d'_2						d_2				
10-11		sens(4)			sens(4)			sens(4)			sens(4)	
11-12		d_4						d_1-d_2				
12-13		d_4						d_3				
13-14		sens(5)			sens(5)			sens(5)			sens(5)	
14-15		d_4						d_3				
15-16		d_5						d_3				
16-17		sens(6)			sens(6)			sens(6)			sens(6)	
17-18		d_6						d_1-d_3				
18-19		d_6						d_4				
19-20		sens(7)			sens(7)			sens(7)			sens(7)	
20-21		d_7						d_4				
21-22		d_7						d_4				
22-23		sens(8)			sens(8)			sens(8)			sens(8)	
23-24		d_7						d_1-d_4				
24-25		d_8						d_5				
25-26		sens(9)			sens(9)			sens(9)			sens(9)	
26-27		d_8						d_5				
27-28		d_8						d_5				
28-29		sens(10)			sens(10)			sens(10)			sens(10)	
29-30		d_7-d_8			d_{10}			d_1-d_5				
30-31		d_9			d_{10}			d_6				
31-32		sens(11)			sens(11)			sens(11)			sens(11)	
32-33		d_9			d_{11}			d_6				
33-34		d_9			d_{11}			d_6				

Sens(i) indicates sensor data for sensed time i is incorporated. See Figure 85 for further explanation.

Figure 93: Experiment E8.3.2 (Part 1)—Environment A, Broadcast, Data Over Time.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
34-35		sens(12)			sens(12)			sens(12)			sens(12)	
35-36		d_7-d_9	d_7-d_9		d_{12}			d_1-d_6	d_1-d_6			
36-37		d_{12}			d_{12}			d_1^2				
37-38		sens(13)			sens(13)			sens(13)			sens(13)	
38-39	d_1-d_6	d_{10}	d_7-d_9		d_{13}			d_2^2				
39-40		d_{10}			d_{13}			d_1^2-d_2^2				
40-41		sens(14)			sens(14)			sens(14)			sens(14)	
41-42		d_1-d_9			d_{13}			d_3^2				
42-43		d_{10}			d_{14}			d_1^2-d_3^2				
43-44		sens(15)			sens(15)			sens(15)			sens(15)	
44-45		d_1-d_{10}			d_{14}		d_7-d_9	d_4^2	d_1^2-d_3^2			
45-46		d_{11}			d_{14}			d_1^2-d_4^2				
46-47		d_{11}			d_{13}-d_{14}			d_1-d_9	d_1-d_9			
47-48		d_{11}			d_{15}							
48-49		d_1-d_{11}			d_{15}							
49-50	d_1-d_9	d_{12}			d_{15}							
50-51		d_{12}			d_{13}-d_{15}	d_{13}-d_{15}						
51-52		d_1-d_{12}	d_1-d_{12}		d_{12}							
52-53		d_1-d_{12}			d_{12}							
53-54	d_{13}-d_{15}	d_1-d_{15}			d_{10}^2							
54-55		*d_1-d_{15}*		d_1-d_{12}	d_1-d_{15}							

Sens(i) indicates sensor data for sensed time i is incorporated. See Figure 85 for further explanation.

Figure 94: Experiment E8.3.2 (Part 2)—Environment A, Broadcast, Data Over Time.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-2		sens(1)			sens(1)			sens(1)			sens(1)	
2-3		d'_1						d_1				
3-4		d'_1						d_1				
4-5		sens(2)			sens(2)			sens(2)			sens(2)	
5-6		d'_1						d_1				
6-7		d'_2						d_2				
7-8		sens(3)			sens(3)			sens(3)			sens(3)	
8-9		d'_2						d_2				
9-10		d'_2						d_2				
10-11		sens(4)			sens(4)			sens(4)			sens(4)	
11-12		d'_1-d'_2						d_1-d_2				
12-13		d'_3						d_3				
13-14		sens(5)			sens(5)			sens(5)			sens(5)	
14-15		d'_3						d_3				
15-16		d'_3						d_3				
16-17		sens(6)			sens(6)			sens(6)			sens(6)	
17-18		d'_1-d'_3						d_1-d_3				
18-19		d'_4						d_4				
19-20		sens(7)			sens(7)			sens(7)			sens(7)	
20-21		d'_4						d_4				
21-22		d'_4						d_4				
22-23		sens(8)			sens(8)			sens(8)			sens(8)	
23-24		d'_1-d'_4						d_1-d_4				
24-25		d'_5						d_5				
25-26		sens(9)			sens(9)			sens(9)			sens(9)	
26-27		d'_5						d_5				
27-28		d'_5						d_5				
28-29		sens(10)			sens(10)			sens(10)			sens(10)	
29-30		d'_1-d'_5			d_{10}			d_1-d_5				
30-31		d'_1-d'_5			d_{10}			d_6				
31-32		sens(11)			sens(11)			sens(11)			sens(11)	
32-33		d_4			d_{10}			d_6				
33-34		d_4			d_{10}			d_6				

Sens(i) indicates sensor data for sensed time i is incorporated. See Figure 85 for explanation.

Figure 95: Experiment E8.3.3 (Part 1)---Environment A, Centralized, Data Over Time.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
34-35		sens(12)			sens(12)			sens(12)			sens(12)	
35-36		d_4			d_{10}			d_1-d_6	d_1-d_6			
36-37		d_5			d_{10}			d_1^2				
37-38		sens(13)			sens(13)			sens(13)			sens(13)	
38-39		d_9			d_{10}			d_2^2				
39-40		d_9			d_{10}			d_1^2-d_2^2				
40-41	d_1-d_6	sens(14)			sens(14)			sens(14)			sens(14)	
41-42		d_9			d_{10}			d_3^2				
42-43		d_8			d_{10}			d_1^2-d_3^2				
43-44		sens(15)			sens(15)			sens(15)			sens(15)	
44-45		d_8			d_{13}			d_4^2	d_1^2-d_3^2			
45-46		d_8			d_{13}			d_1^2-d_4^2				
46-47		d_8-d_9	d_8-d_9		d_{13}			d_5^2				
47-48		d_7			d_{13}			d_1^2-d_5^2				
48-49		d_7			d_{13}			d_6^2				
49-50		d_7		d_8-d_9	d_{13}			d_1^2-d_6^2	d_1^2-d_6^2			
50-51		d_7-d_9			d_{13}							
51-52		d_{10}			d_{13}							
52-53		d_{10}			d_{14}							
53-54	d_1^2-d_6^2	d_{10}			d_{14}							
54-55		d_1-d_9			d_{14}							
55-56		d_1-d_{10}			d_{13}-d_{14}							
56-57		d_{11}			d_{15}							
57-58		d_{11}			d_{15}							
58-59		d_{11}			d_{15}							
59-60		d_1-d_{11}			d_{13}-d_{15}							
60-61		d_{12}			d_{12}							
61-62		d_{12}			d_{12}							
62-63		d_{12}			d_{12}							
63-64		d_1-d_{12}			d_{12}-d_{15}	d_{12}-d_{15}						
64-65		d_1-d_{12}			d_{11}							
65-66		d_7^2			d_{11}							
66-67	d_{12}-d_{15}	d_1-d_{15}			d_{11}							
67-68		*d_1-d_{15}*			d_{10}^2							

Sens(i) indicates sensor data for sensed time i is incorporated. See Figure 85 for explanation.

Figure 96: Experiment E8.3.3 (Part 2)—Environment A, Centralized, Data Over Time.

Turning now to environment B, we first note that in this case the delays between data arrivals actually reduced the time needed to find both solutions (compare E8.3.4 with E8.1.5 in Table 7). The computation, communication, and storage were also reduced. Unlike environment A, there is little ambiguous data in this environment so delays do not trigger exploration of unpromising alternatives. In addition, receiving data over time improves local problem solving because nodes build fewer initial tracks (from the earlier data) and extend them (see Chapter 5), hence the improved performance and reduced cost. With the PG-Planning mechanisms, we once again see that the broadcast organization makes better control decisions than the centralized organization. This time, however, the computation needs of broadcast are also slightly less because the additional problem solving done by the centralized environment in this case offset the lower coordination overhead. The centralized organization does, however, reduce the amount of communication more substantially than in previous experiments.

To analyze the difference between how the broadcast and centralized organizations respond to data arrival over time in environment B, we once again observe the concurrent activities of nodes. The activities in this environment are more complex because of the presence of two solutions and because the data for certain sensed times of the poorer solution (ie. d'_1–d'_3) are strongly sensed and can initially distract nodes into preferring to work on this solution. In the broadcast environment (Figure 97), node 3 begins with this stronger data, as does node 4 when it starts receiving this data at time 7. As the weak data for this track arrives, however, the emphasis shifts to the better track (nodes 3 and 4 shift at time 14). Meanwhile, nodes 1 and 2 both initially get overlapping data in d_4 and d_5 (at times 10 and 13) which they each plan to pursue. Shortly thereafter, node 4 also plans to pursue this data (time 14). This potential duplication of effort causes node 1 to move to inferior vehicle types, node 2 to pursue data for sensed time 6, and node 4 to move back to activities for the less highly-rated solution (all at time 16). Only after more coordination does node 1 return to forming d_4–d_5 at time 20. Nodes 2 and 3 form their pieces of this track (node 3 much earlier (time 27) since its data was available earlier) and node 2 eventually forms the overall solution d_1–d_8 at time 32. Simultaneously, node 4 is forming d'_3–d'_6, although working from d_6 backward since initially it was trying to avoid duplicating work of node 3. Node 3 had already formed d'_1–d'_2 at time 11 before it changed to the

other track, so it simply sends this result to node 4 which integrates it with d'_3-d'_6 at time 32. The combination is sent to node 2 which eventually combines it with its local result (d'_7-d'_8) to form the overall solution (time interval 41-42).

The centralized organization (Figures 98 and 99) is much the same as the broadcast organization except for node 2's activities. Once again, nodes 3 and 4 focus on the strong data until sufficient weak data arrives for them to realize that the other track will likely have a higher belief. Node 3 then forms d_1-d_3 at time 27, integrates this with node 1's result at time 28, and sends the combination to node 2. Node 4, after some false starts (coordinating with nodes 1 and 2) works on d'_3-d'_6 as before and combines this with node 3's result at time 36 to form d'_1-d'_6 which it sends to node 2. Node 2 however is much more of a bottleneck than in the broadcast organization because of the delays in receiving coordination information. Initially, it and nodes 1 and 4 are all capable of working on d_4 and d_5, and the assignment of these tasks eventually rests with node 1. When it gets data at time 16 to extend its plan to work on d_4-d_5 into d_6, node 2 does not pursue it because the PGP indicates that this plan is covered by another. Only after enough time has passed does node 2 receive an updated PGP at time 23 allowing it to pursue this data. Thus, it gets a much later start on forming d_6-d_8 and the first solution is not formed until time 39. It also gets several contradictory PGPs so that it at times thinks that other nodes should do the integration; that is why it at time 32 sends d_1-d_6 to node 1, and node 1 does not incorporate this result because by the time it arrives the PGPs indicate once again that node 2 should do the integration.[10] Finally, node 2 forms d'_7-d'_8 and generates the other solution over time interval 48-49.

In summary, this experiment set has shown a number of things. Most importantly, the PGPlanning mechanisms promote cooperation in more dynamic environments where data arrives (and thus the perceived problems to solve change) over time. However, the additional dynamics do degrade coordination and introduce costs. The quality of plans and PGPs based on only some of the data may not be as high because the nodes have less context for identifying the best activities. The nodes also must exchange more node-plans and PGPs to keep each

[10]This type of oscillation cannot continue indefinitely because with each oscillation the nodes pursue a few more of their tasks so that eventually one or the other will integrate the data before it has a chance to oscillate back again.

other informed as their local plans change more frequently, raising communication overhead. In turn, a greater exchange of coordination information triggers more PGPlanning activity, so computation overhead increases as well. In these dynamically changing environments, more coordination decisions are based on obsolete information, and coordination may thus be impaired, as we have seen. Thus, coordination is more difficult, more costly, and may be less effective as the unpredictability of the environment increases. In the environments described here, however, the costs of coordination are outweighed by the improvements to control decisions and network problem solving.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-2		sens(1)			sens(1)			sens(1)			sens(1)	
2-3								d'_1				
3-4								d'_1				
4-5		sens(2)			sens(2)			sens(2)			sens(2)	
5-6								d'_1				
6-7								d'_2				
7-8		sens(3)			sens(3)			sens(3)			sens(3)	
8-9								d'_2			d'_3	
9-10								d'_2			d'_3	
10-11		sens(4)			sens(4)			sens(4)			sens(4)	
11-12		d_4			d_4			d'_1-d'_2			d'_3	
12-13		d_4			d_4			d'_4			d'_4	
13-14		sens(5)			sens(5)			sens(5)			sens(5)	
14-15		d_4			d_4			d_1			d_5	
15-16		d_5			d_5			d_1			d_5	
16-17		sens(6)			sens(6)			sens(6)			sens(6)	
17-18		d_4^2			d_6			d_1			d'_5	
18-19		d_5^2			d_6			d_2			d'_5	
19-20		sens(7)			sens(7)			sens(7)			sens(7)	
20-21		d_5			d_6			d_2			d'_5	
21-22		d_4-d_5			d_7			d_2			d'_6	
22-23		sens(8)			sens(8)			sens(8)			sens(8)	
23-24		d_4-d_5			d_7			d_1-d_2			d'_6	
24-25		d_5^2			d_7			d_3			d'_6	
25-26		d_1^2-d_5^2	d_4-d_5		d_6-d_7			d_3			d'_5-d'_6	
26-27		d_4^2-$d_5$2			d_8			d_3			d'_4	
27-28					d_8		d_4-d_5	d_1-d_3			d'_4	
28-29					d_8			d_1-d_5	d_1-d_5		d'_4-d'_6	
29-30					d_6-d_8	d'_1-d'_2		d_1-d_5			d'_3-d'_6	
30-31					d_5			d_3			d'_3-d'_6	
31-32				d_1-d_5	d_1-d_8			d_2^2			$d_3^{2'}$	
32-33					*d_1-d_8*			d_2^2-d_3^2		d'_1-d'_2	d'_1-d'_6	
33-34					d'_7			d_1^2			d'_1-d'_6	
34-35					d'_7			d_1^2-d_3^2			$d_4^{2'}$	d'_1-d'_6
35-36					d'_7			d_4^2			$d_3^{2'}$-$d_4^{2'}$	
36-37					d'_8			d_4^2			$d_5^{2'}$	
37-38				d'_1-d'_6	d'_8			d_1^2-d_4^2			$d_3^{2'}$-$d_5^{2'}$	
38-39					d'_8			d_5^2			$d_6^{2'}$	
39-40					d'_7-d'_8			d_5^2			$d_3^{2'}$-$d_6^{2'}$	
40-41					d'_1-d'_8			d_1^2-d_5^2			$d_3^{2'}$-$d_6^{2'}$	
41-42					*d'_1-d'_8*			d_1^2-d_5^2				

Sens(i) indicates sensor data for sensed time i is incorporated. See Figure 85 for explanation.

Figure 97: Experiment E8.3.5—Environment B, Broadcast, Data Over Time.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-2		sens(1)			sens(1)			sens(1)			sens(1)	
2-3								d'_1				
3-4								d'_1				
4-5		sens(2)			sens(2)			sens(2)			sens(2)	
5-6								d'_1				
6-7								d'_2				
7-8		sens(3)			sens(3)			sens(3)			sens(3)	
8-9								d'_2			d'_3	
9-10								d'_2			d'_3	
10-11		sens(4)			sens(4)			sens(4)			sens(4)	
11-12		d_4			d_4			d'_1-d'_2			d_4	
12-13		d_4			d_4			d'_3			d_4	
13-14		sens(5)			sens(5)			sens(5)			sens(5)	
14-15		d_4			d_4			d_1			d'_3	
15-16		d_5			d_5			d_1			d_4	
16-17		sens(6)			sens(6)			sens(6)			sens(6)	
17-18		d_4^2			d'_6			d_1			d_4	
18-19		d_5^2			d'_6			d_2			d_5	
19-20		sens(7)			sens(7)			sens(7)			sens(7)	
20-21		d_5			d'_6			d_2			d'_5	
21-22		d_4-d_5	d_4-d_5		d'_7			d_2			d'_5	
22-23		sens(8)			sens(8)			sens(8)			sens(8)	
23-24		d_4-d_5			d_6			d_1-d_2			d'_5	
24-25		d_5^2			d_6		d_4-d_5	d_3			d'_4	
25-26		d_4^2-d_5^2			d_6			d_3			d'_6	
26-27		d_4^2-$d_5$2			d_5			d_3			d'_6	
27-28					d_7			d_1-d_3			d'_6	
28-29					d_7			d_1-d_5	d_1-d_5		d'_4	
29-30					d_7			d_1-d_5			d'_4	
30-31					d_5			d'_3			d'_4-d'_5	
31-32				d_1-d_5	d_1-d_6			d_4^2		d_1-d_5	d'_5-d'_6	
32-33					d_6-d_7			d'_3			d'_4-d'_6	
33-34					d_1-d_7			d'_1-d'_3	d'_1-d'_2		d'_3-d'_6	d'_3-d'_6
34-35					d_8			d'_4			d_1-d_5	
35-36					d_8			d'_4			$d_3^{2'}$	
36-37					d_8			d'_4		d'_1-d'_2	d'_1-d'_6	d'_1-d'_6
37-38					d_1-d_8			d'_1-d'_4			d'_1-d'_6	
38-39					*d_1-d_8*			d_4^2			$d_4^{2'}$	

Sens(i) indicates sensor data for sensed time i is incorporated. See Figure 85 for explanation.

Figure 98: Experiment E8.3.6 (Part 1)—Environment B, Centralized, Data Over Time.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
39-40					d_7'			d_5^2			$d_3^{2'}$-$d_4^{2'}$	
40-41					d_7'			d_5^2				
41-42				d_1'-d_6'	d_8'			d_4^2-d_5^2				
42-43					d_8'			d_3^2				
43-44					d_8'			d_3^2-d_5^2				
44-45					d_6'-d_7			d_2^2				
45-46					d_7'-d_8'			d_2^2-d_5^2				
46-47					d_6'-d_8'			d_1^2				
47-48					d_1'-d_8'			d_1^2-d_5^2				
48-49					*d_1'-d_8'*			d_1^2-d_5^2				

Sens(i) indicates sensor data for sensed time i is incorporated. See Figure 85 for explanation.

Figure 99: Experiment E8.3.6 (Part 2)—Environment B, Centralized, Data Over Time.

8.1.4 Experiment Set 8.4: Predictability and Responsiveness

This experiment set studies the balance between predictability and responsiveness. Recall that this balance is struck by the value chosen for the time-cushion, which represents how much plans can deviate from what others have been told (through node-plans) before the changes must be told to others. In the previous experiments with PGPlanning, the time-cushion had a value of 1. By exploring how lower and higher values for the time-cushion affect the costs and benefits of PGPlanning, we can gain a fuller understanding of how predictability and responsiveness can be balanced.

We again focus on environments A and B, with the broadcast and centralized meta-level organizations. For each combination of environment and organization, we describe three experiments: with a time-cushion of 0, a time-cushion of 1 (the default), and a time-cushion of 2. The results are summarized in Table 10.

We begin with environment A using a broadcast organization (E8.4.1 - E8.4.3). As the time-cushion increases, several trends become apparent. First, the quality of coordination decreases somewhat because nodes are adapting less to deviations—they continue with PGPs that may not be the best they could form. Second, the computation overhead is substantially reduced, since nodes do not spend as much time recalculating how they should coordinate. Third, the communication overhead is also significantly reduced, since nodes update each other's network-models (by passing node-plans) less often. Fourth, the storage overhead slightly increases due to the extra problem solving caused by less precise coordination: the extra storage is attributable to more hypotheses, goals, and KSIs, while the coordination storage is essentially the same (since updated node-plans replace earlier versions). The same trends are seen with the centralized organization (E8.4.4 - E8.4.6).

Table 10: Experiment Summary for Experiment Set 8.4.

Expt	Env	Org	T-C	STime	Rtime	H-r	M-r	T-r	Store
E8.4.1	A	bc	0	43	76	5	63	68	1280
E8.4.2	A	bc	1	46	64	5	54	59	1352
E8.4.3	A	bc	2	47	57	4	42	46	1357
E8.4.4	A	cn	0	45	59	6	65	71	1306
E8.4.5	A	cn	1	48	52	4	48	52	1331
E8.4.6	A	cn	2	49	50	4	35	39	1347
E8.4.7	B	bc	0	25/34	45	6	95	101	1015
E8.4.8	B	bc	1	25/34	37	5	54	59	1006
E8.4.9	B	bc	2	26/39	39	7	63	70	1093
E8.4.10	B	cn	0	32/41	42	8	85	93	1057
E8.4.11	B	cn	1	26/35	32	7	49	56	985
E8.4.12	B	cn	2	32/47	39	4	41	45	1136

Abbreviations

Env: The problem solving environment
Org: Which of the meta-level organizations are used: bc = broadcast, cn = centralized
T-C: The time-cushion used
Stime: The simulated time to find solution(s); if more than one, earliest time for each is given (better-sol/worse-sol).
Rtime: The total runtime (computation time) to find solution(s) (in minutes).
H-r: The total number of hypothesis messages received and incorporated by nodes.
M-r: The total number of meta-level messages (node-plan and PGP) received and incorporated by nodes.
T-r: The total number of messages (hypothesis and meta-level) received and incorporated by nodes.
Store: The total number of structures stored (storage costs).

