A FRAMEWORK FOR ORGANIZATIONAL SELF-DESIGN IN DISTRIBUTED PROBLEM SOLVING NETWORKS

Daniel D. Corkill

COINS Technical Report 82-33

December 1982

A FRAMEWORK FOR ORGANIZATIONAL SELF-DESIGN IN DISTRIBUTED PROBLEM SOLVING NETWORKS

A Dissertation Presented

By

DANIEL DAVID CORKILL

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

DOCTOR OF PHILOSOPHY

February 1983

Computer and Information Science

This work was supported in part by:

The National Science Foundation
Grant Number MCS78-04212

and

The Office of Naval Research
Grant Number N00014-79-C-0439

ACKNOWLEDGEMENTS

This dissertation would not have happened without the friendship, direction, encouragement, collaboration, and intellectual support I received from Victor Lesser. I am continually amazed by his energy, his insatiable appetite for new ideas, and the multitude and richness of his insights. It was Victor who first opened my eyes to the exciting potential of distributed problem solving networks.

The diverse research interests of the other members of my dissertation committee also contributed much to this work:

Michael Arbib introduced me to the work of Stafford Beer, suggested some changes in the structure of the dissertation, and forced me to clarify many of the ideas. Kenan Sahin encouraged my attempts to relate organizational structuring in distributed problem solving networks to organizational concepts in business and management organizations. Somewhat skeptical of artificial intelligence research, Jack Wileden posed some difficult questions (a few of which remain unanswered). Jack also untied some of the knots in my exposition and my grammar.

I would also like to thank Richard Leifer for discussions of organizational structuring in business and management organizations during the early stages of this work.

Susan Lander, Jon Lander, and Jim Hoffman drew many of the figures appearing in the dissertation. Susan also typed all the figures and caught many of my spelling errors.

A project as large and complex as the Distributed Vehicle Monitoring Testbed required the time and energies of a number of individuals:

The selection of the distributed vehicle monitoring task and the preliminary design of the testbed were done by Victor Lesser, Jasmina Pavlin, and Scott Reed. Raam Mukunda was responsible for the preliminary implementation of the (rather involved) FRONTEND knowledge source. Jasmina Pavlin completed the coding of the FRONTEND and implemented the local problem solving and communication knowledge sources. (Jasmina also showed me the Tyspin quote). Eva Hudlicka coded and debugged several iterations of the subgoaling and goal satisfaction components of the local node planner and implemented the goal rating function. Larry Lefkowitz implemented the knowledge source rating function and the local node and system state measures. Jon Borden, Larry Lefkowitz, Jim Hoffman, and Sheryl Franklin worked on the dynamic color graphic displays of testbed activity (which unfortunately only appear in static black and white in this document). Richard Brooks helped design the format for environment files. (Richard also helped me rediscover the joys of skating.) Victor kept us all on our toes!

The students, faculty, and staff of the Computer and Information Science Department have been a source of numerous dicussions and friendships (and volleyball games). In particular, John Lowrance was a source of many prolonged and productive arguments during our formative years.

On the more personal side, I would like to express my appreciation to my parents for their support (financial and emotional) during what seemed like such a long time (and was).

Finally, I would like to thank my wife Susan for her assistance whenever I needed it and for keeping the household a welcome haven when I was too busy to help. I couldn't have done it without you!

ABSTRACT

A Framework for Organizational Self-Design
in Distributed Problem Solving Networks

February 1983

Daniel D. Corkill

BS, MS, University of Nebraska

PhD, University of Massachusetts

Directed by: Professor Victor Lesser

A distributed problem solving network is a distributed network of semi-autonomous nodes that perform sophisticated problem solving and cooperatively interact with other nodes to solve a single problem. Because interaction among nodes is both limited and unreliable, each node directs its own activities in concert with other nodes, using potentially incomplete, inaccurate, and inconsistent information. Each node must balance its own perceptions of appropriate local problem solving activity with activities deemed important by other nodes.

Organizational self-design is proposed as a multilevel approach to coordinating these networks. An organizational structure specifies the information and control relationships among the nodes in a general way. Each node is responsible for elaborating these relationships into precise activities to be performed by the node.

This work focuses on the design and implementation of a coordination framework that is responsive to organizational structuring decisions. This framework is implemented as part of the distributed vehicle monitoring testbed: a flexible and fully-instrumented research tool for the empirical evaluation of distributed network designs and coordination policies. Each node uses a goal-directed Hearsay-II architecture and a local node planner to provide the sophisticated local control necessary for a node to evaluate its activity decisions based on internal, external, and organizational criteria. The capabilities of the framework are illustrated with testbed experiments using different organizational structures in four-node and five-node distributed networks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

The Cosmic Command, obviously no longer able to supervise every assignment on an individual basis when there were literally trillions of matters in its charge, had switched over to a random system. The assumption would be that every document, circulating endlessly from desk to desk, must eventually hit upon the right one. A time-consuming procedure, perhaps, but one that would never fail.

— Stanislaw Lem

CHAPTER I

OVERVIEW

This chapter presents an overview and the main ideas of the dissertation, as well as delimiting the notion of distributed problem solving, describing its major characteristics, and contrasting it with other problem solving forms. The last section of this chapter describes the overall structure of the dissertation.

1.1 Introduction

Large tasks are not solved simply by throwing more workers at them. Adding more workers appears to obey not only the law of diminishing returns (where each new worker contributes less and less compared to a single individual) but also the law of the committee (where the addition of too many workers actually decreases the performance of the group).

If applying more individuals to a task does lead to a reduction in productivity, what happens to the added capability? It is:

o lost as members wait for something to do;

- o wasted as members work at cross-purposes with one another;
- o redundantly applied as members duplicate efforts;
- o expended in an attempt to avoid these inefficiencies.

In short, the additional capability is spent on coordination or lost due to a lack of coordination.

Of course there are tasks too large for a single individual to accomplish alone. Biological organisms, animal and human societies, businesses, and governments have addressed these tasks by evolving coordination techniques sufficient for their survival. They are successful to the extent that they can accomplish more than the same number of individuals acting independently. Their success depends on striking an appropriate balance between the effort wasted due to a lack of coordination and the effort expended on coordination itself.

There are also problem solving tasks too large to be handled effectively using a single computer system. These include computationally massive tasks requiring the simultaneous application of numerous processors for timely execution. However it is a second form of size that is of interest here: problem solving tasks in which the system's input and output are spatially dispersed, requiring extensive communication to route information to and from a single site. With these <u>distributed problem solving tasks</u>, a distributed network of cooperating, semi-autonomous computer systems can be more effective than a centralized system -- if the activities of the individual systems are sufficiently coordinated.

This dissertation develops a framework for coordinating the individual systems, or nodes, in such distributed problem solving networks. Each node is itself a sophisticated problem solving system, capable of modifying its behavior as circumstances change and planning its own communication and cooperation strategies with other nodes. Such node sophistication is viewed as a requirement for effective cooperation among large numbers of nodes operating in dynamic distributed problem solving environments.

The existence of sophisticated nodes also allows the use of organizational structuring as a means of achieving network coordination. Communication limitations and network reliability considerations make it infeasible to have a single node direct all the activities of the other nodes in the network. Instead, most decisions regarding each node's activity are made within the context of an organizational structure by the node itself, perhaps after consultation with a small group of nearby nodes. The organizational structure specifies the information, communication, and control relationships among the nodes in a very general way. The idea is to include in the organizational structure those decisions that are not quickly outdated and that pertain to large numbers of nodes. The development and use of an organizational structure is intended to be less complex and dynamic than directly coordinating the overall distributed problem solving task.

A simple example of an organizational structure is a hierarchy in which the responsibilities of a number of "worker" nodes are defined by a higher level "integrating" node (Figure 1).[1] The responsibilities of each worker node are delineated along functional, spatial, or temporal lines and serve to restrict the range of possible activities each worker node need consider. Situations that cannot be handled by a single worker node are passed upward for the integrating node to resolve [GALB77].

In a sense, an organizational structure is a high-level, "strategic" plan describing and delimiting the gross responsibilities of each node in the network; an example of the use of meta-level control to coordinate activity in a complex system. A significant portion of the control activity of each node is elaboration of these responsibilities into precise activities to be performed by the node. In the simple hierarchy example, each worker node must still decide what particular activities are required to satisfy its responsibilities and determine what particular information should be passed up to the integrating node and laterally to interested worker nodes (as specified by the integrating node).