With environment B, different phenomena are encountered. In the broadcast organization, the best time-cushion is 1 (E8.4.8). A lower time-cushion (E8.4.7) does not improve coordination (solution time) while it does introduce substantially more computation and communication overhead (because nodes unnecessarily update their node-plans and PGPs more often). Meanwhile, a higher time-cushion (E8.4.9) degrades coordination because nodes do not adequately adapt to incorrect predictions about when they will exchange results. By the time nodes do respond to inappropriate PGPs, they have already wasted time on unnecessary actions (usually duplicating each others results) so network computation is increased due to this extra work. In addition, when a node does finally react to deviations in its local plans and updates its node-plans and PGPs, the exchange of the changed node-plans causes other nodes to change their plans and node-plans, and these cause other nodes to further change—so the net result is that once the nodes start changing their plans the effect snowballs, and that is why meta-level communication is also increased in this experiment.[11]

With a centralized organization, a lower time-cushion actually degrades coordination (E8.4.10), because nodes are *too* responsive. Specifically, the more constant stream of updated node-plans received by node 1 (the coordinating node) causes it to change the network PGPs and nodes oscillate between coordinating one way and then another. For example, the expectation about whether node 3 or node 4 will integrate d'_1–d'_2 and d'_3–d'_6 changes several times, where sticking to either decision would have been resulted in better performance.[12] A higher time-cushion (E8.4.12) also degrades coordination, but this time because nodes are not responsive enough. In the broadcast organization (E8.4.9), nodes build their own PGPs and this introduces inconsistencies that cause nodes to send substantial amounts of modified node-plan information back and forth whenever one node changes its node-plans. With a centralized organization, node 1 forms PGPs for the network, and by imposing consistent views of cooperation on the nodes, it

[11]Most of this extra communication activity occurs near the end of network problem solving when some nodes have finished their local responsibilities for important PGPs and begin pursuing and communicating about less highly-rated plans.

[12]These oscillations must eventually end. With each oscillation, one of the nodes performs some of the integration activities. Even if it is interrupted, when the oscillations once more make it believe it should integrate it picks up where it left off. Thus, despite oscillations some of the nodes make progress toward integrating results so that eventually one or more of them produces the integrated result.

prevents them from triggering these rounds of updates. As a consequence, the nodes pass fewer node-plans (decreasing the amount of communication). In turn, the PGPs formed by node 1 are modified much less frequently, so the nodes pursue PGPs based on outdated information and solution time suffers as a result. Because the network needs to invoke more KSs, overall network computation increases as well. Whether the savings in communication warrant this choice of time-cushion over the time-cushion of 1 (E8.4.11) depends and the available network resources.

To better illustrate how the different time-cushions affect network problem solving, we use as an example environment A with the broadcast organization. The summary for a time-cushion of 1 was presented in Figure 85. With a time-cushion of 0, the network activity is summarized in Figure 100. Unlike the case with a time-cushion of 1 (Figure 85) where nodes 1 and 3 take longer than expected to develop their partial results and coordination suffers, a time-cushion of 0 allows nodes to update their node-plans to reflect these deviations in local plans. Consequently, they recognize that node 1 should only form d_7–d_{11} and send it for integration, and that node 2 forms d_{12}–d_{15} at the same time as node 3 forms d_1–d_6. Node 3 integrates the three results and, since less redundant work is done, the overall solution is found earlier. On the other hand, a time-cushion of 2 changes network performance in a very different way, as shown in Figure 101. In this case, the nodes develop of view of cooperation early—at about time 29, when node 2 changes its plans based on predictive information from node 1. As with a time-cushion of 1 (Figure 85), plans deviate from expected so node 2 forms its piece much earlier than node 1 forms its piece. However, the larger time-cushion adds flexibility to the integration decisions,[13] and the PGPlanner of each node determines that node 4 is the appropriate node for integration because it is the least busy and because it can integrate the results almost as early (within this larger time-cushion) as node 1 could. The nodes pass their pieces to node 4, which forms the overall solution. Unlike the changes to PGPs triggered by node 1's changing of its local plans based on received results with a lower time-cushion

[13]Recall that the time-cushion essentially specifies a time tolerance for the network. This is used by nodes in deciding whether a plan that deviates from predictions should trigger updating PGPs or whether the deviation is tolerable. The time-cushion also indicates the tolerance for when results should be integrated: if two nodes could integrate results at about the same time (within the time-cushion tolerance) then the node with less important work is chosen (see Section 7.3.4).

(Figure 85), the larger time-cushion causes nodes to more predictably follow their PGPs.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-16		sensors			sensors			sensors			sensors	
16-17		d_1'			d_{10}			d_1				
17-18		d_1'			d_{10}			d_1				
18-19		d_9			d_{13}			d_1				
19-20		d_9			d_{13}			d_2				
20-21		d_9			d_{13}			d_2				
21-22		d_8			d_{13}			d_2				
22-23		d_8			d_{13}			d_1-d_2				
23-24		d_8			d_{13}			d_3				
24-25		d_8-d_9	d_8-d_9		d_{13}			d_3				
25-26		d_7			d_{13}			d_3				
26-27		d_7			d_{13}			d_1-d_3				
27-28		d_7		d_8-d_9	d_{14}			d_4				
28-29		d_7-d_9			d_{14}			d_4				
29-30		d_{10}			d_{14}			d_4				
30-31		d_{10}			d_{13}-d_{14}			d_1-d_4				
31-32		d_{10}			d_{15}			d_5				
32-33		d_7-d_{10}			d_{15}			d_5				
33-34		d_{11}			d_{15}			d_5				
34-35		d_{11}			d_{13}-d_{15}			d_1-d_5				
35-36		d_{11}			d_{12}			d_6				
36-37		d_7-d_{11}	d_7-d_{11}		d_{12}			d_6				
37-38		d_{12}			d_{12}			d_6				
38-39		d_7^2			d_{12}-d_{15}	d_{12}-d_{15}		d_1-d_6				
39-40		d_{10}^2			d_{11}		d_7-d_{11}	d_1-d_{11}				
40-41		d_9^2			d_{10}^2			d_1-d_{11}				
41-42		d_6^2			d_{10}^2		d_{12}-d_{15}	d_1-d_{15}				
42-43		d_6^2			d_{10}^2			*d_1-d_{15}*				

See Figure 85 for explanation.

Figure 100: Experiment E8.4.1—Environment A, Broadcast, Time-Cushion of 0.

time	Node 1			Node 2			Node 3			Node 4		
	rec	local sensors	snd	rec	local sensors	snd	rec	local sensors	snd	rec	local sensors	snd
1-16												
16-17		d'_1			d_{10}			d_1				
17-18		d'_1			d_{10}			d_1				
18-19		d_9			d_{13}			d_1				
19-20		d_9			d_{13}			d_2				
20-21		d_9			d_{13}			d_2				
21-22		d_8			d_{13}			d_2				
22-23		d_8			d_{13}			d_1-d_2				
23-24		d_8			d_{13}			d_3				
24-25		d_8-d_9	d_8-d_9		d_{13}			d_3				
25-26		d_7			d_{13}			d_3				
26-27		d_7			d_{13}			d_1-d_3				
27-28		d_7		d_8-d_9	d_{14}			d_4				
28-29		d_7-d_9			d_{14}			d_4				
29-30		d_{10}			d_{14}			d_4				
30-31		d_{10}			d_{13}-d_{14}			d_1-d_4				
31-32		d_{10}			d_{15}			d_5				
32-33		d_7-d_{10}			d_{15}			d_5				
33-34		d_{11}			d_{15}			d_5				
34-35		d_{11}			d_{13}-d_{15}	d_{13}-d_{15}		d_1-d_5				
35-36		d_{11}			d_{12}			d_6				
36-37		d_7-d_{11}			d_{12}			d_6				
37-38		d_{12}			d_{12}			d_6		d_{13}-d_{15}	d_{13}-d_{15}	
38-39		d_{12}			d_{12}-d_{15}			d_1-d_6	d_1-d_6			
39-40		d_{12}			d^2_{10}			d^2_1				
40-41		d_7-d_{12}	d_7-d_{12}		d^2_{10}			d^2_2				
41-42		d^2_7			d^2_{10}			d^2_1-d^2_2		d_1-d_6		
42-43		d^2_6			d^2_{10}			d^2_3				
43-44		d^2_8			d^2_{10}			d^2_1-d^2_3		d_7-d_{12}	d_7-d_{15}	
44-45		d^2_7-d^2_8			d^2_{10}			d^2_4			d_1-d_{12}	
45-46		d^2_9			d^2_{10}			d^2_1-d^2_4			d_1-d_{15}	
46-47		d^2_7-d^2_9			d^2_{10}			d^2_5			*d_1-d_{15}*	

See Figure 85 for explanation.

Figure 101: Experiment E8.4.3—Environment A, Centralized, Time-Cushion of 2.

8.1.5 Experiment Set 8.5: Heterogeneous Nodes and Reliability

This experiment set briefly addresses issues in how nodes can work together when they have different expertise and how they can coordinate to remain more robust in the face of node failures. We once again use environments A and B to illustrate how the PGPlanning mechanisms help nodes coordinate in these situations, and the experiments are summarized in Table 11.

As an initial investigation into issues in differing node expertise, we alter the environments so that a particular node is substantially better at integration than other nodes. This causes the PGPlanning mechanisms to generate PGPs where this node is the chosen integrator for the partial results. To establish one node as a better integrator, we alter the time needs of KSs at some nodes: parameters for integration KSs (to merge partial solutions) are changed in the non-integrating nodes so that they take 10 time units to run (as opposed to taking 1 time unit in the integrating node). This means that the other nodes take 10 times as long to integrate partial results. This information in reflected in the node-models (see Chapter 6) where the expected-integration-costs for the non-integrator nodes is 10 and for the integrator node is 1.[14] Since the PGPlanner uses this information when building the solution-construction-graph, it can focus integration tasks on the best node for integration.

Node 4 was chosen as the integrating node for environment A, since it is otherwise idle. With a broadcast organization (E8.5.1), the network is not quite as effective as when nodes were equally capable of integrating results (E8.2.1 in Table 8). As shown in Figure 102, the communication delay for node 1 to send its partial result d_7–d_{12} to node 4 causes the network to take slightly longer than before (in simulated and real time). However, in the centralized organization, the heterogeneity of nodes actually improves coordination slightly (comparing E8.5.2 with E8.2.2 in Table 8). When nodes were equally good at integration, the network expected node 1 to integrate its result of d_7–d_{11} with the node 3's result d_1–d_6, where the result from node 3 was expected to arrive before it was needed. In fact, the result from node 3 arrived later and node 1 waited for it

[14]This information is established during initial network formation, when the domain-level and meta-level organizations are built.

Table 11: Experiment Summary for Experiment Set 8.5.

Expt	Env	Org	STime	Rtime	H-r	M-r	T-r	Store
E8.5.1	A (4 integrator)	bc	47	65	6	48	54	1369
E8.5.2	A (4 integrator)	cn	45	49	4	39	43	1295
E8.5.3	B (4 integrator)	bc	26/37	39	7	60	67	1048
E8.5.4	B (4 integrator)	cn	28/39	37	7	71	78	1048
E8.5.5	A (redun)	bc	43	134	15	81	96	1333
E8.5.6	A (redun, 4 fails)	bc	43	128	11	63	74	1301
E8.5.7	A (redun, 4 int+fails)	bc	65	127	11	77	88	1439
E8.5.8	B (redun)	bc	25/34	63	14	116	130	1087
E8.5.9	B (redun, 4 fails)	bc	25/34	59	11	109	120	1041
E8.5.10	B (redun, 4 int+fails)	bc	26/60	83	21	141	162	1117

Abbreviations

Env: The problem solving environment (with comments about whether there is solution construction redundancy, whether node 4 is an integrator, and whether node 4 fails)

Org: Which of the meta-level organizations are used: broadcast, centralized, or none (if none, then no planning at all)

Stime: The simulated time to find solution(s); if more than one, earliest time for each is given (better-sol/worse-sol).

Rtime: The total runtime (computation time) to find solution(s) (in minutes).

H-r: The total number of hypothesis messages received and incorporated by nodes.

M-r: The total number of meta-level messages (node-plans and PGPs) received and incorporated by nodes.

T-r: The total number of messages (hypothesis and meta-level) received and incorporated by nodes.

Store: The total number of structures stored (storage costs).

and then combined the results. When all results are sent to node 4, the extra delay is avoided: the two results are at node 4 two time units after the later of the two is formed (because of communication delays) instead of node 4 receiving the combination 5 time units after the later piece is formed (delay of 2 for d_1–d_6 to get to node 1, 1 time unit to combine them, and then a delay of 2 for the combination to get to node 4).

time	Node 1			Node 2			Node 3			Node 4		
	rec	local	snd	rec	local	snd	rec	local	snd	rec	local	snd
1-16		sensors			sensors			sensors			sensors	
16-17		d'_1			d_{10}			d_1				
17-18		d'_1			d_{10}			d_1				
18-19		d_9			d_{13}			d_1				
19-20		d_9			d_{13}			d_2				
20-21		d_9			d_{13}			d_2				
21-22		d_8			d_{13}			d_2				
22-23		d_8			d_{13}			d_1-d_2				
23-24		d_8			d_{13}			d_3				
24-25		d_8-d_9	d_8-d_9		d_{13}			d_3				
25-26		d_7			d_{13}			d_3				
26-27		d_7			d_{13}			d_1-d_3				
27-28		d_7		d_8-d_9	d_{14}			d_4				
28-29		d_7-d_9			d_{14}			d_4				
29-30		d_{10}			d_{14}			d_4				
30-31		d_{10}			d_{13}-d_{14}			d_1-d_4				
31-32		d_{10}			d_{15}			d_5				
32-33		d_7-d_{10}			d_{15}			d_5				
33-34		d_{11}			d_{15}			d_5				
34-35		d_{11}			d_{13}-d_{15}	d_{13}-d_{15}		d_1-d_5				
35-36		d_{11}			d_{12}			d_6				
36-37		d_7-d_{11}			d_{12}			d_6				
37-38		d_{12}			d_{12}			d_6		d_{13}-d_{15}	d_{13}-d_{15}	
38-39		d_{12}			d_{12}-d_{15}			d_1-d_6	d_1-d_6			
39-40		d_{12}			d^2_{10}			d^2_1				
40-41		d_7-d_{12}	d_7-d_{12}		d^2_{10}			d^2_2				
41-42		d^2_7			d^2_{10}			d^2_1-d^2_2		d_1-d_6		
42-43		d^2_{10}			d^2_{10}			d^2_3				
43-44	d_{13}-d_{15}	d^2_6			d^2_{10}			d^2_1-d^2_3		d_7-d_{12}	d_7-d_{15}	
44-45		d^2_6			d^2_{10}			d^2_4			d_1-d_{12}	
45-46		d^2_6-d^2_7			d^2_{10}			d^2_1-d^2_4			d_1-d_{15}	
46-47		d^2_8			d^2_{10}			d^2_5			*d_1-d_{15}*	

See Figure 85 for explanation.

Figure 102: Experiment E8.5.1--Environment A, Broadcast, Heterogeneous.

In environment B, the least busy node is node 1. However, to see what happens when a node that has local data to process is also the best integrator, we once again make node 4 the integrating node. In the broadcast organization, this slows down the solution time for the better solution by one time unit because of the extra communication delay. Node 4 is interrupted by the integration tasks (to combine d_1–d_3 from node 3, d_4–d_5 from node 1, and d_6–d_8 from node 2) and forms its own piece of the less highly-rated solution (d'_3–d'_6) later than it otherwise would have. As a result, the simulated time when the less highly-rated solution is formed is increased compared to when any node(s) can integrate equally well (E8.2.5 in Table 8). Similar results hold with the centralized organization (E8.5.4).

We next explore some simple examples to get a flavor of how the PGPlanning mechanisms deal with node failure. When nodes can fail, plans should be more robust—more resilient to unexpected failures. This means promoting a certain degree of redundancy so that important tasks are being performed at more than one node. An alternative way of addressing node failure is to plan without redundancy but to monitor node interactions to detect when failures occur and then reassigning important tasks. A combination of the two methods could also be employed. Because detection and diagnosis of errors in a distributed domain is a complex problem [Hudlicka, 1986; Hudlicka *et al.*, 1986], the monitoring and repair of PGPs in the face of node failures remains a future research issue (see Chapter 9). The current implementation simply focuses on building more robust PGPs by increasing redundancy.

With environments A and B, we simulate a failure by dropping node 4 from the network after simulated time 30 (node 4 cannot run any local or communication KSs that start after time 30). It is therefore important that any integration of partial results that node 4 is expected to do should also fall to some other node or nodes. For each of the two environments, we examine three cases: where nodes have equivalent expertise and no nodes fail, where nodes have equivalent expertise and node 4 fails at time 30, and where node 4 has much better integration expertise (as in the earlier experiments in this set) but fails at time 30. In particular, when node 4 has better integration expertise (and thus attracts integration tasks), it is crucial that partial results are also sent to other nodes just in case node 4 fails. For each of the six experiments (E8.5.5–E8.5.10), the solution-construction-redundancy is set to 2, so that any partial solution is sent to

at least 2 nodes for integration with other partial solutions. We also only explore the broadcast meta-level organization since a centralized meta-level organization would be inappropriate where node failures are probable.

In environment A where nodes have equivalent expertise (E8.5.5), the solution is formed at simulated time 43 which is comparable to when it would be formed without the extra redundancy (see E8.4.1 and E8.4.2, for example). In fact, it is slightly better: without the redundancy, the nodes changed the PGPs several times as local plans changed (due to predictive information, for example) and the changing views about which node should integrate results delayed completion; with redundancy, PGPs changed over time but possible integrators changed less frequently (because two integrators rather than one are found at any time). However, the redundancy causes two nodes to generate this solution at the same time, and thus increases the computation and communication overhead (comparing E8.5.5 with E8.4.1 and E8.4.2). In the same environment but when node 4 fails (E8.5.6), redundancy allows the solution to be found at the same simulated time. When node 4 fails, it stops using network resources so the network computation, communication, and storage are slightly reduced. When node 4 has more expertise (E8.5.7), the solution is formed much later in simulated time, because node 4 fails and nodes with inferior KSs (that take ten times as long to run) must integrate partial results. At least the result was formed, however, which would not have been the case without the extra solution-construction-redundancy since nodes 1–3 would send partial results to node 4 and not know that node 4 had failed. As before, redundancy significantly increases the computation and communication overhead (comparing E8.5.7 with E8.5.2).

In environment B with equivalent expertise where no nodes fail (E8.5.8), the network performance in simulated time is once again comparable with the time needs without the more robust PGPs (E8.4.8) because node 4 is not responsible for integrating results. The redundancy once again increases the computation and communication overhead (comparing E8.5.8 with E8.4.8). When node 4 fails in this environment (E8.5.9), the network can still find the solution at the same simulated time and the computation, communication, and storage are slightly reduced since node 4 is idle. When node 4 does have better integration expertise (E8.5.10), several things happen. The solution time for the better solution is not seriously affected because node 4 generates it before failing. The solution time

for the second solution is very much affected, however, because node 4 fails and it is up to another node (in this case node 2) to integrate the results using its inferior KSs. Fortunately, before it failed node 4 did send its piece of this solution off to 3; without the extra redundancy introduced into the PGP, node 4 would not have shared this partial result since it would expect to do integration itself. The costs of the additional redundancy are once again high (more computation and overhead comparing E8.5.10 with E8.5.3), but the additional resiliency of the PGPs enabled the network to find the solutions despite node 4 failing.

Although these experiments have only scratched the surface of the possible issues in building reliability into network coordination, they do illustrate how more robust PGPs can be developed within the PGPlanning mechanisms, and the costs of doing so. When we discuss future research directions in the next chapter, one of the principal areas where research can be done concentrates on building other mechanisms for monitoring cooperation, so that nodes can not only dynamically identify when other nodes fail, but can also better deal with failed message transmissions and faulty links between nodes and sensors.

8.1.6 Experiment Set 8.6: Larger Networks

The previous experiments have examined how the local and partial global planning mechanisms affect coordination in networks with four or fewer nodes. It is also important to consider how these mechanisms work in larger networks, where the costs and complexities of coordinating nodes can be higher. In this experiment set, we perform a preliminary exploration into these issues.