The organizational structure of the distributed problem solving network strongly influences its effectiveness in a given situation. This effectiveness is a multifaceted measure incorporating such considerations as processing resources, communication requirements, network reliability, timeliness of activities, and suitability of activities. An inflexible organizational structure can lead to a loss of network effectiveness if the internal or external environment of the

1. Hierarchies and distributed problem solving are not necessarily a good marriage.

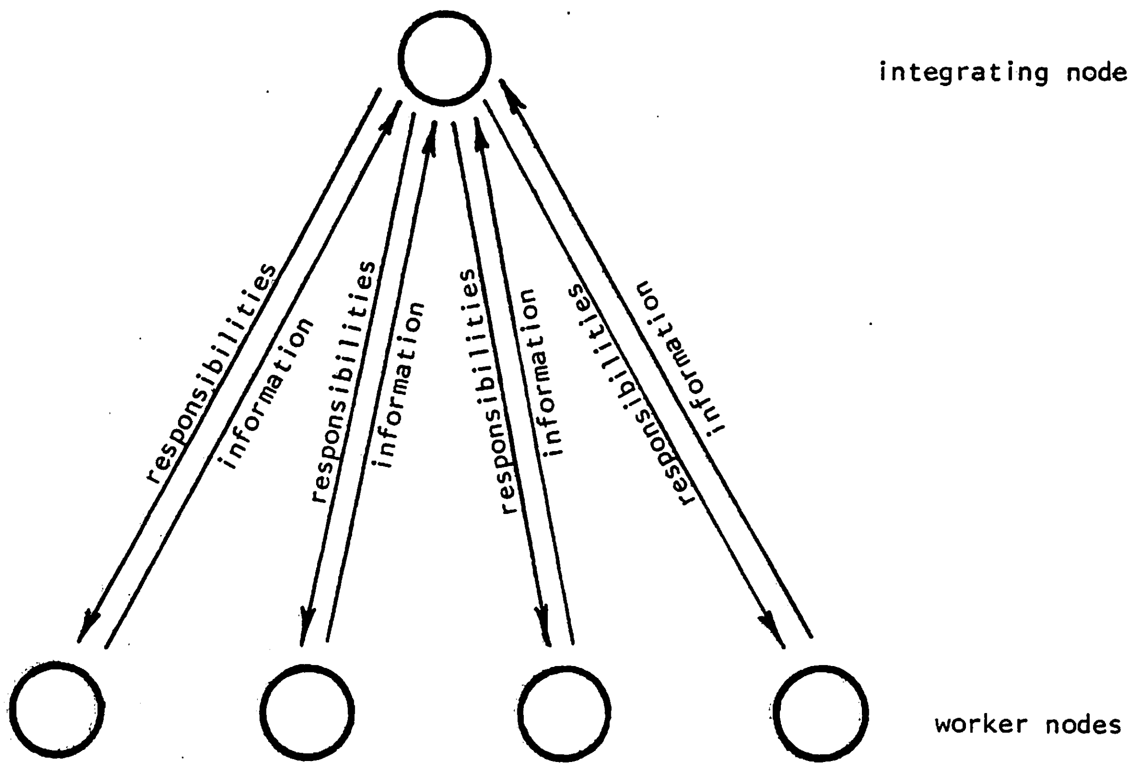


Figure 1: A Simple, Hierarchical Organizational Structure.

A simple, one-level hierarchical organizational structure in which a single "integrating" node oversees the activities of a number of "worker" nodes.

distributed problem solving network changes.

In the simple, one-level hierarchical organizational structure, worker responsibilities may need to be reallocated if some worker nodes are overloaded and other worker nodes remain relatively idle. If the integrating node becomes overloaded, additional non-local decisionmaking authority may need to be passed down to the worker nodes. If the integrating task becomes excessively difficult, the entire hierarchical structure may need to be augmented with an intermediate level of integrating nodes. It may even be appropriate to replace the hierarchical structure with a completely different organizational form.

Because an effective organizational structure is dependent upon the dynamics of the problem solving situation, the distributed problem solving network must initially develop an organizational structure and as problem solving progresses:

- o monitor for decreased effectiveness caused by an inappropriate organizational structure;
- o determine plausible alternative structures;
- o evaluate the cost and benefits of continuing with its current structure versus reorganizing itself into one of the alternative structures;
- o carry out the reorganization if appropriate.

This development and maintenance of an organizational structure by the network itself is <u>organizational self-design</u>.

There are two basic approaches to organizational self-design. One approach is to predetermine a "cookbook" of situation-organization pairs. The network monitors for a change in its problem solving situation and, if a change is detected, uses the associated predetermined organizational structure as its new organizational form. Another approach is to provide the network with knowledge about situations and organizational forms and have the network develop plausible alternative structures as the situation warrants.

A second issue is how the network performs organizational self-design. The design task could be performed at a single "designer" node. Alternatively, the organizational design task could itself be distributed among the nodes, proceeding concurrently with the overall problem solving task. This distributed organizational self-design task competes with the basic problem solving activities of each node in the network.

This dissertation does not directly address the development of an organizational self-designer, and organizational self-design remains on the research horizon. The focus here is on the use of organizational structuring as a coordination technique for large distributed problem solving networks and on the design and implementation of a coordination architecture that is responsive to organizational structuring decisions. (A forward look at organizational self-design research is presented in Chapter VI.)

The ability to coordinate activity within a distributed problem solving network is important for several reasons. One important reason is that actual distributed problem solving networks are now being constructed. The initial motivation for their development came largely from advances in microprocessor and network technologies which made physically distributed problem solving networks economically feasible [CHU77, KIMB75, SCIE77]. While the first networks may consist of only a small number of nodes, distributed problem solving networks may eventually contain hundreds or thousands of individual nodes. We are nearing a situation of exciting hardware possibilities unaccompanied by the problem solving technology required for their effective utilization.

As important as the architectural motivations are, an equal (if not greater) motivation is understanding the process of cooperation in networks of interacting problem solvers. Whether the underlying system is societal, managerial, biological, or mechanical, we seem to understand competition far better than cooperation. It is possible that the development of distributed problem solving networks may serve the same validating role to theories in sociology, management, organizational theory, and biology as the development of artificial intelligence systems has served to theories of problem solving and intelligence in psychology and philosophy.

1.2 What Distributed Problem Solving Is (And What It Is Not)

In the introduction, distributed problem solving was described as the cooperative solution of a problem using a network of interacting problem solving nodes. That description covers a broad range of problem solving situations with widely varying and even contradictory characteristics. In this section, a more restricted view of distributed problem solving is developed and contrasted with parallel processing, distributed processing, and cooperating experts. While all these problem solving forms deal with concurrent and interacting tasks, each has a particular emphasis. To help accentuate these differences, an extreme position is taken on each problem solving form.

1.2.1 Characteristics of distributed problem solving.

The major characteristic of distributed problem solving is limited interaction between the nodes. Limited interaction stems from both modularization of the problem solving task and physical separation of nodes in the network.

Large and complex systems are easier to build and understand if they are logically distributed. The system is decomposed into a number of relatively independent modules which interact to perform the overall task. Logical distribution divides the complexity of the system into two parts: the functionality of each module and the coordination of the modules into an effective overall system.

Intermodule interaction is limited due to the bounded rationality of computation described by Simon [SIMO57]. Processing resources are expended in interactions with other problem solving modules. Because processing resources are limited, a module sending or receiving many messages spends system resources on communicating rather than on computation. In addition, because processing speed is limited, a module receiving or sending a large number of messages encounters a delay in processing all the messages. Finally, if a given module needs to interact with a number of disparate modules, a general interface (which covers the full range of possible messages) is required. Such generality further increases the processing resources required for interaction.

In physically distributed systems, node interaction is also limited by the physical properties of the communication channel. For example, the capacity of the channel is bounded. With packet-radio communication [KAHN78], the possibility of a message interfering with another message increases with the amount of information to be communicated and the distance between interacting nodes. The potential for saturating hard-wired links similarly increases with the amount of information and the number of intermediate nodes needed to forward messages.

Internode communication is also subject to significant time delay. Delay arises from propagation delays in the communication media and queuing delays for accessing the communication channel.

Finally, messages may be incorrectly exchanged or lost altogether. Errors in communication can result from encoding/decoding errors and interference during transmission. Loss can result from hardware failure in communication channels or nodes and from incorrect routing of messages. In some distributed communication networks, the usable capacity of the communications channel is significantly degraded if the correct reception of all messages needs to be verified. In these situations, a problem solving network that can function effectively without explicit acknowledgment of messages may be advantageous.