The experiments in this set are based on the sensor and data configuration shown in Figure 103. There are 10 sensors, arranged diagonally, whose regions have some degree of overlap. In what is essentially an extension of environment B (Figure 84), two vehicles move in parallel among the nodes, but while one vehicle consistently generates moderately sensed data, the other vehicle track alternates between strongly and weakly sensed sections. From this sensor and data configuration we develop two networks. One network is composed of 10 nodes, where node i receives data from sensor i. In this network, the nodes are organized at the domain-level as a *lateral* network: the nodes pass hypotheses among themselves so

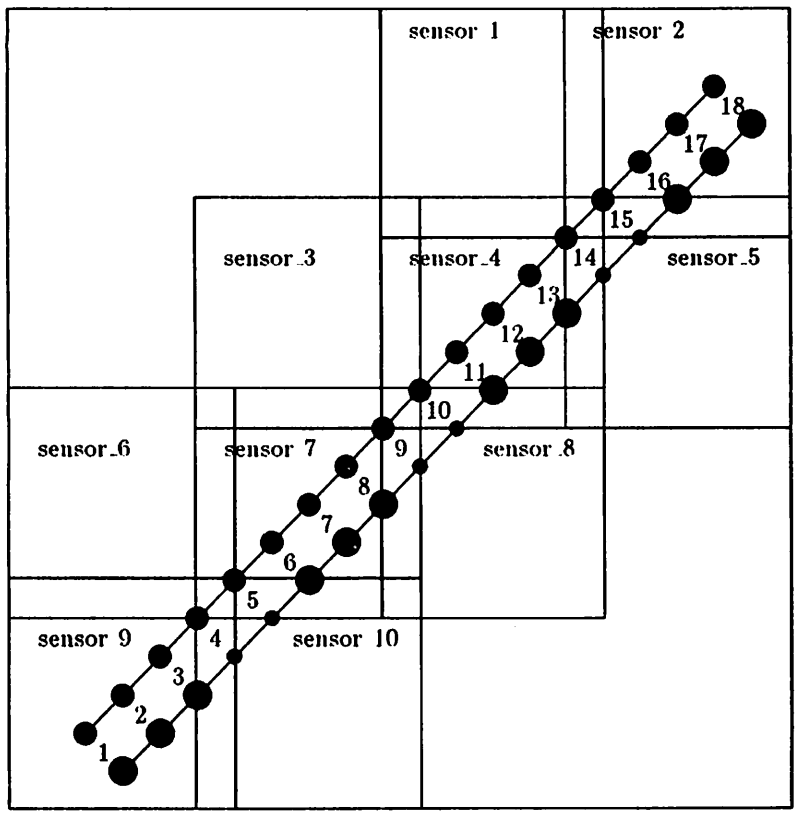
that one or more of them eventually forms overall solutions.[15] The other network is composed of 11 nodes, where the 10 nodes connected to different sensors send hypotheses to the eleventh node which is solely responsible for integrating results. The domain-level organization in this network is thus *centralized*, since node 11 does high-level (integration) activities from the low-level (synthesis) results from the other nodes.

Before we summarize the experimental results for these networks, we first note that in these experiments certain concessions were made due to the limitations of the computing facilities. For example, in the experiments without local or partial global planning, we would have liked to give the nodes all of the other control mechanisms—especially the ability to form subgoals to improve control decisions. Unlike the experiments with the smaller networks where subgoaling was employed, we were forced to turn off the subgoaling mechanisms in the larger networks because the huge number of subgoals used up all available memory and caused the time-shared computer to crash, making the experimenter rather unpopular.[16]

The experimental results for the larger networks are summarized in Table 12. We begin with the 10-node network (lateral domain-level organization) without any of the new local and partial global planning mechanisms (E8.6.1). This experiment was terminated after nearly 24 hours of CPU time, when the progress being made was deemed insignificant because of memory thrashing. Although the domain-level organization restricts communication so that nodes only send hypotheses to neighboring nodes that could extend them, still the amount of communication between nodes causes excessive time being spent on incorporating received information. Thus, the simulation gets bogged down: although the simulated time is not extremely high, the actual computation time needed to perform the simulation is. Because the nodes pursue the strongly sensed data first, the network forms d'_1–d'_{18} first (and because nodes are not well coordinated,

[15]To reduce the amount of communication, the domain-level organization only allows nodes to exchange hypotheses with neighboring nodes (whose sensed areas are adjacent). Thus, although the communication network is broadcast, the organization constrains communication so that large amounts of information are not sent around the network. By doing so, the network performance is improved since nodes do not need to incorporate as much information.

[16]Note that subgoaling was turned off in the environments without the planning mechanisms. In experiments with the planning mechanisms, nodes were allowed to subgoal, but the planning mechanisms controlled subgoaling so that an excessive number of subgoals were not formed.



The sensors' ranges overlap, there are 18 data points in both of the tracks, and the size reflects how strongly sensed the data point is.

Figure 103: Ten-Sensor Configuration with Sensed Data.

Table 12: Experiment Summary for Experiment Set 8.6.

Expt	Nds	D-L-O	M-L-O	STime	Rtime	H-r	M-r	T-r	Store
E8.6.1	10	lateral	none	86+/58	2005	329	0	329	6738
E8.6.2	10	lateral	local	49/65	583	216	0	216	4238
E8.6.3	10	lateral	broadc	42/57	429	19	396	415	3264
E8.6.4	10	lateral	central	42/50	208	41	322	363	3073
E8.6.5	10	lateral	hierarc	41/51	161	19	384	403	2864
E8.6.6	11	central	none	81/54	186	119	0	119	4197
E8.6.7	11	central	central	47/54	146	14	222	236	2721
E8.6.8	11	central	broadc	40/51	262	14	370	384	2914

Abbreviations

Nodes: The number of nodes in the network
D-L Org: Which of the domain-level organizations are used: lateral (any node forms result) or centralized (node 11 forms result)
M-L Org: Which of the meta-level organizations are used: broadcast, centralized, hierarchical, local (no PGPlanning), or none (no planning at all)
Stime: The simulated time to find solution(s); if more than one, earliest time for each is given (better-sol/worse-sol).
Rtime: The total runtime (computation time) to find solution(s) (in minutes).
H-r: The total number of hypothesis messages received and incorporated by nodes.
M-r: The total number of meta-level messages (node-plan and PGP) received and incorporated by nodes.
T-r: The total number of messages received and incorporated by nodes.
Store: The total number of structures stored (storage costs).

several nodes form this solution redundantly).

When nodes are able to plan their local activities (E8.6.2), performance generally improves. Although the less highly-rated solution is found somewhat later than in E8.6.1, the more highly-rated solution is found much faster. The better local control through planning reduces the number of useless hypotheses formed and exchanged, so that the computation time, the amount of communication, and the overall storage are all reduced. Even though they are no more coordinated than in E8.6.1, the nodes still work better together because each is a better individual problem solver. However, problems still arise due to lack of coordination, such as several nodes redundantly forming solutions.

The first meta-level organization we try allows nodes to broadcast their node-plans and to form PGPs individually (E8.6.3). Nodes are essentially duplicating each other's PGPlanning, and to reduce the overhead incurred we set the time-

cushion to 5 so that nodes change node-plans and PGPs less often. PGPlanning leads to better simulated performance for finding both solutions. Its computation overhead is acceptable, since the overall computation costs are reduced (comparing E8.6.3 with E8.6.2), and the reduction in the number of KSIs invoked reduces the storage needs. However, to build and maintain the PGPs does require a substantial amount of communication (recall we measure the messages received so a single broadcast node-plan triggers 9 receptions).

To reduce the network communication and the computation overhead incurred by having nodes redundantly derive PGPs, we next give the network a centralized meta-level organization (E8.6.4). Node 1 receives node-plans from the other nodes, builds PGPs, and sends these PGPs back to the other nodes. Because fewer nodes are building PGPs, we allow the network to be more responsive by setting the time-cushion to 1. As a consequence, the network performance is improved because the less highly-rated solution is found earlier. The computation costs are substantially reduced because only one node is recognizing PGGs and building PGPs (recall once again that this is a reduction in network computation and does not take into consideration possible parallelism in control computations). Moreover, this meta-level organization reduces communication costs, even though more node-plans are sent since the time-cushion is lower. Because it must correlate node-plans from all 10 nodes (and incurs the combinatorics associated with finding all possible combinations), however, node 1 does require substantial computation to build the PGPs.

As a final experiment with the 10 node network, we try a hierarchical meta-level organization to relieve the coordinating node from having to deal with so many combinations (E8.6.5). In this organization, we have half of the nodes send their node-plans to one node (nodes 2–5 send to node 1) and the other half send to another node (nodes 6–9 send to node 10). Each of these nodes builds PGPs for its subset of nodes (including itself) and sends its PGPs to a higher-level coordinator (node 6). The top-level coordinator combines and modifies the PGPs it gets and sends the revised PGPs back to the middle-level nodes, which in turn send PGP information on to the bottom-level nodes. The organization is thus a multi-level hierarchy. Nodes 1 and 10 each correlate node-plans for half the network, which is much simpler than correlating node-plans for the entire network because the number of possible combinations rises exponentially with

the number of node-plans. Node 6 gets these PGPs which "pre-digest" the node-plans, so node 6 needs only to recognize that the PGPs should be combined and to reorder activities appropriately for the complete PGPs. Although the simulated performance is not substantially changed form the centralized meta-level organization (E8.6.4), the overall computation costs are reduced because the coordination problem is decomposed into simpler problems that are distributed among the nodes in the multi-level hierarchy. Having a multi-level hierarchy does increase communication, however, since the nodes in the hierarchy must communicate more (less encompassing) PGPs.

Multi-level hierarchies that decompose the coordination problem are important organizations in many types of cooperative domains (human as well as machine). Our PGPlanning mechanisms were purposely made to be flexible so that experiments with such hierarchies could be performed. However, a drawback with the current implementation is that the PGPs passed among any levels of the hierarchy are essentially the same. This means that, for example, the top-level node in a tall hierarchy must still reason about the actions of individual nodes when coordinating the network. To avoid this, we could allow a node that coordinates some subset of nodes to build a single "group-plan" for that subset. This group-plan would summarize the sequential activities of the group without detailing the actions of individual nodes, so the plan-activity-map would indicate for an activity the group performing it. Higher-level nodes could treat these group-plans as they would node-plans, and would combine them into PGPs to reason about the concurrent activities of several groups. These extensions to the PGPlanning mechanisms are future research directions that we hope to pursue (see Chapter 9).

Turning now to the 11 node network (where node 11 integrates hypotheses formed by the other 10 nodes), we briefly consider issues in coordination. First, we experiment with this network where nodes have neither the local nor the partial global planning mechanisms (E8.6.6). First, it is clear that the centralized domain-level organization performs much better than the lateral organization (comparing E8.6.6 with E8.6.1), not so much in simulated performance but in the actual amount of computation that goes on. This is attributable to the more focused communication. By concentrating communication on node 11, only one node is receiving hypotheses and performing all of the control tasks (goal pro-

cessing) associated with these hypotheses. Consequently, the simulation takes one-tenth the time to run. Its simulated performance, however, is not substantially different, and once again the less highly-rated solution is found significantly earlier.

If the domain-level organization is centralized (a single node is responsible for finding the overall solution), then reliability will not suffer if the meta-level organization is centralized as well (E8.6.7). Node 11 is thus responsible for both integration of problem information and coordination of the network. As expected, this coordination improves simulated performance. The computation costs were also reduced, because the savings in problem solving more than compensated for the additional coordination overhead. The savings in problem solving also reduces the storage since fewer unneeded hypotheses are formed. Once again, however, communication costs are increased since nodes must communicate meta-level information in order to coordinate their plans.

A centralized domain-level organization imposes more structure on how nodes work together on the problem than a lateral organization does because roles and responsibilities are more compartmentalized among the nodes. Given so much structure, the nodes probably could not take as much advantage of the local autonomy and responsiveness that a broadcast meta-level organization provides. However, to illustrate how our new mechanisms separate meta-level from domain-level organization, we impose a broadcast meta-level organization over a centralized domain-level organization (E8.6.8). Because of the increased responsiveness (nodes need not wait for the coordinator), the simulated performance is somewhat improved (compared to E8.6.7). However, the network computation overhead incurred for coordination is high—and even exceeds the computation needs when no local or partial global planning mechanisms are used (E8.6.6). The communication costs are also substantially increased. Thus, this experiment shows both that the mechanisms are flexible (allowing different combinations of domain-level and meta-level organizations) and that their application should be controlled in certain complex situations since their costs may exceed their benefits.

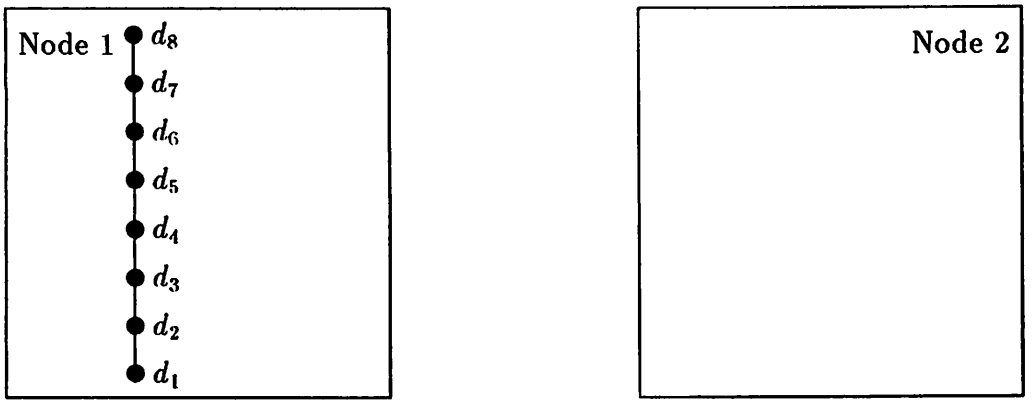
In summary, this experiment set has briefly explored how the mechanisms work in larger networks. We have recognized that as the number of nodes to coordinate increases, so does the cost of coordinating them. In most of our examples, the coordination overhead was acceptable, but judicious application

of the planning mechanisms becomes increasingly important (just as the subgoaling mechanisms could not be used indiscriminately in these environments). The choice of meta-level organization also affects the costs, and these experiments illustrated how decomposing the coordination tasks, or redundantly having nodes form PGPs, influences overhead. Although more large network experiments should be done (when the computation resources allow), the preliminary indications are that the PGPlanning mechanisms do scale up when used appropriately,.

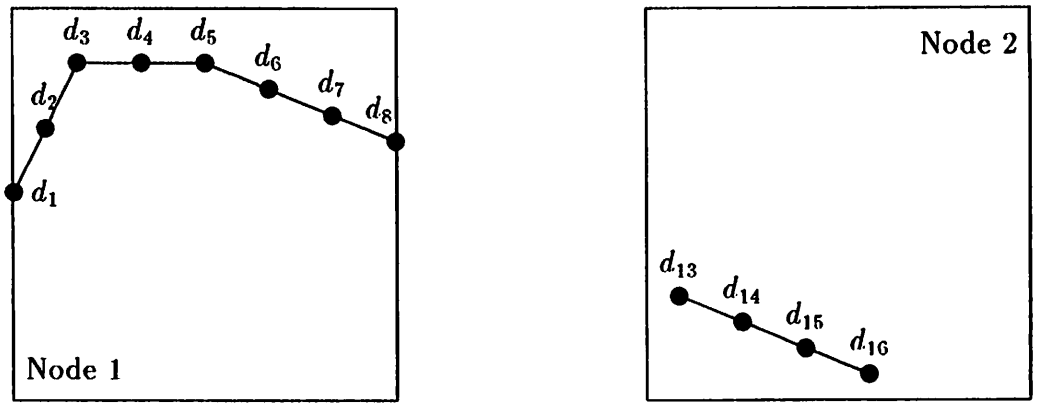
8.1.7 Experiment Set 8.7: Reassigning Tasks to Improve Cooperation

We next illustrate how task-passing is done with the PGPlanner. We begin with a simple two node environment (7A), with sensors and data as shown in Figure 104. A vehicle moves through the area sensed by node 1 and does not at all enter the area sensed by node 2. This is a situation where task-passing is definitely appropriate to improve simulated concurrency: node 1 should pass some of its tasks (data to process) off to node 2, which would otherwise sit idle. Note, however, that passing tasks does not reduce the amount of problem solving in the network, but only redistributes it. Thus, task-passing will not reduce overall computational needs of the network, and may actually increase computation, communication, and storage costs since nodes incur overhead as they decide where to send tasks, send them, and then store received tasks. The overhead of task-passing promotes parallelism that reduces the simulated time the network needs for generating solutions, so that although network computation is increased, parallelism in computation can reduce the time needed to solve problems.

The experimental results are summarized in Table 13. With a meta-level organization where nodes do not communicate about local plans, the network (node 1) needs 41 simulated time units to generate the solution (E8.7.1). In a broadcast organization (E8.7.2), the nodes exchange node-plans—where node 2 builds an idle node-plan—and form PGPs. The data arrives over the time interval 1-9, so local planning is done at time 9 and, because of communication delays, the nodes have models of each other at time 11. At this point, each of them modifies its PGP to represent a possible exchange of tasks. Since node 1 possesses those tasks and there is no coordinating node specified by the meta-level organization,



Environment 7A



Environment 7B

Figure 104: Task Passing Environments.

Table 13: Experiment Summary for Experiment Set 8.7.

Expt	Env	Org	STime	Rtime	H-r	M-r	T-r	Store
E8.7.1	7A	local	41	8.6	1	0	1	305
E8.7.2	7A	broadcast	32	12.2	16	15	31	408
E8.7.3	7A	central	30	11.9	16	15	31	400
E8.7.4	7B	broadcast	44/34	13.6	0	21	21	483
E8.7.5	7B	central	44/34	13.8	0	17	17	475
E8.7.6	7B (local future)	broadcast	44/34	13.8	0	23	23	483
E8.7.7	7B (local future)	central	44/34	13.8	0	17	17	475
E8.7.8	A (task-passing)	broadcast	48	97	77	65	142	1652
E8.7.9	A (task-passing)	central	52	120	81	139	220	1841

Abbreviations

Env: The problem solving environment
Org: Which of the meta-level organizations are used: broadcast, centralized, or local (if local, then no PGPlanning)
Stime: The simulated time to find solution(s); if more than one, earliest time for each is given (better-sol/worse-sol).
Rtime: The total runtime (computation time) to find solution(s) (in minutes).
H-r: The total number of hypothesis messages received and incorporated by nodes.
M-r: The total number of meta-level messages (node-plan and PGP) received and incorporated by nodes.
T-r: The total number of messages received and incorporated by nodes.
Store: The total number of structures stored (storage costs).

node 1 initiates task-passing by sending its modified PGP to node 2. Node 2 receives this at time 13, extracts out the expected tasks, checks them against any future node-plans (it has none), and sends the future node-plan back indicating that it can carry out the tasks right away. Node 1 gets this reply at time 15 and passes the task by sending the data (hypotheses) and the PGP assigning the task to node 2. At time 17, node 2 receives this data, builds a local plan to process it, and fits this local plan into the PGP (including determining when and where it should send the results it forms from the data). It builds the track d_6–d_8 and sends it back to node 1, where the track d_1–d_5 has been formed.

If the organization is centralized (E8.7.3), where node 2 is the coordinator, then behavior is slightly different because the node making task-passing decisions (node 2) is not the same as the node with tasks to pass (node 1). Local planning once again occurs at time 9, and this time node 1 sends its node plans to node 2. At time 11, node 2 forms the more global PGPs and recognizes that it itself is underutilized. Not only does it modify the PGP to represent an exchange of

tasks from node 1 to node 2, but it also forms the future node-plan in response to this PGP and decides to award the task to itself. At this point, it sends the modified PGP to node 1 since node 1 must supply the tasks. Node 1 receives this PGP at time 13 and immediately sends the hypotheses to node 2. Thus, node 2 begins its task at time 15 (instead of time 17 as in E8.7.2) and the nodes form the overall solution 2 time units earlier as a result.

While environment 7A represents a case where task-passing is appropriate, environment 7B does not (Figure 104). As in environment 7A, there are two nodes with different sensor areas. In this case however, the vehicle that first passes through node 1's area does eventually make it to node 2's area as well. Because it passes through other space between the nodes (which may be sensed by some other node not simulated here), the vehicle causes node 2 to sense data later than node 1: we simulate the sensed time occurring at the corresponding simulated time, so node 2 does not begin getting its own data until time 13. Initially, therefore, node 2 is idle when it attempts to coordinate with node 1, but it later will be getting data of its own. When nodes cannot predict future plans, tasks are passed from node 1 to node 2 so that node 2 helps node 1 build its partial result, but by working on node 1's data, node 2 delays building the partial solution out of its own sensor data. This environment thus calls for prediction of future plans based on possible extensions of existing plans to foresee when a node is likely to get sensor data in the future.

We begin once again with a broadcast organization (E8.7.4). As in environment 7A, the time interval from 1 to 9 is used to build incorporate the sensed data (by node 1). At time 9 the nodes build local plans and transmit them, so at time 11 they both recognize that node 2 is currently underutilized. Before node 1 modifies and sends the PGP requesting node 2 to bid on the tasks, it first checks the PGPs to see if node 2 might get data in the near future. By extending the projected track of the vehicle to later time frames, it determines that the vehicle may pass through node 2's area, and that node 2 should start getting data at time 13. As a consequence, node 1 does not modify and send a PGP because there would not be time for node 2 to complete the task before it got data of its own. No tasks are passed, and node 1 forms d_1–d_8 on its own while node 2 forms d_{13}–d_{16} by itself.

Much the same thing occurs with a centralized organization (E8.7.5). At time

11, node 2 forms the more global PGP and determines that task-passing may be in order. Before modifying the PGP, it checks to see if it might get new data in the future based on projected vehicle movements and discovers that it should. Thus, it builds a future node-plan to represent this probable future activity and does not modify the PGP since it could not complete the tasks before the future activities are expected to begin.

As a variation on this theme, we also consider what would happen if the node that modifies the PGP has an incomplete view. In this case, it might be unaware of possible future node-plans for task recipients, and so might request bids from inappropriate nodes. We simulate this by simply not allowing a node to form future node-plans for anyone but itself. Thus, with the broadcast meta-level organization (E8.7.6), node 1 builds the PGP at time 11 and modifies it because it cannot project future node-plans for node 2. When node 2 receives this PGP, however, it builds the future node-plan before it replies to the PGP. Since it sees that local tasks are imminent (data is about to arrive), node 2 extracts the activities from the received node-plan and modifies them to indicate that it will begin them when it is done with the local tasks it expects to get. When node 1 receives this node-plan from node 2 and compares it with the PGP, it recognizes that the tasks will be done sooner where they are (at node 1) than they would be if they were passed (to node 2). Consequently, node 1 modifies the PGP to indicate that the tasks will not be passed, and each node pursues its local plans. This shows how the mechanisms can be robust in the face of incomplete information, since both the sending and potential recipient nodes' views affect task-passing. A centralized organization, with node 2 as coordinator (E8.7.7), has the same behavior as before (E8.7.5) because node 2 recognizes its own future node-plan when deciding whether to modify the PGP.

Finally, we look at a somewhat more complicated example: the four-node environment A. This example points out possible pitfalls in task-passing in situations where nodes are uncertain about what tasks are important, because in this case nodes pass tasks that do not contribute to the overall solution. In a broadcast organization, the nodes exchange node-plans and each builds more global PGPs at time 18. Each node recognizes that node 2 is currently a bottleneck: given the ambiguity of its data, it is expected to form d_{13}-d_{15} well after nodes 1 and 3 form their pieces. In addition, each node identifies node 4 as being underutilized

(idle) and capable of performing some of node 2's tasks. Each node modifies the relevant PGP to indicate a possible transfer of tasks (to process data d_{14} and d_{15}), and node 2 sends its PGP to node 4 to request bids. At time 20, node 4 responds to the received PGP indicating that it could do the tasks as expected since it has no other future node-plans. Node 2 gets this information at time 22 and immediately sends those tasks—which entails a fairly large amount of communication since their is a large amount of data for each of these sensed times. At time 24, node 4 gets this data, forms local plans to process it, links these plans to the more global PGP, and begins processing this data. However, node 2 receives the predictive information from node 1 at time 27, and when it changes its local plans in response (to focus on certain data) it is no longer a bottleneck node. Since no node is a substantial bottleneck anymore (nodes 1, 2, and 3 expect to form their pieces of the solution at about the same time), node 4 need not receive any new tasks. Instead, node 4 simply discontinues pursuing its local plan since it expects node 2 to form the relevant piece sooner. From this point on, problem solving is essentially the same as when no task-passing is done.

In a centralized meta-level organization, behavior is similar. This time, node 4 as coordinator determines that it should receive tasks from node 2 and sends node 2 the PGP indicating that it should send the task out. Node 2 gets this PGP at time 20 and sends the relevant data (d_{14} and d_{15}) to node 4, which receives it and begins working on it at time 22. Once again, the predictive information from node 1 triggers drastic changes in the expected interactions between nodes, causing the task-passing to be unnecessary. It is also interesting to note that the extra delays in coordinating nodes in this environment and the extra problem solving performed in the network (because coordination is not as effective as in the broadcast organization) add to the computation, communication, and storage overhead.

These last experiments thus not only show how task-passing can occur as a response to an unbalanced PGP—where a participating node is expected to take much longer to form its partial solution than any other node—but also how task-passing might not turn out to be such a good idea. Although it was reasonable at the time, the decision to send tasks turns out to use up computation and communication resources unnecessarily. These experiments thus show the capabilities and the limitations of the task-passing mechanisms: how they can

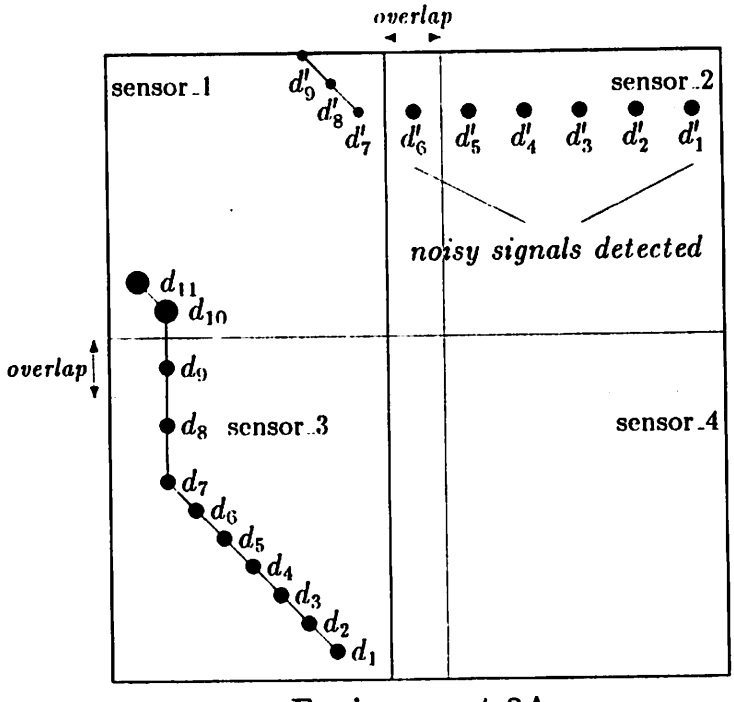
redistribute load for important PGPs, but also how they currently use only simple criteria for recognizing how and when to do so. The mechanisms do not yet have the sophistication to anticipate when predictive information may change a node's local plans, and they also are geared toward passing tasks from a bottleneck node to an underutilized node, and cannot deal well with the situation exemplified by the latter part of these experiments where no node is a bottleneck but if each passes small tasks to node 4 the overall network performance could possibly still improve.

In summary, then, this experiment set illustrated the benefits and the costs of task-passing and the importance of anticipating future plans when deciding when to transfer tasks between nodes. The experiments also showed some limitations of the mechanisms, opening up avenues for future research. Nonetheless, the experiments show how the PGPlanning mechanisms provide nodes with the ability to identify, represent, and coordinate task-passing activities to possibly improve network performance.

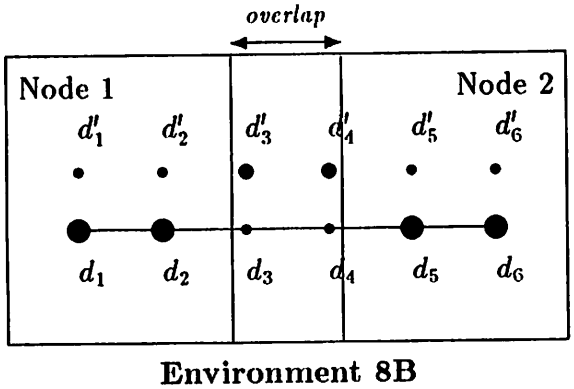
8.1.8 Experiment Set 8.8: Miscellaneous Experiments

This experiment set ties up some loose ends by using certain environments to point out some of the PGPlanner's additional capabilities—and costs. To show some of these capabilities, we begin with environment 8A, shown in Figure 105. This is a four-node network, where two vehicles move among the nodes. Node 1 senses data for both vehicles, node 2 only senses one of the vehicles, node 3 only senses the other vehicle, and node 4 senses neither of the vehicles. Once again, node 2's sensor detects ambiguous data, so that it expects to take more time to generate partial solutions, unless node 1 provides predictive results to help it.

The experimental results are summarized in Table 14, E8.8.1-E8.8.4. First, without any of the planning mechanisms at all (E8.8.1), the network performs so badly that the experiment was aborted after 14 hours of runtime because the thrashing of memory was making progress on problem solving very slow. After it had found the more highly-rated solution (d_1-d_{11}), the network had difficulty forming the other solution (d'_1 d'_9) because of the ambiguous data. The addition of local planning (E8.8.2) substantially improves network performance because each node forms its local solutions more effectively. Once node 1 forms d'_6-d'_9 and sends this predictive information to node 2, node 2 modifies its local plans



Environment 8A



Environment 8B

Figure 105: Miscellaneous Environments.

so that it finds the compatible track d'_1–d'_5 more effectively. However, node 1 forms d_9–d_{11} before d'_6–d'_9 because it is more strongly sensed, and so node 2 gets predictive information later than it otherwise might.

With the PGPlanning mechanisms and a broadcast meta-level organization (E8.8.3), node 1 recognizes several things: that the PGP to form d_1–d_{11} is more highly rated; that node 3 will take much longer to form its piece of the overall solution for this PGP; that node 2 is working on the other PGP (d'_1–d'_9) and might benefit from predictive information from node 1; and that node 1 could possibly develop this predictive information early on without hurting cooperation on the more highly-rated PGP. In short, node 1 sees that it has time to spare for working on the more highly-rated PGP, and so it could try to help out node 2 in that spare time. Node 1 thus forms and sends d'_7–d'_8 to node 2 early on, and still sends d_9–d_{11} to node 3 soon enough to avoid disrupting coordination. As a result, the network finds the better solution no later and the other solution substantially earlier (comparing solution times for E8.8.3 and E8.8.2). The improved decisions reduce overall computation and storage needs, but the improved coordination does add communication overhead. Finally, with a centralized organization (E8.8.4), node 1 once again helps out node 2 early on, although the extra communication delays to and from the coordinator (node 4) cause this help to be delayed by 2 time units. The better solution is not found as quickly, however, because node 3 takes longer to form its partial solution than node 4 expected (since this deviation was within the time-cushion). This imperfect coordination comes about because of the extra time needed to modify PGPs (delays to and from the coordinator). It delays solution construction and also increases the computation, communication, and storage costs, compared to the broadcast organization.

These experiments thus show how the PGPlanning mechanisms do allow flexibility in how and when nodes pursue PGPs, based on their views of coordination. Although it prefers to pursue highly-rated PGPs, a node can recognize cases where it has available time (where some other node is a bottleneck) and can determine whether it could use that time to help nodes pursue other PGPs. This flexibility comes about because nodes do represent how and when they will interact in their PGPs, and with this representation can find room for cooperation in other PGPs.

In environment 8B (Figure 105), we complicate things to increase our understanding of the costs and limitations of the local and partial global planning

Table 14: Experiment Summary for Experiment Set 8.8.

Expt	Env	Org	STime	Rtime	H-r	M-r	T-r	Store
E8.8.1	8A	none	62/294+	840+	116+	0	116+	5189+
E8.8.2	8A	local	45/56	55	17	0	17	1457
E8.8.3	8A	broadcast	45/41	53	5	42	47	1299
E8.8.4	8A	central	51/43	58	4	67	71	1323
E8.8.5	8B	none	78	87	47	0	47	1420
E8.8.6	8B	local	72	361	28	0	28	1808
E8.8.7	8B	broadcast	40	564	7	69	76	1339

Abbreviations

Env: The problem solving environment
Org: Which of the meta-level organizations are used: broadcast, centralized, or none (if none, then no planning at all)
Stime: The simulated time to find solution(s); if more than one, earliest time for each is given (better-sol/worse-sol).
Rtime: The total runtime (computation time) to find solution(s) (in minutes).
H-r: The total number of hypothesis messages received and incorporated by nodes.
M-r: The total number of meta-level messages received and incorporated by nodes.
T-r: The total number of messages received and incorporated by nodes.
Store: The total number of structures stored (storage costs).

mechanisms. In this environment, the actual vehicle track d_1–d_6 is paralleled by a "ghost" track d'_1–d'_6, possibly caused by reflections of sound in the environment. The actual and ghost data are close enough to lead to confusion, so that nodes can build tracks incorporating both actual and ghost data. This confusion leads the planning mechanisms into proposing a much larger number of possible local plans and into forming many more PGPs for different combinations of these plans. Thus, the planning overhead may not be worthwhile given the combinatorics.

When neither local nor partial global planning is used (E8.8.5), the nodes each form hypotheses based on local criteria (KSI ratings) and problem solving is essentially data-driven. Without the longer term view of potential solutions to work toward, the nodes make poorer control decisions and must invoke more KSIs, but the overhead in making control decisions is lower. Giving the nodes the ability to form local plans (E8.8.6) allows them to make better control decisions, which reduces the simulated time needs compared to the experiment without planning (E8.8.5). However, each node has two data points for each of the four sensed times, and thus locally forms 16 alternative-goals (combinations of data). Although the local planner does not initially form plans for each alternative-goal

because it can pursue several concurrently, over time more plans are generated for smaller subsets of these alternative-goals. The planner must reason about more information, and thus the computation overhead is substantially increased. Local planning allows nodes to make better control decisions and to reduce communication costs, but it increases computational needs (and also adds to storage since the number of additional planning structures exceeds the number of hypotheses, goals, and KSIs that are not generated because of the better control decisions).

When nodes are also given PGPlanning abilities (E8.8.7), the control decisions improve even more. However, the combinatoric problems are compounded: in a worst case view, each node forms 16 plans, and every possible combination of plans leads to a PGP, meaning that there are 16^2 (256) PGPs. Not only is finding and storing all of these combinations expensive, but also reasoning about all of these PGPs can be very costly. This can be seen from the summarized results in E8.8.7. The amount of computation is extremely high, the communication about plans is very costly, and the savings in storage caused by making the much better control decisions (forming fewer hypotheses, goals, and KSIs) is practically offset by the additional storage needs for all the plans and PGPs.

These experiments show some of the important costs of the local and partial global planning mechanisms. Planning assumes that the nodes have sufficient information to develop a small set of reasonable goals to work toward. When this assumption holds, planning lets nodes pursue the goals more efficiently. However, when this assumption does not hold, planning is not a useful strategy. Having too many goals is much like having no goals at all: there is insufficient context for choosing some actions over others. The goals do not allow discrimination among actions (because each action corresponds to some goals), and therefore do not help control. In such a situation, the nodes may as well not plan at all (to save on computation overhead), at least initially. Perhaps after a certain amount of data-driven problem solving there will be a better basis for forming goals and plans (some data might fit together well while some data does not).

The planning mechanisms should thus be used judiciously. In essence, the mechanisms for controlling problem solving (local and partial global planning) should themselves be controlled by meta-level control mechanisms (for control of control). It is clear from earlier experiments that there are many situations where local and partial global planning are particularly cost effective; but these

recent experiments also show that planning is not appropriate in all situations. A meta-level control component would make decisions based on the particular situation about how much or little planning should be done. Such a component remains an open research avenue, and is discussed in the next chapter.

8.2 Evaluation

The experiments show that partial global planning improves network coordination, but at a price. In most of the situations we explored, the benefits of partial global planning typically outweighed the costs: better control of actions and interactions reduces the overall amount of computation needed to solve the problem despite the computation overhead needed for that control. We also saw, however, that in particularly complex or ambiguous situations, the cost of control may not be worthwhile. Sometimes a brute force approach to problem solving is more cost effective than a more controlled approach.

Another important cost of the mechanisms is in communication. By dynamically coordinating their interactions, the nodes reduce their domain-level communication because they have more knowledge about what results are worth exchanging in the current situation. To develop this knowledge, however, requires nodes to communicate meta-level information, and the amount of additional meta-level communication usually exceeds the amount of domain-level communication saved. Communication is unavoidable if nodes are to form more dynamic views of network activity, but meta-level communication decisions could be more selective than they currently are, and this is an area for further research described in the next chapter.

The additional storage costs of the plan and PGP information are usually offset by savings in domain-level storage, since better control reduces the amount of domain-level information generated and stored. However, in particularly pathological cases where many plans and PGPs are possible, these costs also may not be worthwhile.

When applied *judiciously*, therefore, the partial global planning mechanisms can be cost effective. The mechanisms have limitations, especially in complex or ambiguous situations where it is impossible or overly time-consuming to form an adequate view for control. An important avenue of future research is the

development of meta-level control to control the control mechanisms, so that the planning mechanisms are applied only when they will be of use. We discuss this direction for future research in the next chapter.

From a practical standpoint, therefore, we have seen that the new mechanisms are often cost effective. We have also witnessed their flexibility: how they allow cooperation in different styles through different meta-level organizations; and how they promote a variety of goals of cooperation, such as avoiding unnecessary redundancy, balancing load, exploiting expertise, and improving reliability through redundancy. Having this flexibility in a single approach to dynamic coordination is both practical (since coordination should adapt to different distributed problem situations) and conceptually pleasing (since a single framework promotes cooperation in a variety of situations to meet different goals). In the next chapter, we focus on how the new mechanisms address the research issues raised in the first chapter and thus detail the contributions of this research.

Based on the preliminary experimental results, therefore, partial global planning is a suitable approach to coordination in a wide (but bounded) range of cooperative computing situations. Although further experimentation may reveal other costs, benefits, limitations, and capabilities of the approach, the initial evaluation shows that the approach shows promise and warrants further exploration.

If to do were as easy as to know what were good to do, chapels had been churches, and poor men's cottages princes' palaces. It is a good divine that follows his own instructions; I can easier teach twenty what were good to be done, than to be one of the twenty to follow mine own teaching. -Shakespeare *(The Merchant of Venice)*

Chapter 9

Conclusions

To coordinate well in a given situation, computing agents such as problem solvers need to plan their interactions rather than acting independently. To plan their interactions, the nodes must first recognize when they should interact—when they are pursuing common or compatible goals. In turn, this means that nodes must communicate about their local goals and plans to form this more global view. But nodes must form local plans before they can exchange them, and must know what goals they should plan to achieve. In many domains, nodes are not given these goals, but instead must recognize goals based on their current situation.

In this research, we have developed an approach for coordination. To implement this approach, we have had to first provide nodes with the ability to recognize local goals and plans, and then give them additional capabilities for identifying more global goals and for building more global plans. In the next section, we summarize the research we have done. After this, we return to the research issues introduced in Chapter 1, and discuss how our approach, as exemplified in the implemented mechanisms, addresses these issues. We then summarize the contributions of the research. Finally, we point out future research directions

to refine and extend this work.

9.1 Summary

The overall objective of this research has been to develop, implement, and evaluate an approach for coordinating cooperating problem solvers, and this was done in two major phases. In the first phase (Chapters 3–5), the control mechanisms for an individual problem solver were modified to improve the problem solver's self-awareness of its goals and plans. A problem solver uses simplified domain knowledge to cheaply generate an abstract view of the problem situation so that it can recognize potential solutions and predict their long-term significance and cost. By reasoning about how the potential solutions are related, the problem solver plans actions that work toward one or more of them, and that at the same time generate information that helps resolve uncertainty about which of the potential solutions to pursue. Because the results of earlier actions may affect how (and whether) a plan should be pursued, the planning mechanisms interleave plan generation, monitoring, and repair with plan execution, leading to more versatile planning where actions to achieve problem solving goals and actions to resolve uncertainty are integrated into a single plan. The experimental results indicate that this self-awareness improves the problem solver's control decisions, and the additional control overhead (computation and storage) is generally compensated for by resources saved in problem solving since fewer incorrect actions are taken.

The second major phase of the research (Chapters 6–8) extended the local planning mechanisms so that problem solvers can communicate summaries of their own plans and thus can increase each other's network awareness. With this additional knowledge, problem solvers identify when they are working on parts of some larger (partial global) goal, and represent their combined efforts as a partial global plan. How information is exchanged and processed depends on the meta-level organization, which indicates the coordination roles of the different problem solvers. By following this organization, one or more problem solvers forms partial global plans and reasons about how the cooperating problem solvers could alter their actions to improve coordination. These partial global plans are distributed among the problem solvers, which individually change their local plans to reflect the more global view. Since local plans may change dynamically, coordination

information is propagated around the network over time, and the problem solvers must cooperate despite possibly inconsistent partial global plans. An important aspect of the approach is that it emphasizes planning *during* problem solving (while problem and network characteristics are changing) and allows problem solvers to cooperate well enough (but not necessarily optimally) to form acceptable solutions. Our experimental evaluation showed the benefits and costs (in overhead) of the approach, and especially how it allows coordination in a changing world. The upshot of the experiments is that partial global planning represents a flexible, versatile, and practical approach to dynamic coordination, but that indiscriminate application of the mechanisms can result in excessive overhead in certain complex situations where the nodes cannot distinguish a small set of possible goals.

The research thus represents a substantial but preliminary investigation into the use of local and partial global planning for coordinating problem solvers. Although the costs and limitations of the mechanisms indicate the need for further research (especially developing more selective criteria for when to apply the various mechanisms), the benefits, flexibility, and versatility of the mechanisms indicates the promise of this approach.

9.2 Research Issues Revisited

In Chapter 1, we introduced several important research issues. Among these are developing a framework for cooperation that allows problem solvers to individually plan their local activities, to communicate about their plans to form more global plans, to achieve different goals of cooperation, and to cooperate in different styles depending on the characteristics of the network and the demands of the environment. In this section, we return to those issues and discuss how the approach detailed in the intervening chapters has addressed them.

9.2.1 Sophisticated Local Control

Three chapters (Chapters 3–5) were devoted to describing and evaluating the mechanisms developed for sophisticated local control of a problem solving node. In Chapter 1, we noted that in dynamically changing domains a problem solver needs to interpret its current state (data) to identify potential long-term goals.