The relative isolation of the nodes gives physically distributed problem solving its special character. Node activity is, by necessity, loosely-coupled. A node can perform significantly more computation than communication. In some cases it can be faster and result in better network performance for a node to redundantly derive information than to request the existing information from another node. Limited node interaction also makes it infeasible to keep every node fully abreast of the current state of problem solving at the other nodes in the network. Therefore nodes must be able to function with information that is incomplete or inconsistent with information at other nodes. Such incompleteness and inconsistency is true not only of the problem solving data in the network, but of the present and planned future activities of the nodes as well. This adds to the difficulty of coordinating node activity.

A node may become temporarily isolated from the rest of the network due to channel saturation or failure. It is appropriate for such a node to continue its processing in isolation, contributing its work if communication with the rest of the network is reestablished. This requires each node to be self-reliant, able to function in temporary isolation from the other nodes in the network, and to direct its own activities based upon its local view of the state of problem solving in the network.

Physically distributed problem solving networks are also logically distributed. However, the spatial distribution of information in the network places limitations on the possible decompositions of the network. When coupled with the additional interaction constraints associated with physical communication between nodes, interaction in physically distributed problem solving appears to be significantly more restricted than in its logically distributed cousin. This research emphasizes physically distributed problem solving.

A second characteristic of distributed problem solving is the need for cooperation among the nodes. The distribution of information in the network may be such that no single node possesses sufficient information to solve the problem alone or even sufficient information to realize that a problem exists to be solved. Therefore, nodes must cooperate to mutually identify and solve the problem. Each node may need to share a little of what it knows with other nodes so that enough information is accumulated to allow the problem to be solved. As will be discussed in Section 2.2, such cooperation does not demand the exchange of accurate information. Given a suitable problem solving approach, the exchange of "educated guesses" can be very effective. Of course, this cooperation must be performed within the bounds of limited communication discussed above.

A third characteristic of distributed problem solving is its potential for reliable problem solving. Since failure of a few nodes or communication paths amounts to only a fraction of the components of a large network, it may be possible to have other components take up the slack. Rather than failing at the loss of a single component, the distributed problem solving network degrades gracefully by decreasing its responsiveness and problem solving accuracy as components fail. In addition, the spatial distribution of components in a physically distributed problem solving network greatly reduces the possibility of massive component failure due to local environmental conditions.

Finally, there is the potential for improved performance due to parallelism. The existence of multiple problem solving nodes means that those nodes can be applied to diverse aspects of the problem.

Unfortunately, the limited interaction between nodes and natural distribution of information among nodes can significantly diminish the effective amount of parallelism in the network.

1.2.2 Comparison with parallel processing.

The most significant difference between parallel processing and distributed problem solving is in the interaction between problem solving components. Unlike distributed problem solving, interaction in a parallel processing system is relatively unrestricted. As a result, processing elements are more tightly coupled and share a much wider view of the state of processing than nodes in a distributed problem solving network.

The major emphasis of parallel processing is speed; executing programs as fast as possible. The data has no extensive initial distribution and can be readily relocated to accommodate parallel activity. The issue in parallel processing is determining exactly what this distribution should be to provide sufficient parallel activity with a particular parallel processing system.

In distributed problem solving, the emphasis is on dealing with the problems arising from restricted communication. The data is naturally distributed among the nodes in the network -- often in a manner ill-suited to a high degree of parallel node activity. As a result, the style of problem solving used in distributed problem solving differs significantly from that used in parallel processing.

1.2.3 Comparison with distributed processing.

As noted by Smith and Davis, distributed problem solving differs from distributed processing in both the style of distribution and the type of problems addressed [SMIT81]. The distributed aspect of most distributed processing is the data, with emphasis placed on synchronized access to shared information and recovery of information in the event of errors or partial system failure. Control of processing activity is generally not distributed, and the processors perform little cooperation.

A distributed processing system typically has multiple, disparate tasks executed concurrently in the network. Shared access to physical or informational resources is the main reason for interaction among tasks. The goal is to preserve the illusion that each task is executing alone on a dedicated system by having the network operating system hide the interactions and conflicts among tasks in the network. Coordination by the network operating system takes the form of a compromise between the various conflicting tasks based upon some measure of the importance of each task.

On the other hand, a distributed problem solving network performs only a single task. Instead of hiding the existence of other nodes in the network, each node is made explicitly aware of the other nodes at the problem solving level. Coordination involves deciding how best to perform the problem solving task given the limited resources of the network, rather than selecting between diverse competing tasks.

1.2.4 Comparison with cooperating experts.

Networks of cooperating nodes are not new to artificial intelligence. However, the relative autonomy and sophistication of the problem solving nodes sets distributed problem solving networks apart from Hewitt's work on the actor formalism, Kornfeld's EITHER language, Lenat's BEINGS system, and the augmented Petri nets of Zisman [HEWI77, KORN79, LENA75, ZISM78].

For example, in the cooperating experts style of system, knowledge is compartmentalized so that each "expert" is a specialist in one particular aspect of the overall problem solving task. An expert has little or no knowledge of the problem solving task as a whole or of general techniques for communication and cooperation. As a result, the expert cannot function outside the context of the other experts in the system nor outside specific communication and cooperation protocols specified in advance by the system designer.

In the view of distributed problem solving networks taken in this research, each node possesses sufficient overall problem solving knowledge that its particular expertise (resulting from a unique perspective of the problem solving situation) can be applied and communicated without assistance from other nodes in the network. This does not imply that a node functions as well alone as when cooperating with other nodes -- internode cooperation is often the only way of developing an acceptable solution -- but every node can at least formulate a solution using only its own knowledge. An argument against the distribution of expertise will be made in the next section.

Each node in the distributed network also possesses significant expertise in communication and control. This knowledge frees the network from the bounds of designed protocols and places its nodes in the situation of developing their own communication and cooperation strategies.

A second difference between cooperating experts and distributed problem solving networks is node motivation. Simply stated, cooperating experts are externally-directed in their behavior. The expert awaits receipt of a message, performs activities based upon that message, communicates results of those activities, and awaits receipt of a new message. The externally-directed approach has been advocated by Feldman, Hewitt, and Smith [FELD79, HEWI77, SMIT78]. As viewed in this research, each node in a distributed problem solving network can be self-directed in its activity, initiating activities in response to changes in its local environment. These self-directed activities can be modified through interactions with other nodes, but if a node does not receive an appropriate message from another node, it is able to continue with its locally initiated activities using whatever data are available at that time. As will be discussed in Chapter III, a node can also choose whether to engage in activities suggested by other nodes or pursue its own best interests. The importance of such self-motivation in program modules has also been suggested by Fox [FOX81].

The result of these differences is that the distributed problem solving network can tolerate changes in node activity, network connectivity, node and communication failure, and error in data and control.

1.2.5 The argument against "distributed experts".

The cooperating experts style of problem solving has become a useful metaphor in artificial intelligence problem solving and, on initial consideration, appears to be a reasonable structure for physically distributed problem solving. Each expert is placed at one (or a few) nodes in the problem solving network, and their interactions are routed over the communication channel. The cooperating experts system is now a "distributed experts" network.

But a distributed experts network is not necessarily an effective problem solving decomposition for a physically distributed problem solving application. A distributed experts network, like its cooperating experts forebear, is essentially functionally decomposed. Particular problem solving expertise translates into particular problem solving activities to be performed at each node. Unfortunately, physically distributed problem solving applications are often spatially decomposed in a manner ill-suited to a functionally distributed solution. Each node may possess the information necessary to perform a portion of each function, but insufficient information to completely perform any function. Extensive communication would be required to redistribute the information to match the functional distribution.

Distribution of expertise to individual nodes can also lead to decreases in network processing performance. Davis and Smith observe that "any unique node is a potential bottleneck" [DAVI81]. If a single node has the expertise for a particular functionality that is in great demand, problem solving can be impeded. Similarly, any unique node is also a candidate for system failure. If the last node possessing a necessary expertise fails, the network will also fail. Redundancy in expertise is an important consideration in a distributed problem solving structure.

If each node were supplied all of the network expertise, it could perform as much of each function as possible, given its local low-level information, and through cooperation with other nodes aggregate a complete result of any desired function. Because most node interaction involves abstracted information, the amount of communication required for aggregating the complete results of node processing is potentially much less than is required for redistributing the low-level information in a functionally decomposed network. Because each node can potentially perform many functions, a node is less likely to be idle. Because each node has the long-term expertise of other nodes, a node is able to share the work of an overloaded node or provide the functionality of a failed node. In such a <u>spatially decomposed</u> network, a node is an "expert" not because of its designer-supplied, long-term expertise, but because of the short-term information it possesses by being at a particular location at a particular time. By providing each node the entire complement of long-term expertise, the full range of functional decompositions (including distributed experts) are available to the network to be selected by its organizational structure. This is an instantiation of McCulloch's concept of the <u>redundancy of potential command</u> in which "the possession of the necessary urgent information constitutes authority in that part possessing the information" [McCU65]. Any node might become the center of a particular function, depending on the problem solving situation.