The mechanisms outlined in Chapter 3 satisfy these needs by allowing a node to apply simplified domain knowledge to its data to build a clustering hierarchy. By using simplified knowledge, the mechanisms minimize the expense of forming this view. However, the view is sufficient for identifying long-term relationships in the data—relationships that indicate potential solutions and their cost and significance. By forming a view uniquely suitable for making control decisions (as opposed to forming actual solutions), the clustering mechanisms allow the node to identify long-term goals. The clustering mechanisms also permit new information (data) to be incorporated into the hierarchy so that dynamic changes to the problem situation lead to changes in control.

Once long-term goals have been identified, local control must plan local actions to achieve those goals. In Chapter 4, we described planning mechanisms that work in a dynamically changing environment. The local planner stresses incremental planning, so that details for the more distant future are not planned until earlier actions have been taken. By interleaving planning and execution, a node avoids unnecessary control overhead on planning for future situations that may never arise. The planner modifies plans when the long-term goals change, and it monitors and repairs plans so that actions that fail can be replaced with alternative actions whenever possible. The plans permit predictions about what results will be formed and when, so that if deadlines are imposed on the node it can modify its actions to meet them. These predictions are also crucial to anticipating interactions when the node cooperates with other nodes.

9.2.2 Goals of Cooperation

In Chapter 1, we listed a number of possible goals of cooperation that a network of problem solvers might possess. In Chapters 6–8, we described and evaluated the partial global planning mechanisms that are intended to provide sufficient flexibility so that nodes might cooperate to achieve these various goals. To understand how well these mechanisms succeed in meeting this objective, we address the different goals of cooperation and briefly indicate how each can be achieved through the partial global planning mechanisms.

- *To increase the solution creation rate by forming subsolutions in parallel.* The inherent distribution of tasks among nodes allows them to work on

tasks in parallel. When tasks are not well distributed initially, the new mechanisms allow computation load to be balanced in any of several ways: the meta-level organization can give additional coordination responsibilities to an underutilized node by making it form PGPs for others; the PGPs can assign integration tasks to an underutilized node; or the task-passing mechanisms can move data to be processed from an overburdened node to an underutilized node.

- *To minimize the time agents must wait for results from each other by coordinating activity.* The PGPlanner reorders the nodes' activities so that nodes generate more globally important results earlier. In addition, the PGPlanner explicitly represents how nodes should interact to form overall solutions: which node provides which partial solution and when. Not only does this allow nodes to exchange important partial solutions in a timely manner, but it also lets them recognize when they can delay work on one result in favor of another so that they cooperate more effectively (as shown in some experiments in experiment set 8.8 in Chapter 8).

- *To improve the overall problem solving by permitting agents to exchange predictive information.* Among the cooperation-parameters is the importance of providing predictive information, and the PGPlanner uses this parameter when reordering activities so that activities that are likely to generate predictive results are performed earlier. The PGPlanner also uses this information when deciding what results should be exchanged between nodes so that predictive results are transmitted. Many of the experiments in Chapter 8, and in particular those involving environments where one of the nodes has a faulty sensor, had providing predictive information as one of the goals of cooperation.

- *To increase the probability that a solution will be found despite agent failures by assigning important tasks to multiple agents.* The PGPlanning mechanisms allow increased reliability in several ways. First, if reliability is more important than avoiding redundancy, the PGPlanner allows nodes to generate duplicate results. Second, the meta-level organization allows decentralization of PGPlanning so that nodes can coordinate despite agent failures. Third, the integration-redundancy cooperation-parameter indicates

how many nodes should have responsibility for integrating results, and if a result is sent to several nodes for integration, that integration is more likely to be done despite node failures. Some experiments in experiment set 8.5 (Chapter 8) illustrated how these mechanisms allow a network to find solutions even when an integrating node fails.

- *To improve the use of computing resources by allowing agents to exchange tasks.* The PGPlanning mechanisms provide task-passing capabilities. When some nodes are underutilized while others are overburdened (and represent bottlenecks in overall network problem solving), the nodes can exchange node-plans and PGPs to negotiate about the possible transfer of tasks. Experiment set 8.7 in Chapter 8 illustrated these mechanisms.

- *To improve the use of individual agent expertise by allowing agents to exchange tasks.* The PGPlanning mechanisms allow nodes to represent each other's expertise as part of their node-models. As shown in some experiments in experiment set 8.5 (Chapter 8), the PGPlanner uses this information when deciding where results should be sent for integration: nodes that are "experts" at integration (for example, they can combine results fastest) are chosen. The PGPlanning framework could also support having nodes find possible task swaps so that a node that could do a task might pass it to a node that could do it better, but the mechanisms to do this swapping have not yet been incorporated in the implementation.

- *To reduce the amount of unnecessary duplication of effort by letting agents recognize and avoid useless redundant activities.* The importance of avoiding redundancy is represented in the cooperation-parameters, and the PGPlanner uses this information when reordering activities: activities that could be done by other nodes are typically postponed to avoid redundancy. The PGPlanner also considers redundancy avoidance when deciding how the overall solution will be constructed. Most of the experiments in Chapter 8 had avoiding redundancy as one of the goals of cooperation.

- *To increase the confidence of a (sub)solution by having agents verify each other's results through rederivation using their potentially different expertise.* The cooperation-parameters specify the relative importance of verification

(reliability) versus redundancy avoidance. If it is more important to verify results than to avoid redundancy, the PGPlanner reorders node activities so that nodes concentrate on their overlapping data first. In addition, the PGPlanner forms expectations about solution construction and about communication that reflect the desire to generate and exchange information in overlapping areas for verification.[1]

- *To increase the variety of solutions by allowing agents to form local solutions without being overly influenced by other agents.* The cooperation parameters include parameters affecting the relative importance of independence. When the independence is high, the cost of changing local plans based on global views is high and a node is much less likely to modify its local plans based on other agents. By striking a balance between independence and the other factors, the network allows nodes to balance their local desires against more global pressures.[2]

- *To reduce the communication resource usage by being more selective about what messages are exchanged.* Because it develops a view about exactly which results should be combined and where, the PGPlanner can dramatically reduce the number of messages about partial results passed between nodes, as illustrated in the experiments in Chapter 8. As was also seen, however, the exchange of messages about node-plans and PGPs adds to the the communication resource usage, and future research must be directed toward making more selective meta-level communication decisions.

The mechanisms thus allow each of the goals of cooperation to be achieved, at least to some extent. Because the importance of achieving conflicting goals

[1] Unfortunately, the current belief calculations in the DVMT's knowledge sources are unable to to increase a node's confidence in corroborated results that have been independently generated by several nodes. Although the PGPlanner can plan for this goal of cooperation (by increasing the reliability and solution-construction-redundancy cooperation-parameters), this capability was not experimentally tested.

[2] The experiments studied emphasized issues in cooperation—particularly in avoiding redundancy and providing predictive information—and independence was not stressed. In environments where, for example, a node locally plans to develop less ambiguous data before more confused data, independence is more important. Because PGPs are based on a more abstract view that has less precise estimates of relative costs (since cluster information is unavailable to recipient nodes), they may reorder local activities so that the node may be expected to work on more confused data first. If nodes have greater independence, the PGP is less likely to reorder the activities.

of cooperation (such as reliability versus redundancy avoidance, or independence versus providing predictive information) is determined by the cooperation-parameters, the PGPlanner has the versatility to achieve different goals of cooperation depending on the specified needs of the network.

9.2.3 Styles of Cooperation

The style of cooperation between nodes—whether centralized or decentralized, hierarchical or lateral—is very situation dependent. A style that works well for one situation may be bad in another situation. For example, when minimizing network computation overhead is desired, a centralized style of cooperation where a single node coordinates the entire network may be the best. However, when reliability is at a premium, a more decentralized style of cooperation where several nodes build PGPs may be better despite the wasted effort when they duplicate each other's reasoning. An important objective of this research was to develop a single, flexible set of mechanisms that permit cooperation in any number of styles.

In the PGPlanning mechanisms, the style of cooperation is dictated by the meta-level organization. The PGPlanner uses this specification of the coordination roles and responsibilities of nodes to determine, for instance, what coordination information (node-plans and PGPs) to send and where, how to treat received coordination information (how credible it is), and what nodes it is responsible for coordinating. In the experiments in Chapter 8, we examined a number of different meta-level organizations. Besides the fully centralized (a single coordinator) and fully decentralized (the broadcast metal-level organization where every node builds PGPs), we also explored meta-level organizations where nodes would pass around PGPs (modify the PGP with the local view and then pass it on), where nodes pass node-plans around in a ring, and where a multi-level hierarchy is used. Many more meta-level organizations are possible, especially in larger, more complex networks, and an important advantage of the PGPlanning mechanisms is that they are versatile enough to work in wide variety of styles of cooperation.

9.2.4 Predictability and Responsiveness

The experiments in Chapter 8 briefly explored some of the tradeoffs between having nodes be predictable versus having them be responsive. The PGPlanning mechanisms do not dictate a particular balance between these conflicting demands because the decision about the relative costs and benefits of predictability and responsiveness depend on the network characteristics. As shown in the experiments, increasing predictability (by enlarging the time-cushion in which nodes' plans can deviate from expectations without the node responding) reduces the overhead for coordination (nodes modify their node-plans and PGPs less often) but can also lead to poorer network problem solving. On the other hand, increasing responsiveness can improve network problem solving but at the cost of more coordination overhead, and being too responsive can sometimes degrade network performance because PGPs change too rapidly for nodes to develop similar views of coordination. Where the line between when to be predictable and when to be responsive should be drawn is thus a difficult and situation dependent question, but the PGPlanning mechanisms allow predictability and responsiveness to be balanced flexibly within a single coordination framework.

9.3 Contributions

Distributed problem solving, and more generally distributed artificial intelligence, is an important meeting ground between artificial intelligence and distributed computing. As the complexity of distributed computing applications grows, so does the need to control the distributed systems more intelligently. As artificially intelligent systems are introduced into environments where they interact with other intelligent agents (whether artificial or natural), they need to reason about and plan coordinated actions and interactions with those other agents. In short, we contend (and we are not alone [Nilsson, 1980]) that distributed artificial intelligence will become an increasingly important area for study in both the AI and the distributed computing communities.

As an investigation into many important issues in cooperation in a distributed problem solving network, this research contributes to extending our limited view of the field. By proposing, developing, implementing, and evaluating an approach to coordination, this research has built on past efforts and has gone beyond.

Although it explores a limited part of the larger issues in distributed AI and distributed computing systems, it provides one more reference point from which to base future exploration.

We have laid down a conceptual foundation that views control of both local and group activity as a planning task, where planning is worthwhile only insofar as it improves the system's ability to achieve its domain goals. Our approach is therefore *not* a distributed planning system, because nodes may solve their domain problems without ever forming a complete plan, particularly in dynamic environments where they should plan incrementally to avoid detailing actions that may never come to pass. Instead, nodes plan actions and interactions as domain problem solving proceeds. They exchange only relevant information about their local plans, determine when changes to local plans should be communicated, and use whatever information they have locally to individually decide how best to cooperate at any given time.

Conceptually, therefore, this research has explored the types of knowledge and reasoning methods that are needed for coordination in a dynamic world. An important conceptual contribution is our view of coordination as a partial global planning process. Whereas previous approaches to coordination developed separate techniques for coordination depending on what was to be gained through cooperation, our new approach provides a unified framework where diverse goals and styles of cooperation can be achieved. Using this framework and a variety of meta-level organizations, we have explored how different styles of cooperation affect network performance and control overhead. Moreover, we have examined how nodes can dynamically exploit available resources and expertise, and how they can develop partial global plans that are more robust when some nodes may fail.

Our research has also made contributions to understanding control in complex problem solvers. We have recognized the need to apply approximate processing to form a representation uniquely suitable for control so that control decisions can be based on more long-term views. We have also examined the role of incremental planning as a technique to control problem solving when long-term goals are uncertain and dynamic. As artificially intelligent systems are applied to more complex domains, we expect that the role of approximate processing and incremental planning in control will become increasingly important.

Since our research methodology focuses on implementing and evaluating our approach, our research also makes several practical contributions. One of these contributions is simply experience. Although many approaches to coordination have been proposed in the literature, only a handful have been implemented. Implementation points out issues that may be inapparent or ignored in more speculative treatments. By implementing our approach, we came upon many issues that we had not anticipated, and by sharing our experience in the previous chapters, we hope to provide an example that others can build on. Implementation also leads to evaluation, and another important contribution of this research has been exploring the practicality of the approach: it does improve coordination as expected, but the costs for better control must also be considered, especially in complex and dynamic situations.

Finally, an important contribution of this research is that it raises new questions about cooperation and provides an implementation so that future research can build upon it both conceptually and experimentally. By opening new avenues for exploration into planning, control, coordination, communication, and organization, this research acts as a springboard for diving deeper into distributed AI and distributed computing. Because it provides an implementation in which to test new ideas, this work also allows subsequent researchers to not only theorize about issues in cooperation, but to experiment with them as well.

9.4 Directions for Future Research

We conclude by speculating on several directions for future research in control of problem solving and distributed problem solving.

9.4.1 Improving the Existing Mechanisms

In our implementation of local and partial global planning, we have striven toward developing enough of the mechanisms to permit a reasonable evaluation of the approach. Along the way, we made some simplifications and assumptions that allowed us to press on, and future research efforts will go back and improve the mechanisms. In this section, we briefly mention some of these areas for improvement.

Several combinatorial issues need to be addressed. As mentioned in Chapter

6, an increase in the number of nodes (hence node-plans) can cause an explosive increase in the number of possible PGPs (combinations of node-plans). Similarly, as the number of concurrent activities in a PGP increases, so does the combinatorics of finding a suitable ordering. An answer to these combinatoric issues is to introduce PGPlanning mechanisms that allow nodes to summarize the goals and activities of a group of nodes as if the group were acting as a single node. In this way, nodes could coordinate individual actions in small groups, then communicate about their overall group-plans. This group-plan would include a plan-activity-map indicating a sequence of group activities: unlike a PGP plan-activity-map that represents the concurrent activities of individual nodes, the group-plan plan-activity-map would represent a single sequence of activities being taken by the overall group. A received group-plan could thus be treated much like a single node-plan, and encapsulates several node-plans. By exchanging a smaller number of group-plans instead of the larger number of node-plans that they represent, therefore, the PGPlanning mechanisms can reduce the combinatorics of finding and reasoning about PGPs.

Simulation of control costs should be improved. Because we could not assign simulated costs to the various control activities, we could not experimentally address certain questions, such as what would happen if control was twice as expensive as problem solving. We need to better understand all the activities of a node, both problem solving and control, and provide the experimenter with the ability to simulate different costs for each of those activities. Moreover, by simulating time spent on control activities, we can identify how control affects the timing of coordination. Adding these capabilities is a difficult research problem because they require a deep understanding of what control activities occur, how they are related, and where and how the costs should be simulated.

The planning mechanisms could be modified to integrate local and partial global planning even more completely. In the current implementation, each node builds its local plans based on its local view and then the partial global planning mechanisms modify these plans to coordinate better. In the future, we would like to explore how local planning could be based in part on the current knowledge about partial global plans. By using this knowledge, a node could build local plans that are less likely to need substantial modifications later on, so that nodes could intentionally build more stable plans.

The way that the goals of cooperation and the meta-level organizations are represented could also be improved upon. In the initial implementation, the goals of cooperation were represented numerically to simplify decisions about the importance of each. Symbolic representations may better capture the intent and more subtle interactions between these goals. The meta-level organization is a fairly amorphous collection of information about how nodes should interact to form PGPs. We hope that future research will lead to a more structured view of this information, so that the design (and re-design) of the meta-level organization can be automated.

A node's representation of the network could also be improved. The attentive reader will have noticed that some information in a node-plan (such as the activity-per-time) is currently unused. Our initial implementation divided local activities for problem solving into two broad classes: synthesis and integration. When reasoning about each other's abilities to assign tasks, nodes only considered activities in these broad classes. In future implementations, nodes should have more specific knowledge about the quality of each other's KSs, so that when they communicate about particular activities (to apply some sequence of KSs) the nodes can more completely reason about their aptitudes for performing those activities.

Another important open research issue is in monitoring and repairing PGPs. Although the local planning mechanisms allow nodes to monitor and repair their plans, the PGPlanning mechanisms do not yet have this ability. Instead, the PGPlanner has the ability to form "forgiving" PGPs—by adding redundancy to PGPs in case of node failures, for instance. In the future, we would like the PGPlanner to monitor the progress of PGPs to detect problems, such as when some node fails to fulfill its part of the plan, when nodes take longer or shorter than expected to build results, or when results that should have been received have failed to arrive (since the communication KSs do not explicitly acknowledge message receptions). By detecting when expected results fail to arrive, for example, the PGPlanner can not only modify the PGP to counter this problem but can also retrigger the node that built the result to send it again, and the PGPlanner thus allows nodes to coordinate better when the communication channels are errorful.

The task-passing mechanisms can be extended to promote task-passing in

a wider range of situations. In particular, task-passing should occur not only to move tasks from overburdened to underutilized nodes, but also to let nodes negotiate in more complex ways such as swapping tasks to exploit their expertise. Further research will be needed to determine how nodes can recognize potential task swaps and to understand the costs and benefits of finding and invoking these swaps. We would also like to generalize the mechanisms to decompose larger tasks into an arbitrary number of subtasks (rather than just roughly breaking the task in half) to promote more parallelism. Finally, sometimes it may be more advantageous for nodes to move resources or expertise rather than tasks, and we hope to increase the flexibility of the task-passing mechanisms so that nodes can recognize and respond to these situations.[3]

Although the PGPlanning mechanisms improve domain-level communication because nodes have a better view of what partial results other nodes could use, the mechanisms add to communication resource use because of the exchange of coordination information. Although the meta-level organization affects this communication to some extent, more sophisticated techniques are needed to enable nodes to decide which coordination information is important enough to warrant sending to other nodes. We also need a better model of the network connections to simulate contention for communication channels so that we are forced to develop mechanisms that can be more or less selective about what is transmitted.

Finally, the mechanisms have been implemented in a local dialect of Lisp. The DVMT is currently being reimplemented in Commonlisp, and once that reimplementation is complete, the local and partial global planning mechanisms will also be reimplemented. In the course of reimplementation, it is hoped that many of the improvements previously described, as well as improvements to efficiency and modularity of the code, can be incorporated. The PGPlanning mechanisms evolved over time, and the current implementation bears the marks of this evolution (such as obsolete functions and functions whose activities have changed). Moreover, some of the more general, domain-independent aspects of the mechanisms were not initially obvious, and as a result many of the mechanisms do not make a clear distinction between the domain-dependent and the domain-independent aspects of local and partial global planning. We hope to recode the

[3]The ability to reassign resources and expertise (KSs) is also an important aspect of organizational self-design.

mechanisms to reflect our current view, and in particular to build a more generic framework to simplify the process of implementing our mechanisms in other domains, much like the reimplementation of the underlying blackboard architecture has led to the development of GBB (a generic blackboard development system) [Corkill *et al.*, 1986].

9.4.2 Goal Processing

The planning mechanisms rely on goal processing. The planner uses a high-level, long-term view of the problem situation to find suitable areas to work. This view has insufficient detail for determining exactly what KSs should be applied to what data, however. The goal processor has the more detailed view needed for forming KSIs, but the indiscriminate application of goal processing adds substantial control overhead. Moreover, as the number of ways that the goal processor can form and manipulate goals to achieve more precise control increases [Hernandez *et al.*, 1987], this overhead grows substantially.

Control of goal processing is thus crucial as the sophistication of goal processing increases. The planner helps control goal processing by selectively invoking goal processing mechanisms when it must resolve uncertainty. A future research direction is to better understand the interactions between detailed goal processing and more abstract planning, and to better integrate them into a multi-level control paradigm.

9.4.3 Real-Time Problem Solving

When problem solving is not planned, the problem solver is unable to predict when results will be formed. The planning mechanisms and their ability to predict how and when results will be formed has opened new areas for research in real-time problem solving [Lesser and Pavlin, 1987]. An important future research direction will be to further explore how a problem solver can modify its plans so that it balances conflicting needs of forming good results and of meeting deadlines. In addition, we also hope to explore techniques for forming meta-level organizations that better meet deadlines by reducing redundancy, and understanding the costs of such organizations in terms of reliability.

9.4.4 Node Parallelism

Throughout this research, we have concentrated on sequences of activities: sequences of activities to form solutions, and sequences of control activities to generate plans and PGPs that affect local decisions. We have assumed that a node is a serial system, and that parallelism arises from the concurrent activities of nodes. However, there is no reason why an individual problem solver must work serially. If each node were a multi-processor, then it could exploit the parallelism inherent in its own tasks as well. When concurrency is possible within nodes as well as between them, questions arise as to how local parallelism can be exploited within the partial global planning framework.[4]

One way of exploiting parallelism would be to let control activities occur in parallel with problem solving activities. In the overall framework of our mechanisms (recall Figure 1 in Chapter 1), there is no reason why control and problem solving cannot occur in parallel. The local problem solver invokes the most highly-rated KSI available, and need not wait for the planner. Concurrently, the planner builds and maintains plans. It pursues the best local plan by modifying the rating of the appropriate KSI (which thus affects local problem solving), and need not wait for the PGPlanner. Concurrently with both the planner and the problem solver, the PGPlanner builds and maintains PGPs, and pursues the best PGP by modifying the appropriate local plan (so that it is the most highly rated). Problem solving and control at different levels can thus occur asynchronously, although some synchronization might lead to better overall performance (for example, if the problem solver sometimes waits for the planner to monitor and repair plans). An area of future research is to better understand how control and problem solving can occur in parallel, and the costs and benefits of this parallelism.

Another way of exploiting parallelism would be to invoke several problem solving actions concurrently. In this case, an important consideration is how to decide what actions should occur in parallel and how this affects planning. At one level, each processor could be assigned a separate plan, so that the problem solver pursues several plans concurrently but each of these plans serially. At

[4]Questions also arise as to whether the control mechanisms developed to coordinate concurrent activities between nodes could be used to coordinate internal parallelism as well, and we hope to explore this possibility. However, the more tightly coupled nature of multi-processing (compared to distributed processing) does significantly change the coordination problem.

a finer-grained level, each processor could be assigned a different intermediate-goal for the same plan. Because parts of this plan are achieved concurrently, the overall results of the plan are achieved faster. However, the sequences of actions to achieve intermediate-goals are still serial. At an even finer-grained level, each processor could be assigned a different KSI for the same intermediate-goal, to increase the rate at which the intermediate-goal is achieved. Parallelism introduced at any of these levels can cause difficulties. Since the planner decides which action, intermediate-goal, or plan to pursue next based on previous results, the scheduling decisions for each processor might be poorer. Thus, mechanisms that use parallelism effectively must choose the granularity of concurrent tasks carefully, and this is an important future research topic.

9.4.5 Ranking Alternatives for Satisficing

Decisions about what PGPs to pursue are made in the current implementation by calculating numeric ratings and pursuing the most highly-rated PGP. The network thus concentrates on developing the best result first, then the second best, and so on. In the future, we would like to develop techniques for reasoning about several PGPs at once. For example, if the network could form several slightly inferior results in the time it would take to form the most highly-rated result, which should it do?

The nodes cannot expect to form optimal plans and PGPs, both because the costs of finding optimal plans and PGPs can be prohibitively large, and because the environment is dynamic and uncertain. Our mechanisms thus strive for acceptable (satisficing [March and Simon, 1958]) control rather than optimal control. To make better satisficing decisions, the local and partial global planning mechanisms should consider other factors such as balancing quality and quantity of results. Instead of always giving preference to the most highly-rated PGP, for example, our mechanisms could be extended to pursue a satisfactory set of PGPs that meet some other criteria.

9.4.6 Communication

As mentioned above, an important area for improvements to the mechanisms is in making meta-level communication more selective. A future research direction

of particular interest is in balancing communication with computation. For example, a node need not receive information from another node if it could infer that information locally, but making such inferences requires computation. In our current implementation, we concentrated on using communication resources: a node monitors and modifies only its own plans, and communicates about important changes. In this way, computation is reduced since nodes do not reason about how others might change their plans, but communication is increased since they depend on learning about changes from other nodes. In the future, we hope to extend the mechanisms so that nodes can reason about probable changes in each other's plans. For example, if a node finds that it underestimated the time needs for a local plan, it might surmise that other nodes made similar mistakes if it assumes that all nodes form predictions in similar ways. With this capability, nodes no longer need to communicate as much because they locally anticipate what the messages would convey. However, making these inferences can be time consuming, and the savings in communication should be weighed against the additional computation overhead to find a suitable balance.

We would also like to explore issues in more errorful communication channels. As mentioned above, the PGPs provide a basis for recognizing when domain-level messages have been lost: a node that expects to receive a result from another node based on a PGP can signal that other node if the result fails to arrive. Because they know what domain-level information they should receive, nodes can trigger retransmission when these messages are lost. However, PGPs do not represent meta-level communication, and the nodes may be unable to detect when meta-level messages are lost. Although the PGPlanning mechanisms allow nodes to tolerate a certain amount of inconsistency in their PGPs, issues in how much inconsistency (caused by message losses) can be tolerated should be studied in the future.

9.4.7 Cooperation Among Heterogeneous Systems

The mechanisms that we have developed allow for some heterogeneity in the network (nodes may have different expertise) but they do assume that nodes have much in common in terms of their architecture and control structure. An area for future research concerns how different nodes can be before coordination becomes impossible. To coordinate dynamically, nodes must have substantial

common knowledge because they must be able to exchange plans and interpret each other's plans. They also need to recognize relationships among their goals. Finally, to decide how to modify local actions and interactions to cooperate with others, a node must have expectations about how its actions and interactions affect others.

Coordination is thus not simply a layer of control that is independent of local control. On the contrary, mechanisms that coordinate nodes must be familiar with the local control mechanisms of the nodes so that coordination can affect local control. Without this familiarity, coordination cannot affect local decisions, so nodes cannot cooperate intentionally. An important avenue for future research is thus the exploration of the intimate relationship between local control and coordination, how this relationship affects cooperation among more heterogeneous systems, and whether a more generic framework for cooperation is possible.

9.4.8 Fault Detection and Diagnosis

Local plans represent expectations about future node problem solving, and PGPs represent expectations about future network problem solving. When these expectations are violated, it indicates that a fault may have occurred. The local and partial global planning mechanisms thus open new avenues for fault detection in problem solving and distributed problem solving systems. Although the plans are tentative (they are based on simplified views of the data), the failure of several highly-rated plans may be indicative of some fault in the problem solver or network. The view afforded by plans thus provides a starting place for detecting faults.

By analyzing the failed plans, moreover, the source of the failure might be found. Perhaps they all failed for similar reasons, such as while invoking a particular KS, and this could be a starting point for further diagnosis. The relationships between the failed plans and the information about these plans thus provide a view for preliminary diagnosis. In essence, the plans represent models of expected system behavior, and by comparing the actual behavior with these plans, initial steps toward diagnosis can be taken (although a more detailed model may be needed to completely isolate the fault [Hudlicka, 1986]). A direction for future research is thus to explore how plans aid in the detection and diagnosis of faults, including what happens when the fault lies in the planner rather than in the

problem solver.

9.4.9 Understanding Problem Solving Behavior

In a complex problem solver, it is often difficult to develop a view of problem solving behavior: the myriad actions taken seem to have little in common so that understanding the larger implications and goals of actions is difficult [Pavlin and Corkill, 1984]. The local and partial global planning mechanisms help alleviate this difficulty because they group actions together into plans and plans together into PGPs. Using plans and PGPs as a frame of reference, the experimenter can more easily grasp the intentions of the problem solvers. A future research direction is to develop techniques that exploit this information to generate high-level explanations of local and network behavior.

9.4.10 Organizations

A crucial assumption made throughout this research has been that, somehow, the nodes have been organized both for problem solving and for coordination. By using the static domain-level and meta-level organizations, the planning and PGPlanning mechanisms were able to dynamically coordinate nodes to work as an effective team in a particular situation. An important area for future research is developing mechanisms for forming organizations, and this research will benefit from the planning and PGPlanning mechanisms.

PGPlanning provides a basis for detecting when nodes are poorly organized (or not organized at all). For example, if the network is performing excessive task-passing, if results are being integrated by some nodes when other nodes are better suited (have more resources or expertise), or if a large proportion of time is being spent on coordination, it might be appropriate to reorganize. The view of the network provided by the network-model can also help a node propose likely new organizations. Thus, although the local and partial global planning mechanisms do not perform reorganization, they may provide useful information to some future organizational self-design component.

9.4.11 Meta-Level Control

By experimentally evaluating our new mechanisms, we recognized that they do indeed improve control decisions. However, making better control decisions introduces overhead, and in particularly complex or ambiguous situations the increase in overhead might not justify the improvement to control. To balance the time spent on control with the time spent on actual problem solving, a node needs meta-level control (for control of control). Meta-level control examines the current situation and decides what if any control mechanisms should be applied, just like control decides what actions should be taken. An important future research direction is to build meta-level control mechanisms, perhaps based on the blackboard model of control developed by Hayes-Roth [Hayes-Roth, 1985].

Of course, once we have meta-level control, then we need meta-meta-level control to control it, and the need for higher level control mechanisms can go on indefinitely. Until further work is done on these problems, it is unclear when (if ever) a top-level of control can be reached, but it is our suspicion that control decisions become less complex and less dynamic at higher levels so that eventually a level is reached where the information is so abstract and static that relatively few options are available and decisionmaking is simplified.

9.4.12 Other Applications

The detailed description of how our mechanisms have been implemented serves not only as a basis for evaluation but also an exemplar for other applications. When we generalized the different mechanisms in the preceding chapters, we indicated other application domains that might implement versions of the same mechanisms. However, the emphasis of this research has been on building a complete implementation in one domain so that we could evaluate the promise of our approach. Our evaluation has shown that our approach is indeed a practical means for improving coordination among cooperating problem solvers in our domain. In the future, we hope to use our experience in this domain to explore how our approach works in other domains like those outlined in the earlier chapters, and we would like to encourage others to apply similar mechanisms as well, especially in real (not simulated) networks. No doubt our mechanisms will need substantial modification before they can be more generally used for coordination

in a variety of domains, but our initial experiments show that our approach is practical and worth pursuing.

Appendix A

Implementation Information

The DVMT and all of the new mechanisms for local and partial global planning, except where noted, have been implemented in a local dialect of Lisp called CLisp. There are roughly 20,000 lines of code, including comments, and the experiments were run on a VAX11/750.

The DVMT is in the process of being reimplemented in CommonLisp on TI Explorers. Once that reimplementation is complete, the local and partial global planning mechanisms will similarly be recoded to run on lisp machines, which we hope will decrease the time needs for simulations and will make the mechanisms more portable.

Appendix B

Annotated Trace

To give a flavor of the experimental output, we here present annotated excerpts from a DVMT trace. The experimental situation is environment A (Figure 84) where nodes can perform partial global planning with a broadcast meta-level organization. This corresponds to the experiment summarized in Figure 85 and as E8.2.1 in Table 8 (also see E8.1.4 in Table 7 and E8.4.2 in Table 10).

Each line of the trace represents a particular action during a node execution, and the trace shown only includes a selected set of such actions. A line begins with a character, where a *, a +, or a - denote whether the action is dealing with data that is true, false but consistent with the eventual solution, or false and inconsistent, respectively. The character # indicates that the action corresponds to planning activities. Also, DD-GOALS are data-driven goals (possible extensions or improvements to existing hypotheses) and GD-GOALS are goal-directed goals (subgoals of DD-GOALS). Node executions are separated by a string of $'s.