Providing full expertise at each node is not unrealistic for computer networks. The cooperating experts model is based, in part, on group problem solving behavior of human experts. The typical metaphor is a group of human experts sitting around a table and working together to complete a complex task. With humans, the transfer of long-term expertise is an arduous process, often involving years of training and experience. Individuals with sufficient expertise to deal with all aspects of a complex task are difficult, if not impossible, to locate.

This difficulty in obtaining human expertise underlies many business organizational structures. Mundane tasks are transferred to less skilled workers in an attempt to maximize the amount of time that highly trained (and expensive) individuals spend using their expertise. If highly skilled workers were readily available at the same cost as unskilled workers there would be no economic need for such task transference. Each worker would perform both skilled and unskilled activities.

With computer systems the acquisition of long-term expertise is an arduous process, but the transfer of this expertise, once encoded, is simple (although, possibly requiring translation from one machine type to another). If the knowledge can be encoded at one node, it can be easily transferred to all the nodes. This suggests that computer organizations are more akin to business organizations operating where expertise is relatively inexpensive.

Although it is reasonable for a node to have the capability to perform any problem solving activity, it is unreasonable for the node to perform all activities. Determining the activities that are best suited to the current information available to each node is a major issue in network coordination.

The human group problem solving metaphor is also characterized by a fairly rich communication channel used by the participants. Experiments have shown that even simple tasks become difficult when the communication channel between human problem solvers is restricted [CHAP75]. As discussed in Section 1.2.1, restricted communication is a basic and unavoidable characteristic of distributed problem solving networks. The problems of restricted node interaction and network coordination in the distributed experts network are not addressed by the cooperating experts problem solving model.

On the other hand, the cooperating experts style of problem solving is suited to the problem solving activity within a node itself. The functional decomposition of the cooperating experts can help to reduce the control complexity within a node. A form of the cooperating experts style of problem solving, based on a generalization of the Hearsay-II architecture, is developed as the architecture of a node in Chapter II.

1.2.6 Describing distributed problem solving networks.

We must be cautious when describing distributed problem solving networks. These are multilevel systems. Each node is itself a complex system and the entire distributed problem solving network may be a component of a still larger system. The boundaries of the system often depend on how the system is viewed.

From a reductionist perspective, a distributed network can be viewed as a system that is decomposed over a number of nodes, each of which is a part in the overall network. From a constructionist perspective, a distributed network is a society of nodes, where each node is an individual system. While both perspectives view the same reality, the reductionist viewpoint tends to encourage a search for appropriate ways of pulling apart existing centralized systems. The constructionist viewpoint tends to encourage a search for ways of organizing individually complete systems into a society of cooperating nodes [LESS80b]. The constructionist view will be used throughout the dissertation, with the term "network" referring to the overall distributed system and "system" referring to an individual node in the network.

Mixing the two viewpoints can lead to conceptual inconsistencies. For example, viewing a distributed problem solving network as a society of nodes which are to be coordinated using some suitably decomposed, centralized problem solving technique can lead to forgetting that the control technique must be distributed among the nodes -- the technique itself becoming a distributed problem solving problem and needing itself to be coordinated.

1.3 The Major Contributions

The major contributions of this research can be grouped into four categories:

1. development of the philosophy of organizational structuring as a means of achieving coordinated node activity in distributed problem solving networks;

2. a major design and implementation effort in creating a unique experimental testbed that is both parameterized and fully-instrumented for realistic exploration of coordination techniques for distributed problem solving networks;

3. concrete implementation of a computational framework for the philosophical ideas on organizational structuring in the testbed;

4. an initial exploration demonstrating the flexibility of the testbed, the sophisticated internal coordination capability of a testbed node and that an organizational structure can be used to bias the activities of the network. These results suggest that different organizational structures are indeed appropriate in different problem solving situations.

The following sections survey these contributions and indicate where each is elaborated in subsequent chapters.

1.3.1 Philosophical ideas.

Functionally accurate, cooperative distributed problem solving.

A straightforward approach to constructing a distributed problem solving network is to ensure that all nodes have mutually consistent views of the problem. An attempt is made to provide each node with the information needed to perform its portion of the overall problem solving task. If a node is missing some information, it is obtained from another node as a complete, self-contained result. If the problem cannot be partitioned so that each node works on a relatively independent portion of the problem, a high degree of coupling between nodes is required. In a distributed problem solving environment, such tight node coupling introduces significant communication and computation costs.

Fortunately, the requirement for complete consistency can be relaxed if the style of problem solving is changed. Instead of always producing complete, self-contained results, each node produces tentative results which may be incomplete, inaccurate, incorrect, and inconsistent with the tentative results produced by other nodes. An iterative, coroutine style of node interaction is used in which the nodes exchange their tentative partial results to cooperatively converge to acceptable network behavior in the face of locally incomplete and inconsistent views of the problem. This approach is called functionally accurate, cooperative [LESS81].

Functionally accurate, cooperative problem solving is discussed in Section 2.2.

Functionally accurate, cooperative control.

Coordinating the activity of the nodes in a distributed problem solving network is difficult because the information needed to make control decisions is distributed throughout the network. In a functionally accurate, cooperative network, node coordination is even more difficult due to the presence of inaccurate and inconsistent information. To cope with this situation, we can apply the functionally accurate, cooperative approach to the problem of coordinating node activity. Instead of ensuring that the activity decisions made by each node are completely consistent with those made by other nodes, nodes determine their activities using incomplete and inconsistent local views of the state of problem solving in the network. If coordination errors are made, such as a node undertaking an activity before receiving all input information from another node or a node selecting an activity which is globally inappropriate, the tentative nature of the functionally accurate, cooperative style of problem solving reduces their impact. The intent is that the additional communication and computation caused by these local control decisions is less than would be required to maintain complete and consistent views of network problem solving activity and is, to some degree, offset by increases in parallel node activity.

In order to be successful, functionally accurate, cooperative control must achieve the following conditions:

<u>coverage</u>	— any given portion of the overall problem must be included in the activities of at least one node;
<u>connectivity</u>	— nodes must interact in a manner which permits the covering activities to be developed and integrated into an overall solution;
<u>capability</u>	— coverage and connectivity must be achievable within the communication and computation resource limitations of the network.

Sections 3.1 through 3.6 lay the groundwork for functionally accurate, cooperative control which is discussed in Section 3.7.

<u>Organizational structuring as a framework for control</u>.

Even using functionally accurate, cooperative control there are insufficient communication and computation resources to permit each node to consider the network-wide ramifications of all of its possible local control decisions. An organizational structure, developed and maintained by the network, can be used to limit the range of decisions which must be considered by a node and still accomplish an acceptable level of coverage, connectivity, and capability.

The organizational approach to coordinating a distributed problem solving network splits the coordination problem into two concurrent activities:

1. construction and maintenance of the organizational structure;
2. continuous local elaboration of this structure into precise activities by the nodes.

If the coordination problem is complex enough, development and maintenance of the organizational structure may itself need to be coordinated by developing levels of meta-organizational structure.

Included in the organizational structure are control decisions that are not quickly outdated and that pertain to a large number of nodes. The organizational structure represents general "ballpark" control decisions which are dynamically tailored by the local, functionally accurate, cooperative control decisions of the nodes. To be effective, the organizational structure must be based on the dynamics of the problem solving situation and the internal characteristics of the network. As these change, the network may need to change its organizational structure in order to maintain its effectiveness.

But organizational change can have negative consequences. Change disrupts the progress of problem solving activities and the flow of information in the network. Information and partially completed tasks may have to be transferred among nodes, consuming valuable communication resources. Processing time can also be lost as nodes await relocation of tasks and information and as organizational activities override basic problem solving. Even small changes in one part of the organizational structure can have significant effects on the network as a whole.

In order to enhance the positive aspects of organizational change while reducing its negative consequences, the basic organizational framework must be capable of providing (adapted from Kast and Rosenzweig [KAST74]):

- o enough stability to allow effective problem solving;
- o enough continuity to ensure orderly change;
- o enough adaptability to react appropriately to external demands as well as changing internal conditions;
- o enough innovativeness to allow the organization to initiate change when conditions warrant.

Of course, the network still has the problem of developing and maintaining the organizational structure.

Section 3.8 introduces the use of an organizational structure as a control framework and the dynamics of organizational change.

Node skepticism.