The fields associated with the trace lines:

```
# CREATED PLAN ----------> name-of-plan i-goal-order time-regions
                           vehicle-event-classes rating
# CREATED NODE PLAN -----> name-of-new-node-plan corresponding-local-plan
# UPDATED PLAN ----------> plan-name reason old-time-order --> new-time-order
                           <old-time-regions --> new-time-regions> rating
# UPDATED NODE PLAN -----> node-plan-name plan-name
# PGP FORMED ------------> pgp-name list-of-participating-node-plans
# PGP REORDERED ---------> pgp-name
# PLAN QUEUE ------------> queue-of-plan-and-rating-pairs
# PGP INVOKED -----------> pgp-name time-regions
                           list-of-participating-node-plans
# PLAN INVOKED ----------> plan-name i-goal-order time-regions rating
# PLANNER GOALING -------> hyps-for-goal-processing direction-for-goaling
# PLANNING SHORT TERM ---> plan-name
# SENT PLAN -------------> node-plan-name corresponding-local-plan-name to-node
# RECEIVED PLAN ---------> name-of-plan-sent node-that-sent-plan
* INVOKED KSI -----------> ksi-name times-ksi-was-rated ks-to-invoke
                           current-time stimulus-goals stimulus-hyps rating
* CREATED HYP -----------> hypothesis-name bb-level time-locations
                           event-class belief
* CREATED DD GOAL -------> goal-name bb-level active-time-regions
                           inactive-time-regions event-classes stimulus-hyps
                           rating
* INSTANTIATED KSI ------> ksi-name ks-to-invoke stimulus-goals stimulus-hyps
                           <goal-directed-rating data-directed-rating>
                           overall-rating
* SENT HYP --------------> to-node time-it-will-be-received hyp-name bb-level
                           time-locations event-class {belief}
* RECEIVED HYP ----------> from-node time-it-is-received hyp-name bb-level
                           time-locations event-class {belief}
```

```
  ENVIRONMENT: Abroadcast
  STARTED:     22-MAY-1987 17:57:59.13
  KERNEL DATE: 22-MAY-1987 17:56:30.79
  ************************************************************************
;
; over times 1--16, nodes are incorporating sensor data.
;

 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  Executing Node 1 ------ Inv Ksis 15 ------ Time Frame 15 ------ Node Time 16

        ; node execution for node 1, which has invoked 15 KSIs,
        ; having incorporated data for sensed time frames 1--15, and at node
        ; time 16

        ; the node builds the clustering hierarchy, generates local plans,
        ; and builds node-plans for the local plans
        ; there are four local plans because there are two long-term vehicle
        ; tracks and two vehicle-event-classes for each (where one of those
        ; vehicle-event-classes has much more likelihood).

# CREATED NODE PLAN -----> node-plan:01:001 plan:01:002
# CREATED PLAN ----------> plan:01:002 (4 5 6 7 8 9 10 11 12) ((4 ((4 10 4 10)))
                           (5 ((5 11 5 11))) (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                           (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                           (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18)))) (2) 338
# CREATED NODE PLAN -----> node-plan:01:002 plan:01:001
# CREATED PLAN ----------> plan:01:001 (4 5 6 7 8 9 10 11 12) ((4 ((4 10 4 10)))
                           (5 ((5 11 5 11))) (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                           (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                           (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18)))) (1) 2796
# CREATED NODE PLAN -----> node-plan:01:003 plan:01:004
# CREATED PLAN ----------> plan:01:004 (1 2 3 4 5) ((1 ((5 21 5 21)))
                           (2 ((4 20 4 20))) (3 ((3 19 3 19))) (4 ((2 20 2 20)))
                           (5 ((1 19 1 19)))) (2) 442
# CREATED NODE PLAN -----> node-plan:01:004 plan:01:003
# CREATED PLAN ----------> plan:01:003 (1 2 3 4 5) ((1 ((5 21 5 21)))
                           (2 ((4 20 4 20))) (3 ((3 19 3 19))) (4 ((2 20 2 20)))
                           (5 ((1 19 1 19)))) (1) 2942

        ; after plans and node-plans are generated, the pgplanner builds PGPs
        ; for each PGG.  It forms any PGPs (plan-activity-maps,
        ; solution-construction-graphs) until it finds the current PGP
```

```
# PGP FORMED ------------> pgp:01:001 (node-plan:01:004)
        ; this first PGP is the current one
# PLAN QUEUE ------------> ((plan:01:003 2942) (plan:01:001 2796)
                           (plan:01:004 442) (plan:01:002 338))
        ; the highest-rated PGP is invoked
# PGP INVOKED -----------> pgp:01:001 ((1 ((5 21 5 21))) (2 ((4 20 4 20)))
                           (3 ((3 19 3 19))) (4 ((2 20 2 20)))
                           (5 ((1 19 1 19)))) (node-plan:01:004)
        ; the local plan for this PGP is invoked
# PLAN INVOKED ----------> plan:01:003 (1 2 3 4 5) ((1 ((5 21 5 21)))
                           (2 ((4 20 4 20))) (3 ((3 19 3 19))) (4 ((2 20 2 20)))
                           (5 ((1 19 1 19)))) 2942
        ; and the node-plans for the highest-rated local plans are
        ; transmitted to the relevant nodes
# SENT PLAN -------------> node-plan:01:004 plan:01:003 2
# SENT PLAN -------------> node-plan:01:004 plan:01:003 3
# SENT PLAN -------------> node-plan:01:004 plan:01:003 4
# SENT PLAN -------------> node-plan:01:002 plan:01:001 2
# SENT PLAN -------------> node-plan:01:002 plan:01:001 3
# SENT PLAN -------------> node-plan:01:002 plan:01:001 4
        ; the KSI corresponding to the next action of the plan is invoked
- INVOKED KSI -----------> ksi:01:0016 (2) s:sl:gl 16 (g:01:0001) (h:01:0001
                           h:01:0002) {1700}
        ; and creates several hypotheses
- CREATED HYP -----------> h:01:0071 gl ((1 (5 21))) 2 {7500}
- CREATED HYP -----------> h:01:0072 gl ((1 (5 21))) 3 {4375}
- CREATED HYP -----------> h:01:0073 gl ((1 (5 21))) 1 {4375}
        ; one of these new hypotheses satisfies the expectations, and the
        ; planner determines that this hypothesis should be synthesized to
        ; the next bb-level
# PLANNER GOALING -------> (h:01:0071) synthesis
- CREATED DD GOAL -------> g:01:0043 vl ((1 (4 20 6 22))) nil (1) (h:01:0071)
                           {750}
        ; and a KSI is instantiated to achieve this goal
- INSTANTIATED KSI ------> ksi:01:0044 s:gl:vl (g:01:0043) (h:01:0071)
                           <750 3750> {1350}
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Executing Node 2 ------ Inv Ksis 15 ------ Time Frame 15 ------ Node Time 16

      ; node execution for node 2, which has invoked 15 KSIs,
      ; having incorporated data for sensed time frames 1--15, and at node
      ; time 16

      ; the node builds the clustering hierarchy, generates local plans,
      ; and builds node-plans for the local plans.  Because of its faulty
      ; sensor, node 2 finds a number of possible vehicle-event-classes
      ; equally likely for the long-term vehicle track, and so builds only
      ; a single plan to cover them all.

# CREATED NODE PLAN -----> node-plan:02:001 plan:02:001
# CREATED PLAN ----------> plan:02:001 (10 11 12 13 14 15) ((10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) (7 6 5 4 3 2 1) 2796
      ; a single PGP is made and its attributes are formed
# PGP FORMED ------------> pgp:02:001 (node-plan:02:001)
# PLAN QUEUE ------------> ((plan:02:001 2796))
# PGP INVOKED-----------> pgp:02:001 ((10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18))) (13 ((13 19 13 19))) (14 ((14
                           20 14 20))) (15 ((15 21 15 21)))) (node-plan:02:001)
# PLAN INVOKED ----------> plan:02:001 (10 11 12 13 14 15) ((10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) 2796
# SENT PLAN -------------> node-plan:02:001 plan:02:001 1
# SENT PLAN -------------> node-plan:02:001 plan:02:001 3
# SENT PLAN -------------> node-plan:02:001 plan:02:001 4
* INVOKED KSI -----------> ksi:02:0016 (11) s:sl:gl 16 (g:02:0001) (h:02:0001
                           h:02:0002) {1440}
* CREATED HYP -----------> h:02:0205 gl ((10 (10 16))) 2 {6400}
- CREATED HYP -----------> h:02:0206 gl ((10 (10 16))) 3 {3600}
- CREATED HYP -----------> h:02:0207 gl ((10 (10 16))) 1 {3600}
# PLANNER GOALING -------> (h:02:0205) synthesis
* CREATED DD GOAL -------> g:02:0085 vl ((10 (9 15 11 17))) nil (1)
                           (h:02:0205) {640}
* INSTANTIATED KSI ------> ksi:02:0094 s:gl:vl (g:02:0085) (h:02:0205)
                           <640 3200> {1152}
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Executing Node 3 ------ Inv Ksis 15 ------ Time Frame 15 ------ Node Time 16

      ; node execution for node 3, which has invoked 15 KSIs,
      ; having incorporated data for sensed time frames 1--15, and at node
      ; time 16

      ; the node builds the clustering hierarchy, generates local plans,
      ; and builds node-plans for the local plans.

# CREATED NODE PLAN -----> node-plan:03:001 plan:03:002
# CREATED PLAN ----------> plan:03:002 (1 2 3 4 5 6) ((1 ((1 7 1 7)))
                           (2 ((2 8 2 8))) (3 ((3 9 3 9))) (4 ((4 10 4 10)))
                           (5 ((5 11 5 11))) (6 ((6 12 6 12)))) (2) 335
# CREATED NODE PLAN -----> node-plan:03:002 plan:03:001
# CREATED PLAN ----------> plan:03:001 (1 2 3 4 5 6) ((1 ((1 7 1 7)))
                           (2 ((2 8 2 8))) (3 ((3 9 3 9))) (4 ((4 10 4 10)))
                           (5 ((5 11 5 11))) (6 ((6 12 6 12)))) (1) 2796
# PGP FORMED ------------> pgp:03:001 (node-plan:03:002)
# PLAN QUEUE ------------> ((plan:03:001 2796) (plan:03:002 335))
# PGP INVOKED-----------> pgp:03:001 ((1 ((1 7 1 7))) (2 ((2 8 2 8)))
                           (3 ((3 9 3 9))) (4 ((4 10 4 10))) (5 ((5 11 5 11)))
                           (6 ((6 12 6 12)))) (node-plan:03:002)
# PLAN INVOKED ----------> plan:03:001 (1 2 3 4 5 6) ((1 ((1 7 1 7)))
                           (2 ((2 8 2 8))) (3 ((3 9 3 9))) (4 ((4 10 4 10)))
                           (5 ((5 11 5 11))) (6 ((6 12 6 12)))) 2796
# SENT PLAN -------------> node-plan:03:002 plan:03:001 1
# SENT PLAN -------------> node-plan:03:002 plan:03:001 2
# SENT PLAN -------------> node-plan:03:002 plan:03:001 4
* INVOKED KSI -----------> ksi:03:0016 (2) s:sl:gl 16 (g:03:0001) (h:03:0001
                           h:03:0002) {1440}
* CREATED HYP -----------> h:03:0031 gl ((1 (1 7))) 2 {6400}
- CREATED HYP -----------> h:03:0032 gl ((1 (1 7))) 3 {3600}
- CREATED HYP -----------> h:03:0033 gl ((1 (1 7))) 1 {3600}
# PLANNER GOALING -------> (h:03:0031) synthesis
* CREATED DD GOAL -------> g:03:0019 vl ((1 (0 6 2 8))) nil (1) (h:03:0031)
                           {640}
* INSTANTIATED KSI ------> ksi:03:0028 s:gl:vl (g:03:0019) (h:03:0031)
                           <640 3200> {1152}
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Executing Node 4 ------ Inv Ksis 15 ------ Time Frame 15 ------ Node Time 16

      ; node execution for node 4, which has invoked 15 KSIs,
      ; having incorporated data for sensed time frames 1--15, and at node
```

```
        ; time 16

        ; since node 4 has received no local sensor data, it cannot form any
        ; local plans and builds an idle node-plan

# CREATED NODE PLAN -----> node-plan:04:001 idle
# SENT PLAN -------------> node-plan:04:001 nil 1
# SENT PLAN -------------> node-plan:04:001 nil 2
# SENT PLAN -------------> node-plan:04:001 nil 3
  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
        ; the node-plan messages sent by the nodes at time 16 are received by
        ; other nodes at time 18

  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  Executing Node 1 ------ Inv Ksis 17 ------ Time Frame 15 ------ Node Time 18

        ; executing node 1, which has invoked 17 KSIs, incorporated data for
        ; sensed times 1--15, and executing at node time 18

        ; initially, the node receives the node-plan messages and builds
        ; local node-plans for each received

# RECEIVED PLAN ---------> nil 4
# CREATED NODE PLAN -----> node-plan:01:005 nil
# RECEIVED PLAN ---------> plan:03:001 3
# CREATED NODE PLAN -----> node-plan:01:006 plan:03:001
# RECEIVED PLAN ---------> plan:02:001 2
# CREATED NODE PLAN -----> node-plan:01:007 plan:02:001
        ; the node finds the various combinations of node-plans to build
        ; several PGPs.  The most highly-rated PGP (based on combined
        ; node-plan ratings) is chosen as the top PGP and its attributes are
        ; formed
# PGP FORMED ------------> pgp:01:007 (node-plan:01:007 node-plan:01:006
                           node-plan:01:002)
        ; when building the plan-activity-map, the pgplanner reorders the
        ; activities to improve coordination
# PGP REORDERED ---------> pgp:01:007
        ; the reordered plan-activity-map is used to modify the local plan
        ; and its node-plan
# UPDATED PLAN ----------> plan:01:001 global effects (4 5 6 7 8 9 10 11 12)
                           ---> (9 8 7 10 11 12 6 5 4) 2796
# UPDATED NODE PLAN -----> node-plan:01:002 plan:01:001
        ; since its plans have changed, the pgplanner once again checks its
        ; attributes
# PGP FORMED ------------> pgp:01:007 (node-plan:01:007 node-plan:01:006
                           node-plan:01:002)
# PLAN QUEUE ------------> ((plan:01:003 2942) (plan:01:001 2796)
                           (plan:01:004 442) (plan:01:002 338))
# PGP INVOKED------------> pgp:01:007 ((1 ((1 7 1 7))) (2 ((2 8 2 8)))
                           (3 ((3 9 3 9))) (4 ((4 10 4 10))) (5 ((5 11 5 11)))
                           (6 ((6 12 6 12))) (7 ((7 13 7 13))) (8 ((8 14 8 14)))
                           (9 ((9 15 9 15))) (10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
```

```
                          (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                          (15 ((15 21 15 21)))) (node-plan:01:007
                           node-plan:01:006 node-plan:01:002)
# PLAN INVOKED ----------> plan:01:001 (9 8 7 10 11 12 6 5 4) ((4 ((4 10 4 10)))
                          (5 ((5 11 5 11))) (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                          (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                          (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                          (12 ((12 18 12 18)))) 2796
        ; since the plan and node-plan have changed, the updated node-plan is
        ; sent
# SENT PLAN -------------> node-plan:01:002 plan:01:001 2
# SENT PLAN -------------> node-plan:01:002 plan:01:001 3
# SENT PLAN -------------> node-plan:01:002 plan:01:001 4
* INVOKED KSI -----------> ksi:01:0036 (10) s:sl:gl 18 (g:01:0031) (h:01:0051
                          h:01:0052) {1440}
* CREATED HYP -----------> h:01:0080 gl ((9 (9 15))) 2 {6400}
- CREATED HYP -----------> h:01:0081 gl ((9 (9 15))) 3 {3600}
- CREATED HYP -----------> h:01:0082 gl ((9 (9 15))) 1 {3600}
# PLANNER GOALING -------> (h:01:0080) synthesis
* CREATED DD GOAL -------> g:01:0044 vl ((9 (8 14 10 16))) nil (1)
                          (h:01:0080) {640}
* INSTANTIATED KSI ------> ksi:01:0046 s:gl:vl (g:01:0044) (h:01:0080) <640
                          3200> {1152}
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Executing Node 2 ------ Inv Ksis 17 ------ Time Frame 15 ------ Node Time 18

       ; executing node 2, which has invoked 17 KSIs, incorporated data for
       ; sensed times 1--15, and executing at node time 18

       ; initially, the node receives the node-plan messages and builds
       ; local node-plans for each received

# RECEIVED PLAN ---------> nil 4
# CREATED NODE PLAN -----> node-plan:02:002 nil
# RECEIVED PLAN ---------> plan:03:001 3
# CREATED NODE PLAN -----> node-plan:02:003 plan:03:001
# RECEIVED PLAN ---------> plan:01:001 1
# CREATED NODE PLAN -----> node-plan:02:004 plan:01:001
# RECEIVED PLAN ---------> plan:01:003 1
# CREATED NODE PLAN -----> node-plan:02:005 plan:01:003
       ; the node finds the various combinations of node-plans to build
       ; several PGPs.  The most highly-rated PGP (based on combined
       ; node-plan ratings) is chosen as the top PGP and its attributes are
       ; formed
# PGP FORMED ------------> pgp:02:005 (node-plan:02:003 node-plan:02:001
                           node-plan:02:004)
       ; when building the plan-activity-map, the pgplanner reorders the
       ; activities to improve coordination
# PGP REORDERED ---------> pgp:02:005
       ; the reordered plan-activity-map is used to modify the local plan
       ; and its node-plan
# UPDATED PLAN ----------> plan:02:001 global effects (10 11 12 13 14 15) --->
                           (13 14 15 12 11 10) 2796
# UPDATED NODE PLAN -----> node-plan:02:001 plan:02:001
       ; since its plans have changed, the pgplanner once again checks its
       ; attributes
# PGP FORMED ------------> pgp:02:005 (node-plan:02:003 node-plan:02:001
                           node-plan:02:004)
# PLAN QUEUE ------------> ((plan:02:001 2796))
# PGP INVOKED------------> pgp:02:005 ((1 ((1 7 1 7))) (2 ((2 8 2 8)))
                           (3 ((3 9 3 9))) (4 ((4 10 4 10))) (5 ((5 11 5 11)))
                           (6 ((6 12 6 12))) (7 ((7 13 7 13))) (8 ((8 14 8 14)))
                           (9 ((9 15 9 15))) (10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) (node-plan:02:003
                           node-plan:02:001 node-plan:02:004)
```

```
# PLAN INVOKED ----------> plan:02:001 (13 14 15 12 11 10) ((10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) 2796
        ; since the plan and node-plan have changed, the updated node-plan is
        ; sent
# SENT PLAN -------------> node-plan:02:001 plan:02:001 1
# SENT PLAN -------------> node-plan:02:001 plan:02:001 3
# SENT PLAN -------------> node-plan:02:001 plan:02:001 4
* INVOKED KSI -----------> ksi:02:0055 (14) s:sl:gl 18 (g:02:0043) (h:02:0103
                           h:02:0104) {1440}
* CREATED HYP -----------> h:02:0214 gl ((13 (13 19))) 2 {6400}
- CREATED HYP -----------> h:02:0215 gl ((13 (13 19))) 3 {3600}
- CREATED HYP -----------> h:02:0216 gl ((13 (13 19))) 1 {3600}
# PLANNER GOALING -------> (h:02:0214) synthesis
* CREATED DD GOAL -------> g:02:0086 vl ((13 (12 18 14 20))) nil (1)
                           (h:02:0214) {640}
* INSTANTIATED KSI ------> ksi:02:0096 s:gl:vl (g:02:0086) (h:02:0214)
                           <640 3200> {1152}
  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

        ; the other nodes form similar PGPs
```

```
        ; the nodes then pursue their plans as expected ...

  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  Executing Node 1 ------ Inv Ksis 23 ------ Time Frame 15 ------ Node Time 24

        ; node 1 continues its local plan to provide predictive information

# PLAN QUEUE ------------> ((plan:01:003 2942) (plan:01:001 3065)
                           (plan:01:004 442) (plan:01:002 338))
# PGP INVOKED------------> pgp:01:007 ((1 ((1 7 1 7))) (2 ((2 8 2 8)))
                           (3 ((3 9 3 9))) (4 ((4 10 4 10))) (5 ((5 11 5 11)))
                           (6 ((6 12 6 12))) (7 ((7 13 7 13))) (8 ((8 14 8 14)))
                           (9 ((9 15 9 15))) (10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) (node-plan:01:007
                            node-plan:01:006 node-plan:01:002)
# PLAN INVOKED ----------> plan:01:001 (9 8 7 10 11 12 6 5 4) ((4 ((4 10 4 10)))
                           (5 ((5 11 5 11))) (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                           (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                           (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18)))) 3065
* INVOKED KSI -----------> ksi:01:0050 (24) ff:vl:vt 24 (g:01:0046)
                           (h:01:0099) {5532}
* CREATED HYP -----------> h:01:0100 vt ((8 (8 14)) (9 (9 15))) 1 {9223}
        ; the predictive hypothesis is formed and a KSI is formed and invoked
        ; to send it
* INSTANTIATED KSI ------> ksi:01:0051 hyp-send:vt nil (h:01:0100)
                           <-10000 9223> {9223}
* INVOKED KSI -----------> ksi:01:0051 (25) hyp-send:vt 25 nil (h:01:0100)
                           {9223}
* SENT HYP --------------> 2 27 h:01:0100 vt ((8 (8 14)) (9 (9 15))) 1 {9223}
* SENT HYP --------------> 3 27 h:01:0100 vt ((8 (8 14)) (9 (9 15))) 1 {9223}
* SENT HYP --------------> 4 27 h:01:0100 vt ((8 (8 14)) (9 (9 15))) 1 {9223}
  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
        ; nodes continue pursuing their plans

  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  Executing Node 2 ------ Inv Ksis 26 ------ Time Frame 15 ------ Node Time 27

        ; node 2 receives the predictive hypothesis sent by node 1

* RECEIVED HYP ----------> 1 27 h:01:0100 vt ((8 (8 14)) (9 (9 15))) 1 {9223}
* INSTANTIATED KSI ------> ksi:02:0102 hyp-receive:vt nil nil <-10000 9223>
                           {9223}
  INVOKED KSI -----------> ksi:02:0102 (27) hyp-receive:vt 27 nil nil {9223}
* CREATED HYP -----------> h:02:0238 vt ((8 (8 14)) (9 (9 15))) 1 {9223}
        ; with the new information, the node modifies the clustering
        ; hierarchy
        ; because the new hypothesis is compatible with only a subset of the
        ; data previously part of the same plan, the plan is divided -- a new
        ; plan is formed for the incompatible subset
# CREATED NODE PLAN -----> node-plan:02:006 plan:02:002
# CREATED PLAN ----------> plan:02:002 (10 11 12 13 14 15) (10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) (7 6 5 4 3 2) 2825
        ; and the existing plan is updated
# UPDATED NODE PLAN -----> node-plan:02:001 plan:02:001
# UPDATED PLAN ----------> plan:02:001 agoals changed (13 14 15 12 11 10) --->
                           (8 9 10 11 12 13 14 15)      ((10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) ---> ((8 ((8 14 8 14)))
                           (9 ((9 15 9 15))) (10 ((10 16 10 16)))
                           (11 ((11 17 11 17))) (12 ((12 18 12 18)))
                           (13 ((13 19 13 19))) (14 ((14 20 14 20)))
                           (15 ((15 21 15 21)))) 3393
        ; because the plans have changed, the PGP attributes are modified and
        ; the PGP plan-activity-map is modified
# PGP FORMED ------------> pgp:02:005 (node-plan:02:003 node-plan:02:001
                           node-plan:02:004)
# PGP REORDERED ---------> pgp:02:005
        ; the reordered plan-activity-map is used to modify the local plan
# UPDATED PLAN ----------> plan:02:001 global effects (8 9 10 11 12 13 14 15)
                           ---> (13 14 15 12 11 10 9 8) 3402
# UPDATED NODE PLAN -----> node-plan:02:001 plan:02:001
        ; because the plans have changed, the attributes are checked
# PGP FORMED ------------> pgp:02:005 (node-plan:02:003 node-plan:02:001
```

```
                        node-plan:02:004)
       ; and the local plan's short-term actions are updated
# PLANNING SHORT TERM ---> plan:02:001
# PLAN QUEUE ------------> ((plan:02:001 3704) (plan:02:002 2825))
# PGP INVOKED -----------> pgp:02:005 ((1 ((1 7 1 7))) (2 ((2 8 2 8)))
                        (3 ((3 9 3 9))) (4 ((4 10 4 10))) (5 ((5 11 5 11)))
                        (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                        (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                        (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                        (12 ((12 18 12 18))) (13 ((13 19 13 19)))
                        (14 ((14 20 14 20))) (15 ((15 21 15 21))))
                        (node-plan:02:003 node-plan:02:001 node-plan:02:004)
# PLAN INVOKED ----------> plan:02:001 (13 14 15 12 11 10 9 8)
                        ((8 ((8 14 8 14))) (9 ((9 15 9 15)))
                        (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                        (12 ((12 18 12 18))) (13 ((13 19 13 19)))
                        (14 ((14 20 14 20))) (15 ((15 21 15 21)))) 3704
# SENT PLAN -------------> node-plan:02:001 plan:02:001 1
# SENT PLAN -------------> node-plan:02:001 plan:02:001 3
# SENT PLAN -------------> node-plan:02:001 plan:02:001 4
* INVOKED KSI -----------> ksi:02:0068 (15) s:sl:gl 27 (g:02:0057) (h:02:0137
                        h:02:0138) {1440}
* CREATED HYP -----------> h:02:0239 gl ((14 (14 20))) 2 {6400}
- CREATED HYP -----------> h:02:0240 gl ((14 (14 20))) 3 {3600}
- CREATED HYP -----------> h:02:0241 gl ((14 (14 20))) 1 {3600}
# PLANNER GOALING -------> (h:02:0239) synthesis
* CREATED DD GOAL -------> g:02:0090 vl ((14 (13 19 15 21))) nil (1)
                        (h:02:0239) {640}
* INSTANTIATED KSI ------> ksi:02:0103 s:gl:vl (g:02:0090) (h:02:0239)
                        <640 3200> {1152}
  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Executing Node 1 ------ Inv Ksis 40 ------ Time Frame 15 ------ Node Time 41

      ; later on, node 1 receives the partial solution from node 3 to
      ; integrate into a larger result

* RECEIVED HYP ----------> 3 41 h:03:0095 vt ((1 (1 7)) (2 (2 8)) (3 (3 9))
                           (4 (4 10)) (5 (5 11)) (6 (6 12))) 1 {9223}
* INSTANTIATED KSI ------> ksi:01:0071 hyp-receive:vt nil nil <-10000 9223>
                           {9223}
  INVOKED KSI -----------> ksi:01:0071 (41) hyp-receive:vt 41 nil nil {9223}
* CREATED HYP -----------> h:01:0155 vt ((1 (1 7)) (2 (2 8)) (3 (3 9))
                           (4 (4 10)) (5 (5 11)) (6 (6 12))) 1 {9223}
      ; the received hypothesis causes changes to the clustering hierarchy
      ; which then causes changes to plans and node-plans
# UPDATED NODE PLAN -----> node-plan:01:002 plan:01:001
# UPDATED PLAN ----------> plan:01:001 agoals changed
                           (9 8 7 10 11 15 6 12 5 4 13 14)
                           ---> (9 8 7 10 11 15 14 13 6 5 4 3 2 1 12)
                           ((4 ((4 10 4 10))) (5 ((5 11 5 11)))
                           (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                           (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                           (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18))) (13 ((13 19 13 19)))
                           (14 ((14 20 14 20))) (15 ((15 21 15 21))))
                           ---> ((1 ((1 7 1 7))) (2 ((2 8 2 8)))
                           (3 ((3 9 3 9))) (4 ((4 10 4 10))) (5 ((5 11 5 11)))
                           (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                           (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                           (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18))) (13 ((13 19 13 19)))
                           (14 ((14 20 14 20))) (15 ((15 21 15 21)))) 4640
      ; the attributes of the pgp are reset and its plan-activity-map
      ; reordered, triggering changes to the local plan and node-plan
# PGP FORMED ------------> pgp:01:007 (node-plan:01:008 node-plan:01:007
                           node-plan:01:006 node-plan:01:002)
# PGP REORDERED ---------> pgp:01:007
# UPDATED PLAN ----------> plan:01:001 global effects
                           (9 8 7 10 11 15 14 13 6 5 4 3 2 1 12) -->
                           (9 8 7 10 11 15 6 5 4 3 2 1 12 13 14) 4640
# UPDATED NODE PLAN -----> node-plan:01:002 plan:01:001
# PGP FORMED ------------> pgp:01:007 (node-plan:01:008 node-plan:01:007
                           node-plan:01:006 node-plan:01:002)
      ; to pursue the next plan step, the planner triggers a goal to extend
```

```
        ; the track for times 1--6 into the area where it has already worked
# PLANNER GOALING -------> (h:01:0155) extension forward 7 (7 13 7 13)
* CREATED DD GOAL -------> g:01:0055 vt ((7 (7 13 7 13))) ((1 (1 7 1 7))
                           (2 (2 8 2 8)) (3 (3 9 3 9)) (4 (4 10 4 10))
                           (5 (5 11 5 11)) (6 ((6 12 6 12)) (7 (7 13 7 13))
                           (8 (8 14 8 14)) (9 (9 15 9 15)) (10 (10 16 10 16))
                           (11 (11 17 11 17))) (1) (h:01:0155) {0}
        ; and instantiates a concatenate forward KSI to join two partial
        ; tracks
* INSTANTIATED KSI ------> ksi:01:0072 cf:vt (g:01:0055) (h:01:0155) <0 9223>
                           {1844}
# PLAN QUEUE ------------> ((plan:01:001 4640) (plan:01:003 2942)
                           (plan:01:004 442) (plan:01:002 -1))
# PGP INVOKED------------> pgp:01:007 ((1 ((1 7 1 7))) (2 ((2 8 2 8)))
                           (3 ((3 9 3 9))) (4 ((4 10 4 10))) (5 ((5 11 5 11)))
                           (6 ((6 12 6 12))) (7 ((7 13 7 13)))
                           (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                           (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18))) (13 ((13 19 13 19)))
                           (14 ((14 20 14 20))) (15 ((15 21 15 21))))
                           (node-plan:01:008 node-plan:01:007
                           node-plan:01:006 node-plan:01:002)
# PLAN INVOKED ----------> plan:01:001 (9 8 7 10 11 15 6 5 4 3 2 1 12 13 14)
                           ((1 ((1 7 1 7))) (2 ((2 8 2 8))) (3 ((3 9 3 9)))
                           (4 ((4 10 4 10))) (5 ((5 11 5 11))) (6 ((6 12 6 12)))
                           (7 ((7 13 7 13))) (8 ((8 14 8 14))) (9 ((9 15 9 15)))
                           (10 ((10 16 10 16))) (11 ((11 17 11 17)))
                           (12 ((12 18 12 18))) (13 ((13 19 13 19)))
                           (14 ((14 20 14 20))) (15 ((15 21 15 21)))) 4640
# SENT PLAN -------------> node-plan:01:002 plan:01:001 2
# SENT PLAN -------------> node-plan:01:002 plan:01:001 3
# SENT PLAN -------------> node-plan:01:002 plan:01:001 4
* INVOKED KSI ----------> ksi:01:0072 (41) cf:vt 41 (g:01:0055) (h:01:0155)
                           {1844}
* CREATED HYP ----------> h:01:0156 vt ((1 (1 7)) (2 (2 8)) (3 (3 9))
                           (4 (4 10)) (5 (5 11)) (6 (6 12)) (7 (7 13))
                           (8 (8 14)) (9 (9 15)) (10 (10 16)) (11 (11 17)))
                           1 {9223}
* CREATED HYP ----------> h:01:0157 vt ((1 (1 7)) (2 (2 8)) (3 (3 9))
                           (4 (4 10)) (5 (5 11)) (6 (6 12)) (7 (7 13))
                           (8 (8 14)) (9 (9 15)) (10 (10 16))) 1 {9223}
* CREATED HYP ----------> h:01:0158 vt ((1 (1 7)) (2 (2 8)) (3 (3 9))
                           (4 (4 10)) (5 (5 11)) (6 (6 12)) (7 (7 13))
                           (8 (8 14)) (9 (9 15))) 1 {9223}
```

```
        ; since each of the hypotheses that it forms are potentially of
        ; interest to other nodes, the node instantiates KSIs to send them,
        ; but the PGP does not indicate that any should be sent currently
        ; so none are invoked
* INSTANTIATED KSI ------> ksi:01:0073 hyp-send:vt nil (h:01:0158)
                           <-10000 9223> {9223}
* INSTANTIATED KSI ------> ksi:01:0074 hyp-send:vt nil (h:01:0157)
                           <-10000 9223> {9223}
* INSTANTIATED KSI ------> ksi:01:0075 hyp-send:vt nil (h:01:0156)
                           <-10000 9223> {9223}
```

Appendix C

Glossary

alternative-goal: A possible long-term solution that a node can work toward. Corresponds to a top-level cluster in the node's clustering hierarchy. Related to other alternative-goals that share clusters at lower levels of the clustering hierarchy.

belief: The confidence in a hypothesis. In the DVMT, belief is represented as in integer (to avoid floating-point calculations) between 0 and 10000, where 0 equals no confidence and 10000 equals full confidence.

blackboard: A data structure organized for efficient storage and retrieval of information by knowledge sources. Typically has multiple blackboard-levels corresponding to different amounts of processing knowledge applied to data.

blackboard-level: Level of abstraction of data. Low blackboard-levels correspond to relatively raw data, while information at higher blackboard-levels has undergone processing (application of knowledge) that groups data into entries that abstract the attributes of the data.

cluster: An entry in the clustering hierarchy. Combines data related by some set of relationships and summarizes their attributes and the past and pending actions that can be taken on the data. Emphasizes aspects that influence control decisions.

clustering hierarchy: A hierarchy of clusters, where clusters at higher levels represent combinations of clusters at lower levels. Formed by combining clusters using a succession of relationships, to emphasize different relationships that affect control at different levels.

cooperation-parameters: A set of numeric parameters that affect how the PG-Planner coordinates nodes. Gives relative importance of redundancy versus reliability, of independence versus predictiveness, and so on (see Figure 65). Also specifies information about integration redundancy and task-passing.