What happens if the organizational responsibilities of a node, as determined by the organizational self-design process, are incongruous with the potential actions of a node? This situation can arise if the problem solving situation changes faster than the organization can restructure itself or when organizational self-design decisions are made using incomplete and inaccurate information (which is often the case in a functionally accurate, cooperative network).

One approach is to be dogmatic about the organizational structure and force all nodes to abide by its dictates. An alternative approach developed in this research views the organizational structure as a guide, not a rigid structure. The network is viewed as a society of skeptical nodes working within the framework of an organizational structure but always alert for signs of trouble. The organizational structure provides a basis for global coherence to what would otherwise be an anarchic network.

A skeptical node's local activity decisions are constantly pulled in two directions: toward the responsibilities specified by the organizational structure and toward the activities suggested by its local data and interactions with other nodes. The tension between these two directions can lead to an increase in the network's ability to tolerate organizational control errors. If a node's organizational responsibilities are inappropriate to its potential activities, the node can proceed with locally generated activities. Similarly, organizational responsibilities can be ignored by nodes which possess strong information to the contrary; a node with a unique perspective is not necessarily stifled by an uninformed majority. A strong mismatch between organizational responsibilities and locally generated activities is an indication of the need for organizational change. The ability of a node to balance its organizational responsibilities with its locally generated activities and to sense when a mismatch between them has occurred is a key aspect of this research.

The degree of skepticism exhibited by a node should dynamically change according to the node's certainty as to the network importance of its own locally generated activities: as the certainty of a node's own perspective decreases, it should become more receptive to organizationally generated responsibilities; as a node becomes convinced of its own approach, it should become more skeptical of organizational responsibilities which are in conflict with that approach.

In distributed problem solving networks composed of large numbers of nodes, the existence of idiosyncratic degrees of skepticism can further increase the robustness of the network. In situations where there exist two competing approaches (one advocated by the organization and one apparent to some of its members) individual variances in node skepticism will ensure that both approaches are pursued by the organization. The approach apparent to a portion of the node population is implicitly pursued, without the cost of making an explicit organizational decision. Of course this robustness comes at the price of uncontrolled expenditure of resources by the skeptical nodes.

The motivation for skeptical nodes and a look at similar behaviors in various "natural" systems is in Section 3.9.

1.3.2 Implementation of the Distributed Vehicle Monitoring Testbed.

An evaluation of the ideas outlined in the previous section required a complex distributed problem solving network to coordinate. Because distributed problem solving networks are both difficult to analyze formally and very expensive to construct and modify for empirical evaluation, an intermediate approach of simulating the network at a detailed level was taken. The result of this implementation effort is the distributed vehicle monitoring testbed.

The nodes in the testbed attempt to identify, locate, and track patterns of vehicles moving through a two-dimensional space by listening to the acoustic signals generated by the vehicles. This task was selected because it is a realistic and extremely flexible distributed problem solving task. As a passive interpretation task, distributed vehicle monitoring does not modify the environment, eliminating any interaction between network problem solving and the future behavior of the sensed vehicles.

Distributed vehicle monitoring and the testbed are described in Chapter II.

1.3.3 Testbed mechanisms for implementing the philosophical ideas.

The philosophical ideas developed in the dissertation require each node to make sophisticated local activity decisions based on both external and internal criteria. Each node must be able to reason about what it is doing and how its activities relate to the network as a whole. It must be sensitive to changes in the problem solving situation, making different decisions in different situations.

These prerequisite characteristics are not well developed in existing interpretation systems. Therefore, realization of the philosophical ideas required several architectural developments, which are surveyed below.

Goal-directed Hearsay-II architecture.

Elaboration of the organizational responsibilities by each node requires significant local planning capabilities. For a number of reasons (discussed in Chapter II) Hearsay-II was the problem solving architecture of choice for each node in the experimental testbed. Unfortunately, control of problem solving activity in the original data-directed Hearsay-II architecture is very limited. Hearsay-II reacts to changes in its developing solution by proposing a set of plausible actions. These actions are scheduled for execution based on an estimate of their individual potential for improving the solution.

While simple and economical, this form of control has three major limitations:

1. lack of information as to the intended purpose of each potential activity;

2. an instantaneous, single action view of potential activity;

3. a passive, "wait and see" approach to the generation of needed information.

Hearsay-II's <u>scheduler</u> is not aware of the role a particular action plays in the overall problem solving process or of other potential actions that could achieve the same desired result. Similarly, the scheduler cannot consider as a unit a sequence of actions which produce a particular result, terminate such a sequence if a preliminary action is unsuccessful, nor compare alternative sequences for producing the same result. In addition, Hearsay-II cannot initiate actions to produce results needed by other proposed actions or requested by other nodes, but must await bottom-up synthesis of the information through normal data-directed problem solving activity.

If a Hearsay-II system is to serve as the architecture for each node in a distributed problem solving network, more sophisticated problem solving control capabilities are required. An extended Hearsay-II architecture which integrates both <u>data-directed</u> and <u>goal-directed</u> control into a common framework was developed and implemented in the testbed to provide these capabilities. Goal-directed control permits nodes to affect the processing of other nodes, not only through the transmission of information (hypotheses), but also through transmission of goals which request the generation of particular information. Goals reside on a separate blackboard called the goal blackboard, which parallels the dimensionality of the original Hearsay-II data blackboard. Included in the goal-directed Hearsay-II architecture is a full-fledged planner for initiating and directing the activities within each node. The integrated control framework also improves the scheduling decisions made within each node by providing the scheduler with additional information about the relationship among proposed actions. This additional information also helps in the planning of communication activities with other nodes.

The details of the goal-directed architecture and the local node planner are described in Sections 4.1 through 4.3.

Organizational roles and blackboard.

The planner, scheduler, and communication knowledge sources at each node are influenced by the network organizational structure, which is contained on the organizational blackboard (Figure 2). The organizational structure is represented by data structures called organizational roles that indicate the areas for which the node is responsible for problem solving activity, the responsibilities of other nodes, the areas in which the node is to send results to particular nodes or receive results from particular nodes, and the areas in which the node is expected to accept goals from particular nodes or to request the achievement of goals from particular nodes. The relationship between the organizational blackboard and the local control components of the node is described in Section 4.4.

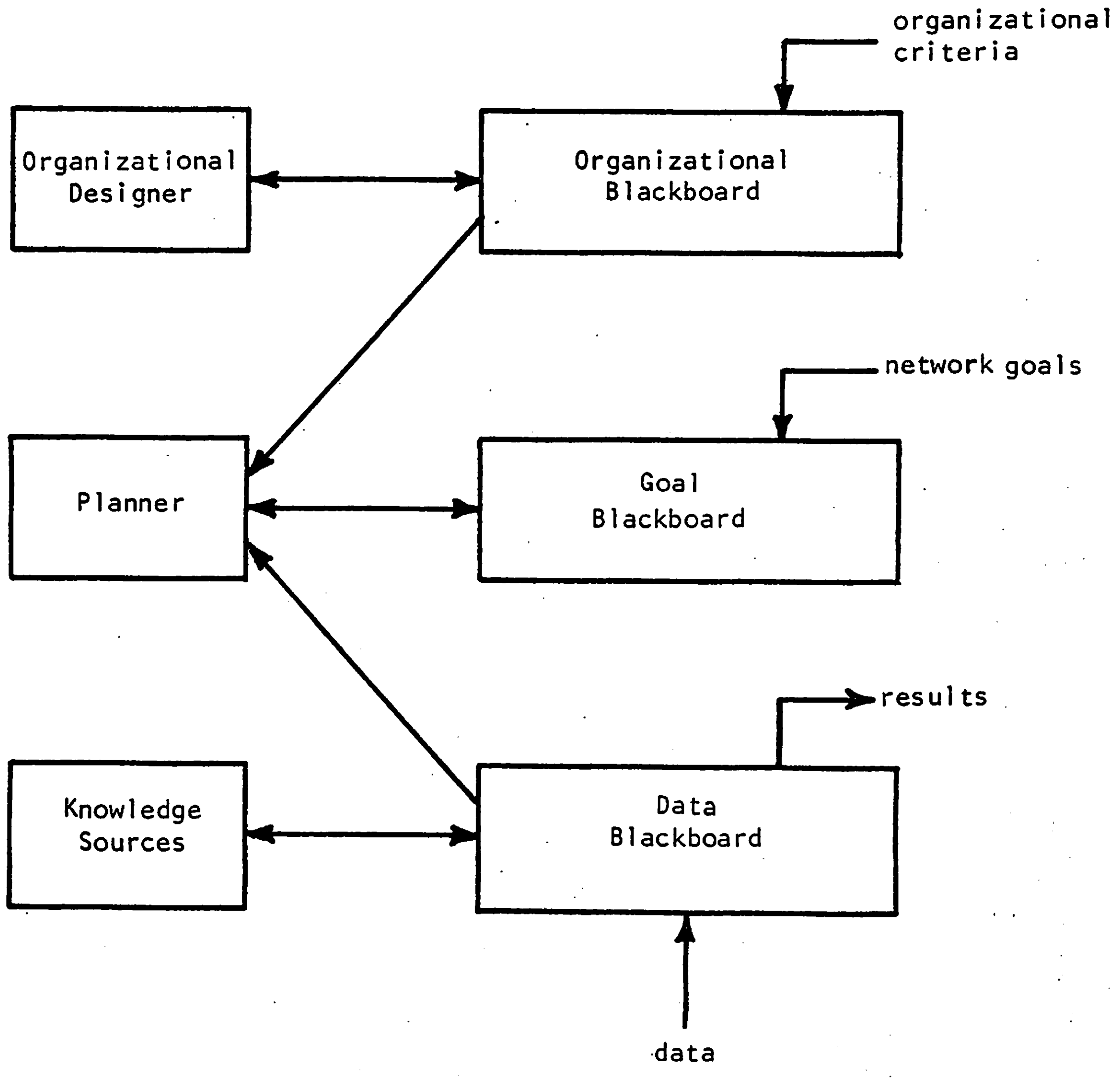


Figure 2: Basic Node Architecture.

The major components of a Distributed Vehicle Monitoring Testbed node can be grouped into three levels. The knowledge sources and data blackboard form the basic problem solving level; the planner and goal blackboard form the node control level; and the organizational designer and organizational blackboard form the organizational structuring level. The "heart" of each node is the planner which integrates organizational criteria, network goals, and the current state of problem solving into precise activities for the node to perform.

Implementation of the goal-directed Hearsay-II architecture, the local node planner, and the organizational blackboard provide the computational framework necessary for implementing a distributed organizational designer. However, a distributed organizational designer was not implemented. It remains the object of continuing work discussed in Section 6.3.

1.3.4 Initial explorations using the testbed.

A set of experiments illustrating the flexibility of the distributed vehicle monitoring testbed and the control capabilities of the goal-directed node architecture were performed. These experiments show that the mechanisms do indeed work and can have a significant positive effect on problem solving performance.

To indicate the need for organizational change, a number of additional experiments were performed with the testbed. These experiments involved a mixture of static organizational structures, network sizes, and environmental scenarios.

The experiments indicate that different organizational structures and cooperation strategies do make a difference in network problem solving performance. These experiments also illustrate the problem of distraction, where one node communicates incorrect information that temporarily draw the other nodes away from working on the correct solution. The experiments suggest that organizational structures which divide the effort spent on distracting information among nodes are less affected by the distracting information.

The details of these experiments are the subject of Chapter V.

1.4 Navigational Aids for the Reader

The major chapters of the dissertation can be partitioned into four groups. The first group (Chapters I and III) develops the basic philosophy of organizational self-design as a means of coordinating activity within a distributed problem solving network. The second group (Chapter II) describes the basic structure of the Distributed Vehicle Monitoring Testbed and the characteristics of its underlying problem domain. The third group (Chapter IV) describes the design and implementation of the sophisticated local control component of a testbed node which permits organizational structuring decisions to affect its local activity decisions. The fourth group (Chapter V) presents experimental results obtained using the testbed.

Chapter I presented an overview and the main ideas of this research, as well as delimiting the notion of distributed problem solving, describing its major characteristics, and contrasting it with other problem solving forms.

Chapter II starts with a description of the problem of distributed vehicle monitoring, the task domain used throughout the dissertation. It then explains that, due to the particular characteristics of distributed problem solving networks, an unconventional style of problem solving, called functionally accurate, cooperative, is needed for this task. The last section of this chapter introduces the distributed vehicle monitoring testbed, a unique and versatile research tool for the empirical evaluation of alternative distributed problem solving designs. The basic architecture of the testbed and its parameterization and measurement capabilities are described.

Chapter III begins with the problem of controlling the activities of a single node in the distributed vehicle monitoring testbed and then details the problem of coordinating the activities of multiple nodes. Two diverse approaches to control are discussed and a general view of coordination that integrates both approaches is developed. The last sections of Chapter III introduce the use of an organizational structure to serve as a framework for making local node control decisions and develop the basic philosophy of organizational self-design as a means of achieving coordinated activity in a distributed problem solving network. A discussion of node skepticism rounds out the chapter.

Chapter IV details the implementation of a computational framework for organizational self-design in the distributed vehicle monitoring testbed. This chapter first describes the goal-directed Hearsay-II architecture developed to provide the sophisticated local control necessary to implement the ideas of Chapter III. A key aspect of the implementation is the use of a nonprocedural and dynamically variable specification of the behaviors of each local node's planner, its scheduler, and its communication knowledge sources. Next, the testbed implementation of the goal-directed architecture and the local node planner is presented. The chapter concludes with a discussion of how organizational structuring decisions and node skepticism can be used to modify these behavioral specifications.

Chapter V presents the experimental evidence demonstrating the flexibility provided by the behavioral specifications, the capabilities of the goal-directed architecture, and the impact of organizational structuring decisions on the local control component of a testbed node.

Chapter VI summarizes the contributions of this research and suggests areas for further work. The potential contributions of organizational theories of business, biological, and social systems for distributed problem solving networks are also surveyed.

For the reader interested in the philosophical issues of organizational self-design as a coordination technique, a straight dosage of Chapters I through III followed by Chapter VI is suggested. For the implementer, Chapter II followed by Chapter IV bypasses most of the discussion while detailing what was done. For the reader interested only in results, Chapter II followed by Chapters IV and V is the recommended abridgement.

They tore my legs off, and they threw them over there! Then they took my chest out, and they threw it over there!

— the Scarecrow in the movie The Wizard of Oz

CHAPTER II
DISTRIBUTED VEHICLE MONITORING AND THE TESTBED

This chapter begins with a description of a specific distributed problem solving application, distributed vehicle monitoring. This application is representative of many distributed problem solving applications which are poorly-suited to conventional distributed-system methodologies. The second section explains why a new problem solving approach, called functionally accurate, cooperative, is more appropriate for the distributed vehicle monitoring task.

Distributed vehicle monitoring also serves as the task domain for the distributed vehicle monitoring testbed: a flexible and fully-instrumented research tool constructed for the empirical evaluation of alternative coordination techniques for distributed problem solving networks. Its structure is detailed in the last sections of this chapter and is continued in Chapter IV.

2.1 The Vehicle Monitoring Task

Vehicle monitoring is the task of generating a dynamic, area-wide map of vehicles moving through a geographical area. Acoustic sensors are distributed over the area to be monitored and provide the input data for the network. Each sensor includes the actual acoustic transducer, low-level signal processing hardware and software, and communication equipment necessary to transmit the processed signals to a high-level (symbolic) processing site. As a vehicle moves through the monitoring area, it generates characteristic acoustic signals. Some of these signals are detected by nearby sensors which detect the frequency and approximate location of the source of the signals. An acoustic sensor has a limited range and accuracy, and the raw data it generates contains a significant amount of error. Using data from only one sensor can result in "identification" of non-existent vehicles and ghosts, missed detection of actual vehicles, and incorrect location and identification of actual vehicles. To reduce these errors, information from various sensors must be correlated over time to produce the answer map.

2.1.1 Vehicle monitoring task processing levels.

The vehicle monitoring problem has been investigated by a number of researchers [LACO78, NII78, SMIT78]. Typically, processing has been divided into a data hierarchy containing five identifiable levels (Figure 3).

At the lowest level are signals. Signals are the output of low-level analysis of the sensory data and each signal includes the frequency, approximate position, time of detection, and belief (based partly on signal strength and sensor quality) of the acoustic signal as well as the identity of the detecting sensor. Signals are the basic input to the problem solving network.