DVMT: See distributed vehicle monitoring testbed.

distributed problem solving network: A network of communicating problem solvers that can work independently but that must cooperate to achieve larger network goals due to their individual limitations.

distributed vehicle monitoring testbed: The experimental testbed in which the new mechanisms are implemented and evaluated. Simulates a network of vehicle monitoring problem solvers where each problem solver monitors an area and problem solvers must cooperatively track vehicles that pass through more than one of their areas.

environment: A network problem solving situation, specified by an environment file. Indicates both the characteristics of the network (number and location of nodes, communication topology and delay, sensed areas) and of the problem (what data arrives where and when).

event-class: Characterizes information in a hypothesis. At low blackboard-levels in the vehicle monitoring task, an event-class corresponds to a particular frequency or group of frequencies, while at higher blackboard-levels an event-class corresponds to a type of vehicle or vehicle formation.

goal: An explicit representation indicating a possible way that a hypothesis could be used. Stored on a separate goal blackboard and used to reason about relationships between possible actions (KSIs) and their results.

goal processor: Forms and manipulates goals. Responsible for generating goals based on new hypotheses, forming subgoals of important goals, and generating KSIs to satisfy goals.

goals of cooperation: Different and often conflicting objectives of cooperation, such as to provide predictive information, avoid interference, verify results, avoid duplication of effort, and exchange only useful information.

hypothesis: A partial solution at some blackboard-level. Represents some combination of sensory data that meets criteria specified by one or more KSs. Given a belief depending on how well it meets those criteria, and also has time-locations and an event-class.

i-goal: See intermediate-goal.

incremental planning: Interleaving planning with execution so that plans are formed over time. Useful when later steps of a plan depend on earlier steps, but where the outcome of the earlier steps cannot be predicted with certainty. Avoids detailing actions for future situations that may never occur but plans actions for the near future based on long-term strategic objectives.

interest area: Specifies a problem solving responsibility of a node as part of the domain-level organization. Includes information about blackboard-levels, sensed times, spatial regions, and event-classes where the node should prefer developing hypotheses.

intermediate-goal: A major subgoal of a long-term problem solving goal. In the vehicle monitoring task, an intermediate-goal is to develop data for a particular sensed time and incorporate that data into existing partial solutions. Each intermediate-goal may require several actions (KSIs) to achieve it.

KS: See knowledge source.

KSI: See knowledge source instantiation.

knowledge source: Contains domain problem solving knowledge. Takes goals to achieve and hypotheses (data) to use as input, searches the blackboard for other relevant hypotheses, applies domain knowledge, and forms new hypotheses as output.

knowledge source instantiation: Represents the potential application of a KS on specific hypotheses to achieve certain goals. Rated and stored on a queue.

local control: Control of an individual computing node. Determines what action(s) that node will take at any given time.

local node-plan: A node-plan that corresponds to plan at the node. The node can also have node-plans for other nodes.

local planner: A node's mechanisms for identifying its long-term goals and planning its own problem solving actions.

meta-level organization: Static information that guides nodes coordination activities. Specifies coordination roles and where coordination information should be transmitted for some nodes in the network to develop partial global plans.

network control: Control of network activity. Determines actions and interactions for nodes in the network.

network-model: A node's view of the network. Contains node-models for each of the nodes that are known to this node (including whatever node-plans it has received from them) and partial global plans for groups of nodes.

node: An independent, often artificially intelligent, problem solver that can develop partial solutions from its local information. Must cooperate with others to form more complete solutions.

node execution: A sequence of activities performed by a node that it repeats each time it invokes a KSI. These activities include incorporating any received information into its data structures, modifying its view for control (goals, KSIs, plans, PGPs), transmitting any messages, and invoking a KSI. The DVMT simulates concurrent activity in the network by interleaving node-executions for the various nodes.

node-model: Represents what a node knows about the plans and characteristics of another node. Contains node-plans for that node that have been

received (and are updated through communication) and more static information about communicating with that node and about its problem solving capabilities.

node-plan: Summarizes a plan. When nodes communicate about their individual (local) plans, they summarize these plans into node-plans to convey the information that is most relevant for coordination, including the objectives and the long-term aspects of how the plan will be pursued.

organization: Information about the roles and responsibilities of nodes. The domain-level organization guides problem solving activities while the meta-level organization guides coordination activities. In the current implementation, an organization is assumed to be static and known to all nodes.

PGG: See partial global goal.

PGP: See partial global plan.

PGPlanner: See partial global planner.

PGP-partial-solution: A representation used when forming a solution-construction-graph. Represents a partial solution formed by some node(s) in achieving the PGP's objective.

partial global goal: Specifies an objective that one or more nodes should plan to achieve. Combines the objectives of one or more local plans into a possibly larger objective (to track a vehicle moving among a group of nodes).

partial global plan: Represents the concurrent actions and expected interactions of nodes pursuing a PGG.

partial global planner: Mechanisms that are part of a node that the node uses to develop PGGs, to form PGPs, and to modify local plans based on PGPs so that it better contributes to network problem solving.

plan: Represents how a node expects to achieve one or more local long-term goals (alternative-goals). Includes long-term plans about sequences of i-goals to achieve and short-term plans about specific KSIs to invoke in the near future.

plan-activity-map: A representation of what activities nodes will pursue over various expected intervals. When part of a node-plan, indicates the sequence of actions the node will take to achieve the plan and when it will form partial results. When part of a PGP, shows the concurrent activities of nodes and when they will individually form partial results.

solution-construction-graph: An attribute of a PGP. Indicates when and where partial solutions formed by nodes should be integrated into larger partial solutions until the overall solution is formed.

time-cushion: A numeric value indicating a "negligible" time difference. Used in several ways by the PGPlanner. When a plan deviates from predictions in terms of *when* results are formed, the PGPlanner compares the deviation to the time-cushion to determine whether the difference is significant enough to require changes to PGPs. When two partial solutions could be combined at more than one node, the PGPlanner uses the time-cushion to decide whether the combination could be formed *substantially* earlier somewhere, or whether the difference is negligible.

time-locations: A series of times and locations, each indicating where a vehicle was at a particular sensed time.

time-order: Represents a sequence of i-goals to achieve. Since an i-goal in the DVMT domain corresponds to processing data for a certain sensed time, an i-goal can be identified by that sensed time. The time-order is a sequence of sensed times where each represents an i-goal.

time-predictions: A list of time-predictions, one for each i-goal in the time-order. Each time-prediction indicates the predicted time needed to achieve a particular i-goal.

Bibliography

[Aikins, 1980] Jan S. Aikins. *Prototypes and Production Rules: A Knowledge Representation for Computer Consultations.* PhD thesis, Stanford University, 1980. Available as Technical Report STAN-CSD-80-814, Computer Science Department, Stanford University.

[Allen, 1983] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[Allen, 1984] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[Arbib, 1972] Michael A. Arbib. *The Metaphorical Brain.* Wiley, 1972.

[Axelrod, 1984] Robert Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

[Barto, 1985] Andrew G. Barto. *Learning by Statistical Cooperation of Self-Interested Neuron-Like Computing Elements.* Technical Report 85-11, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, April 1985.

[Bruce and Newman, 1978] Bertram Bruce and Denis Newman. Interacting plans. *Cognitive Science*, 2:195–233, 1978.

[Cammarata *et al.*, 1983] Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 767–770, August 1983.

[Carver *et al.*, 1984] Norman F. Carver, Victor R. Lesser, and Daniel L. McCue. Focusing in plan recognition. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 42–48, August 1984.

[Chandrasekaran, 1981] B. Chandrasekaran. Natural and social system metaphors for distributed problem solving: Introduction to the issue. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):1–5, January 1981.

[Cheeseman, 1984] Peter Cheeseman. A representation of time for automatic planning. In *Proceedings of the IEEE Conference on Robotics*, pages 513–518, 1984.

[Chien and Weissman, 1975] R. T. Chien and S. Weissman. Planning and execution in incompletely specified environments. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 169–174, August 1975.

[Clancey and Letsinger, 1981] William J. Clancey and Reed Letsinger. NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 829–836, August 1981.

[Cohen, 1978] Philip R. Cohen. *On Knowing What to Say: Planning Speech Acts.* PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, Toronto, Canada, January 1978. Technical Report 118.

[Cohen and Levesque, 1986] Philip R. Cohen and Hector J. Levesque. Persistence, intention, and commitment. In *Proceedings of the 1986 Workshop on Reasoning About Actions and Plans*, July 1986.

[Corkill, 1979] Daniel D. Corkill. Hierarchical planning in a distributed environment. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 168–175, August 1979.

[Corkill, 1983] Daniel David Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks.* PhD thesis, University of Massachusetts, Amherst, Massachusetts 01003, February 1983. Available as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1982.

[Corkill and Lesser, 1983] Daniel D. Corkill and Victor R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, August 1983.

[Corkill and Lesser, 1987] Daniel D. Corkill and Victor R. Lesser. Distributed problem solving. In *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, 1987.

[Corkill *et al.*, 1986] Daniel D. Corkill, Kevin Q. Gallagher, and Kelly E. Murray. GBB: A generic blackboard development system. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1008–1014, 1986. (Also to appear in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, Addison-Wesley, in press, 1987).

[Corkill *et al.*, 1982] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlicka. Unifying data-directed and goal-directed control: An example and experiments. In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 143–147, August 1982.

[Davis, 1981] Randall Davis. *A Model for Planning in a Multi-Agent Environment: Steps Toward Principles of Teamwork.* Technical Report MIT AI Working Paper 217, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, Massachusetts, June 1981.

[Davis and Smith, 1983] Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[Dawkins, 1976] Richard Dawkins. *The Selfish Gene.* Oxford University Press, 1976.

[Dean, 1986] Thomas L. Dean. Intractability and time dependent planning. In *Proceedings of the 1986 Workshop on Reasoning About Actions and Plans*, July 1986.

[Drummond *et al.*, 1987] Mark Drummond, Ken Currie, and Austin Tate. Contingent plan structures for spacecraft. In *Proceedings of the Space Telerobotics Workshop*, January 1987.

[Durfee and Lesser, 1986] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 58–64, August 1986.

[Durfee and Lesser, 1987] Edmund H. Durfee and Victor R. Lesser. *Using Partial Global Plans to Coordinate Distributed Problem Solvers.* Technical Report 87-06, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, January 1987. Also to appear in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August, 1987.

[Durfee *et al.*, 1984] Edmund H. Durfee, Daniel D. Corkill, and Victor R. Lesser. Distributing a distributed problem solving network simulator. In *Proceedings of the Fifth Real-Time Systems Symposium*, pages 237–246, December 1984.

[Durfee *et al.*, 1985a] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. *Coherent Cooperation Among Communicating Problem Solvers.* Technical Report 85-15, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, April 1985. Also to appear in *IEEE Transactions on Computers.*

[Durfee *et al.*, 1985b] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Increasing coherence in a distributed problem solving network. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1025–1030, August 1985.

[Durfee *et al.*, 1987] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Cooperation through communication in a distributed problem solving network. In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, Pitman, 1987. (In press. Also to appear as Chapter 7 in Scott P. Robertson, Wayne Zachary, and John Black, editors, *Cognition, Computing, and Cooperation: Collected works on cooperation in complex systems*, in press).

[Erman *et al.*, 1980] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[Fehling and Erman, 1983] Michael Fehling and Lee Erman. Report on the third annual workshop on distributed artificial intelligence. *SIGART Newsletter*, 84:3–12, April 1983.

[Feldman and Ballard, 1982] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6:205–254, 1982.

[Feldman and Sproull, 1977] Jerome A. Feldman and Robert F. Sproull. Decision theory and artificial intelligence II: The hungry monkey. *Cognitive Science*, 1:158–192, 1977.

[Fennell and Lesser, 1977] Richard D. Fennell and Victor R. Lesser. Parallelism in artificial intelligence problem solving: A case study of Hearsay II. *IEEE Transactions on Computers*, C-26(2):98–111, February 1977.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fox, 1983] Mark S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* PhD thesis, Carnegie-Mellon University, December 1983. Available as Technical

Report CMU-CS-83-161, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA 15213, December 1983.

[Georgeff, 1983] Michael Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 125–129, August 1983.

[Georgeff, 1984] Michael Georgeff. A theory of action for multiagent planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 121–125, August 1984.

[Grosz and Sidner, 1985] Barbara J. Grosz and Candace L. Sidner. Discourse structure and the proper treatment of interruptions. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 832–839, August 1985.

[Halpern and Moses, 1984] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In *Third ACM Conference on Principles of Distributed Computing*, 1984.

[Hanson and Riseman, 1978] A. R. Hanson and E. M. Riseman. *Computer Vision Systems (CVS)*. Academic Press, 1978.

[Hayes-Roth, 1985] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[Hayes-Roth and Lesser, 1977] Frederick Hayes-Roth and Victor R. Lesser. Focus of attention in the Hearsay-II speech understanding system. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 27–35, August 1977.

[Hernandez *et al.*, 1987] Joseph A. Hernandez, Daniel D. Corkill, and Victor R. Lesser. *Goal Processing in Blackboard Architectures*. Technical Report 87-24, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, March 1987.

[Hewitt, 1977] Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8:323–364, 1977.

[Hudlicka, 1986] Eva Hudlicka. *Diagnosing Problem-Solving System Behavior*. PhD thesis, University of Massachusetts, February 1986. (Also published as Technical Report 86-03, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, February 1986.).

[Hudlicka *et al.*, 1986] Eva Hudlicka, Victor R. Lesser, Jasmina Pavlin, and Anil Rewari. *Design of a Distributed Diagnosis System.* Technical Report 86-63, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1986.

[Jagannathan and Dodhiawala, 1986] V. Jagannathan and Rajendra Dodhiawala. Distributed artificial intelligence: An annotated bibliography. *SIGART Newsletter*, (95):44–56, January 1986.

[Konolige, 1984] Kurt Konolige. A deductive model of belief. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 377–381, August 1984.

[Kurose *et al.*, 1985] J. F. Kurose, M. Schwartz, and Y. Yemini. A microeconomic approach to optimization of channel access policies in multiaccess networks. In *Proceedings of the Fifth International Symposium on Distributed Computing Systems*, pages 70–80, May 1985.

[Lansky, 1985] Amy L. Lansky. *Behavioral Specification and Planning for Multiagent Domains.* Technical Report 360, SRI International, Menlo Park, CA 94025, November 1985.

[Lesser and Corkill, 1981] Victor R. Lesser and Daniel D. Corkill. Functionally-accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):81–96, January 1981.

[Lesser and Corkill, 1983] Victor R. Lesser and Daniel D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, Fall 1983.

[Lesser and Erman, 1980] Victor R. Lesser and Lee D. Erman. Distributed interpretation: A model and experiment. *IEEE Transactions on Computers*, C-29(12):1144–1163, December 1980.

[Lesser and Pavlin, 1987] Victor R. Lesser and Jasmina Pavlin. *Real-time control in AI problem solving.* Technical Report, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, 1987 (in preparation).

[Lowrance and Friedman, 1977] John D. Lowrance and Daniel P. Friedman. Hendrix's model for simultaneous actions and continuous processes: An introduction and implementation. *International Journal of Man-Machine Studies*, 9:537–581, 1977.

[Malone and Smith, 1984] Thomas W. Malone and Stephen A. Smith. *Tradeoffs in Designing Organizations: Implications for New Forms of Human Organizations and Computer Systems.* Technical Report Sloan Working Paper 1541-84, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, March 1984.

[March and Simon, 1958] James G. March and Herbert A. Simon. *Organizations.* Wiley, 1958.

[McCalla *et al.*, 1982] Gordon I. McCalla, Larry Reid, and Peter F. Schneider. Plan creation, plan execution, and knowledge acquisition in a dynamic microworld. *International Journal of Man-Machine Studies*, 16:89–112, 1982.

[McDermott, 1982] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Nilsson, 1980] Nils J. Nilsson. Two heads are better than one. *SIGART Newsletter*, (73):43, October 1980.

[Pattison *et al.*, 1985] H. Edward Pattison, Daniel D. Corkill, and Victor R. Lesser. *Instantiating Descriptions of Organizational Structures.* Technical Report 85-45, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1985.

[Pavlin, 1983] Jasmina Pavlin. Predicting the performance of distributed knowledge-based systems: A modeling approach. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 314–319, August 1983.

[Pavlin and Corkill, 1984] Jasmina Pavlin and Daniel D. Corkill. Selective abstraction of AI system activity. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 264–268, August 1984.

[Ramamritham and Stankovic, 1984] Krithivasan Ramamritham and John A. Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE Software*, 1(3):65–75, July 1984.

[Rosenschein, 1985] Jeffery Solomon Rosenschein. *Rational Interaction: Cooperation Among Intelligent Agents.* PhD thesis, Stanford University, October 1985.

[Rosenschein and Genesereth, 1985] Jeffrey S. Rosenschein and Michael R. Genesereth. Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 91–99, August 1985.

[Rosenschein and Genesereth, 1987] Jeffrey S. Rosenschein and Michael R. Genesereth. Communication and cooperation among logic-based agents. In *Proceedings of the Sixth Phoenix Conference on Computers and Communications*, pages 594–600, February 1987.

[Sacerdoti, 1977] Earl D. Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, 1977.

[Sacerdoti, 1979] Earl D. Sacerdoti. Problem solving tactics. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 1077–1085, August 1979.

[Shortliffe, 1976] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* American Elsevier, 1976.

[Shubik, 1982] Martin Shubik. *Game Theory in the Social Sciences: Concepts and Solutions.* MIT Press, 1982.

[Smith, 1980] Reid G. Smith. The contract-net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[Smith and Davis, 1981] Reid G. Smith and Randall Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):61–70, January 1981.

[Stankovic, 1984] John A. Stankovic. A perspective on distributed computer systems. *IEEE Transactions on Computers*, C-33(12):1102–1115, December 1984.

[Stankovic *et al.*, 1985] John A. Stankovic, Krithivasan Ramamritham, and ShengChang Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Transactions on Computers*, C-34(12):1130–1143, December 1985.

[Steeb *et al.*, 1986] Randall Steeb, Stephanie Cammarata, Sanjai Narain, Jeff Rothenberg, and William Giarla. *Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control.* Technical Report R-3408-ARPA, Rand Corporation, October 1986.

[Stefik, 1981] Mark Stefik. Planning with constraints. *Artificial Intelligence*, 16:111–140, 1981.

[Tate, 1975] Austin Tate. Interacting goals and their use. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 215–218, August 1975.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893, August 1977.

[Vere, 1983] Steven A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(3):246–267, May 1983.

[Waldinger, 1977] Richard Waldinger. Achieving several goals simultaneously. In *Machine Intelligence 8*, pages 94–136, 1977.

[Wesson, 1977] Robert B. Wesson. Planning in the world of the air traffic controller. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 473–479, August 1977.

[Wilkins, 1984] David E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[Woods, 1977] W. A. Woods. Shortfall and density scoring strategies for speech understanding control. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 13–26, August 1977.

[Zhao, 1986] Wei Zhao. *A Heuristic Approach to Scheduling Hard Real-Time Tasks with Resource Requirements in Distributed Systems.* PhD thesis, University of Massachusetts, Amherst, Massachusetts 01003, February 1986.

[Zhao *et al.*, 1987] Wei Zhao, Krithi Ramamritham, and John A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, C-36(8):949–960, August 1987.