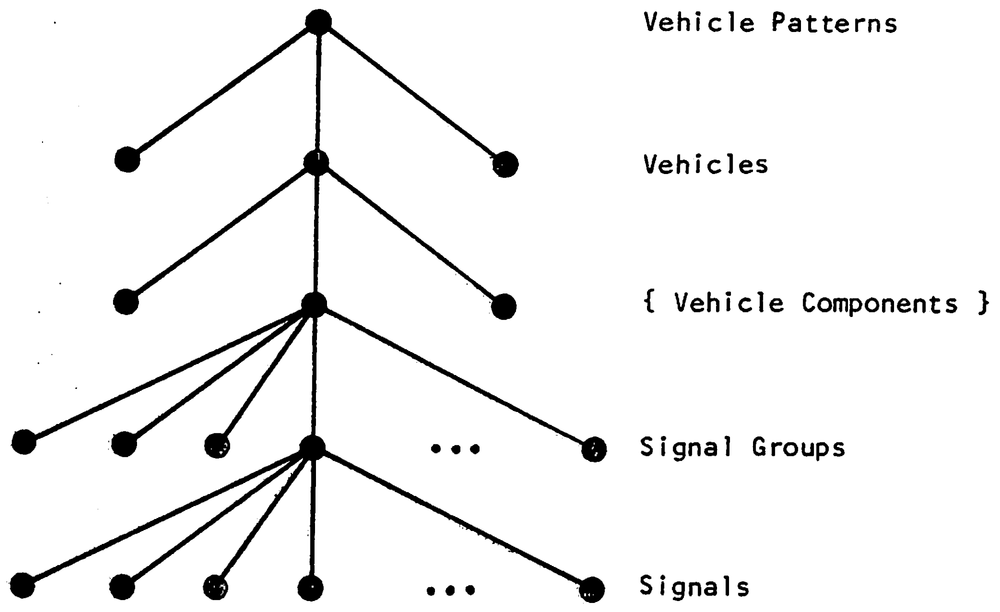


Figure 3: Vehicle Monitoring Task Processing Levels.

Forming a vehicle pattern from sensory signals involves combining harmonically related signals into signal groups. Various signal groups can collectively indicate a particular piece of machinery (component) on a vehicle, and these components can be used to identify the type of vehicle. Specific vehicle types with a particular spatial relationship among themselves form a vehicle pattern. (The vehicle component level is omitted in the testbed, with vehicles formed directly from signal groups.)

At the next level in the data hierarchy are signal groups. A group is a collection of harmonically related signals (emanating from a common source). Each group includes the fundamental frequency of the related signals and its approximate position, time (based on the time of detection of the related signals), and belief (a function of the beliefs and characteristics of the related signals).

Groups are combined to form vehicle components. A <u>component</u> consists of groups associated with a piece of machinery on the vehicle. Each component includes the identity of the component and its time, approximate position, and belief calculated from the lower data levels. For simplicity this level was omitted by Smith [SMIT78] and will be similarly omitted here.

Vehicles are the next level in the data hierarchy. A <u>vehicle</u> consists of a collection of components (or in our case groups) associated with a particular vehicle. Vehicles include the identity of the vehicle and its time, approximate position, and belief.

At the highest level of processing are vehicle patterns. A <u>pattern</u> is a collection of particular vehicle types with a particular spatial relationship among them. Patterns can provide strong constraints between distant nodes. A pattern includes the identity of the pattern and its time, approximate position, and belief. A single vehicle can be a pattern.

The answer map is produced from the vehicle patterns based upon their beliefs and continuity over time. There are two types of answer map distribution: one where a complete map is to be located at one or more answer sites within the monitored area and one where a partial (spatially relevant) map is to be located at numerous sites within the area. In distributed vehicle monitoring tasks such as air or ship traffic control, both distributions of the answer map may be required. Each node would use its portion of the distributed map to control nearby vehicles, while the complete map would be produced for external monitoring of the network.

2.1.2 Centralized vehicle monitoring.

One means of generating the map is to have each acoustic sensor transmit its sensory data to a central computer (Figure 4). The central computer processes the data and produces the map which is distributed to the answer sites. This approach has the advantage of performing all problem solving activity at a single site using conventional methods for processing and control. In addition, there is no communication involved in the problem solving process itself.

However, the centralized approach has several disadvantages:

- The massive amount of environmental data collected by the various sensors must be transmitted to the central computer. Even if the computer is located in the center of the monitored area, processors located on the periphery must send their data half the distance across the area. Due to the distance, transmission of sensory data is subject to propagation delay which can, in turn, degrade network responsiveness. Since communication cost is a function of both amount and distance, collection of sensory data can be expensive.

- The answer map must be distributed to the appropriate sites. However, the communication requirements here are small compared to data collection.

- Central computer failure means complete network failure.

- Communication channel failure means possible network failure.

2.1.3 Distributed vehicle monitoring.

A second approach to the problem, termed distributed vehicle monitoring, is to place a number of processing nodes throughout the area to be monitored (Figure 5). The acoustic sensors transmit their sensory data to nearby processing nodes. Nodes interact with other nodes to construct the answer map. Depending on the desired distribution of the

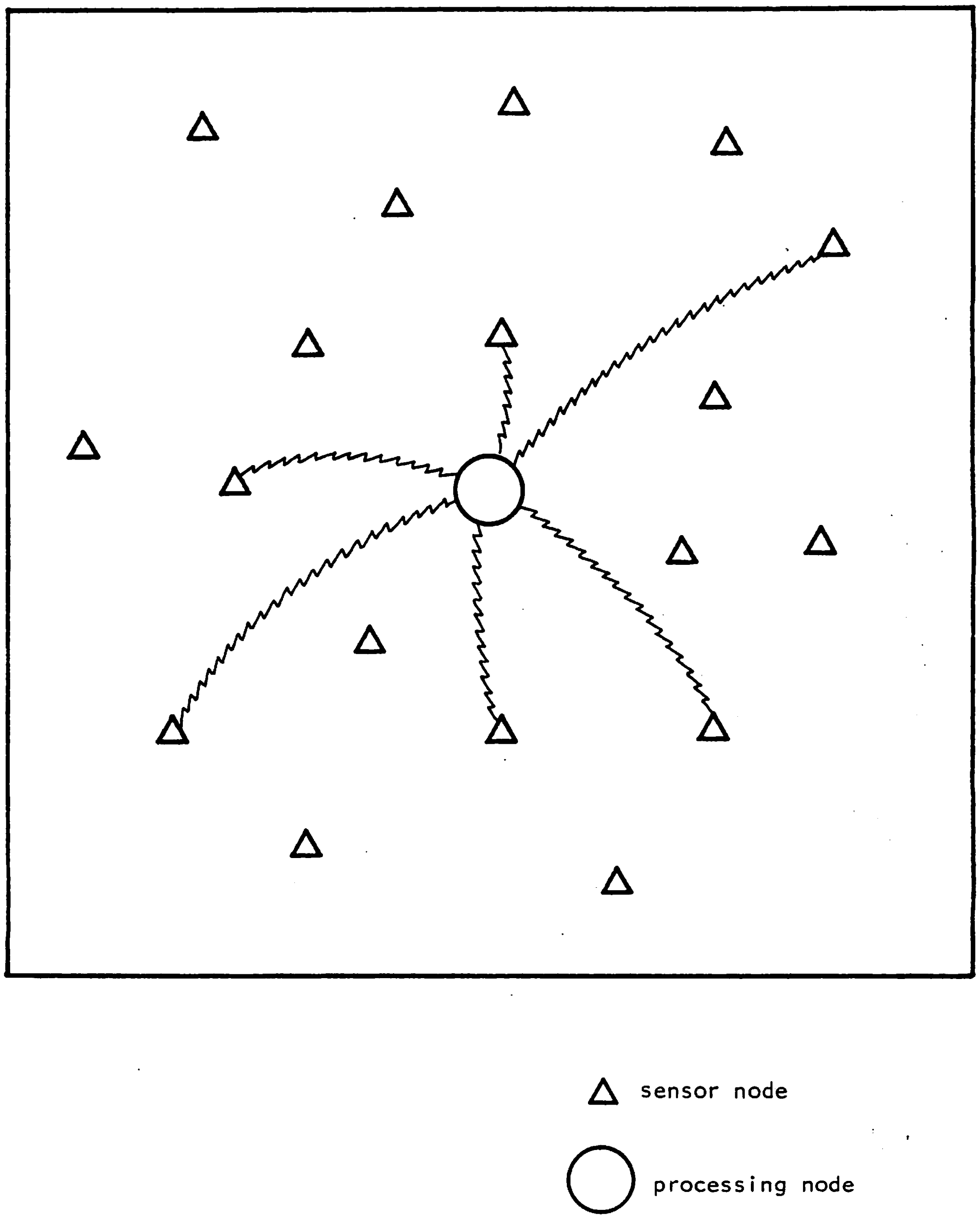


Figure 4: Centralized Vehicle Monitoring.

In centralized vehicle monitoring, all sensory data is communicated to a central processing site that performs all problem solving activities and transmits the resulting answer map to the appropriate locations.

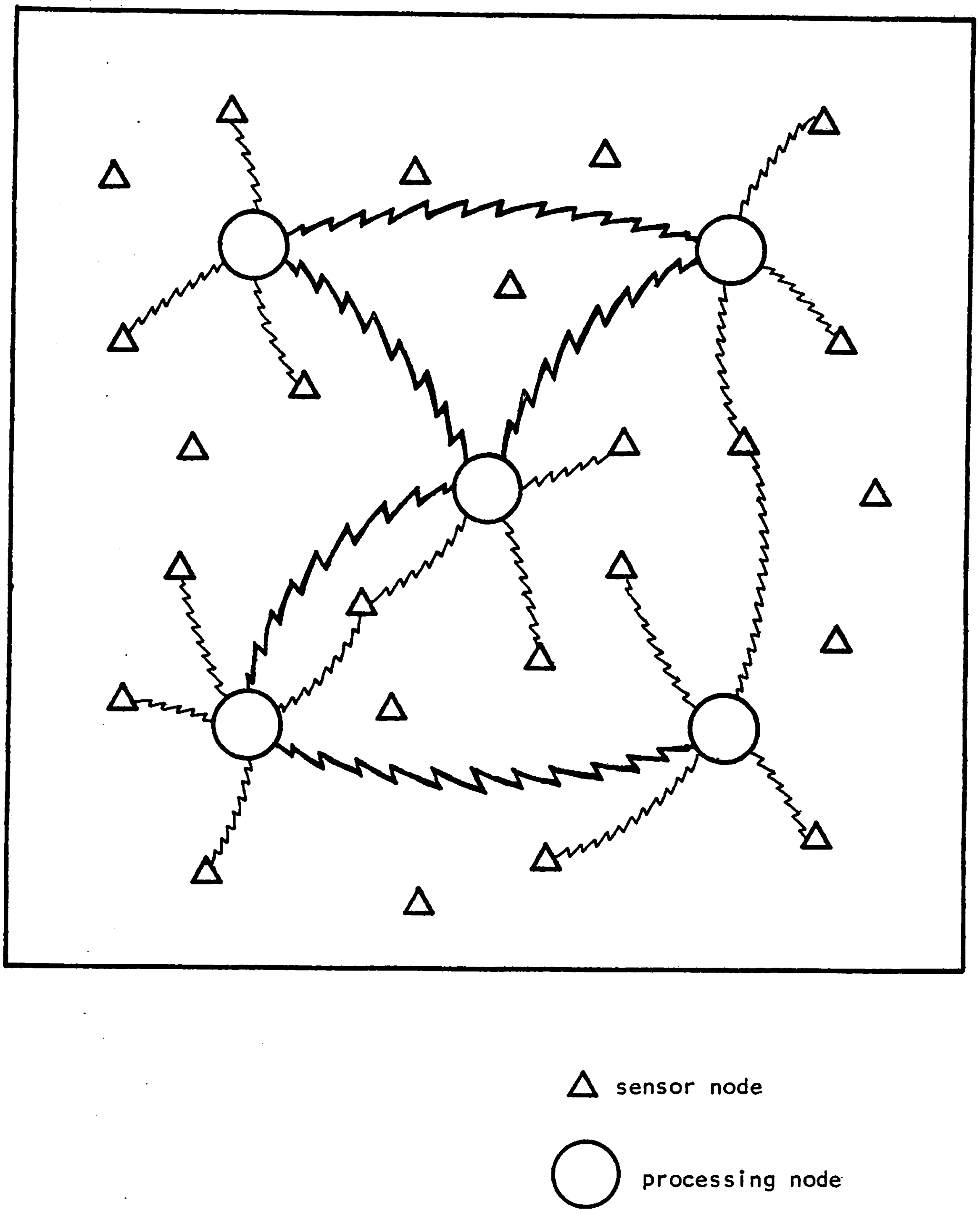


Figure 5: Distributed Vehicle Monitoring.

In distributed vehicle monitoring, sensory data is only communicated to nearby processing nodes. The nodes must cooperate to construct and possibly redistribute the answer map.

answer map, the complete map is generated at one or more answer nodes within the monitored area or a partial (spatially relevant) map is generated at numerous processing nodes within the area.

The amount and distance that data is communicated in the distributed approach are potentially much smaller than in the centralized approach. Much of the environmental data needed by a node is available from nearby sensors. Similarly, the distributed answer maps can be generated near the answer sites. The distributed approach is potentially more reliable than the centralized approach, degrading gracefully in coverage, accuracy, and timeliness as nodes and communication channels fail. Finally, the distributed approach may be more responsive due to the closeness of problem solving to the sensory data and to parallelism.

However there are disadvantages:

- o Because each node only receives data from nearby sensors, techniques for problem solving using partial environmental information are needed. This will be discussed in Section 2.2.
- o Techniques for coordinating this type of problem solving in distributed environments must be developed.
- o Communication is required to perform and coordinate the distributed problem solving.

2.1.4 Why distributed vehicle monitoring?.

Distributed vehicle monitoring has four characteristics making it an ideal initial problem domain for research on distributed problem solving.

First, distributed vehicle monitoring is a natural task for a distributed problem solving approach, since the acoustic sensors are located throughout a large geographical area. The massive amount of sensory data that must be reduced to a highly abstract, dynamic map seems appropriate for a distributed approach.

Second, distributed vehicle monitoring can be formulated as an interpretation task in which information is incrementally aggregated to generate the answer map. Nilsson has termed systems with this characteristic <u>commutative</u> [NILS80a]. Commutative systems have the following properties:

1. Actions that are possible at a given time remain possible for all future times.
2. The system state that results from performing a sequence of actions that are possible at a given time is invariant under permutations of that sequence.

Commutativity allows the distributed vehicle monitoring network to be very liberal in making tentative initial vehicle identifications, since generation of incorrect information never precludes the generation of a correct answer map. Without commutativity, the basic problem solving task would be much more difficult. (Commutativity is an important property of functionally accurate, cooperative distributed systems to be described in the next section.)

Although the generation of the answer map is commutative, controlling node activity is not. Here we enter the realm of limited time and limited resources. If a crucial aspect of the answer map is not immediately undertaken by at least one node in the network, the network can fail to generate the map in the required time. In the determination of node activities, mistakes cause the loss of unrecoverable problem solving time and can therefore eliminate the possibility of arriving at a timely answer map. If the nodes and sensors are mobile, their placement adds another non-commutative aspect to the distributed vehicle monitoring task. A misplaced node or sensor can require substantial time to be repositioned. (The complexities of mobile nodes and sensors are beyond the scope of this dissertation.)

Third, the complexity of the distributed vehicle monitoring task can be easily varied. For example:

- o Increasing the density of vehicle patterns in the environment increases the computational and communication load on the network.
- o Increasing the similarity of the vehicles and patterns known to the network increases the effort required to adequately distinguish them.
- o Increasing the amount of error in the sensory data increases the effort required to discriminate noise from reality.

Fourth, the hierarchical task processing levels coupled with the spatial (x,y) and temporal dimensions of the distributed vehicle monitoring task permit a wide range of spatial, temporal, and functional network decompositions. Node responsibilities can be delineated along any of these dimensions.

2.2 The Functionally Accurate, Cooperative Approach

Due to the large amount of error in the sensory data and the small number of sensors reporting to each node, no single node in the distributed vehicle monitoring network may possess sufficient sensory information to accurately determine the movement and identity of vehicles in its area. Furthermore, communication limitations preclude the exchange of a significant amount of sensory information among nodes. Therefore, the nodes must cooperate by exchanging tentative and possibly incorrect partial results with one another. For example, each node's tentative vehicle identifications can be used to indicate to other nodes the areas in which vehicles are more likely to be found and the details (vehicle type, approximate location, and speed) of probable vehicles. In addition, consistencies between these tentative identifications serve to reinforce confidence in each node's identifications. Cooperation is not only appropriate at the vehicle level, but at all the processing levels.

This type of node cooperation differs significantly from conventional distributed-system design, which emphasizes the maintenance of correctness in all aspects of the distributed computation [KOHL81]. The conventional distributed processing network is structured so that a node rarely needs the assistance of another node in carrying out its activities. This conventional type of distributed processing decomposition is called <u>completely accurate, nearly autonomous</u>, because each node's algorithms operate on complete and correct information ("completely accurate") and because each node usually has in its local database the information it requires to complete its processing correctly ("nearly autonomous") [LESS81]. When such information is not available locally, a node requests another node to determine the required information, which is returned as a complete and accurate result. This form of node interaction is often implemented using asynchronous subroutine calls, in which one node is the master and the other is the slave.

This approach, however, is not suitable for applications such as distributed vehicle monitoring in which the algorithms and control structures do not match the natural distribution of data in the network. In these applications, a completely accurate, nearly autonomous network is expensive to implement because of the amount of communication and synchronization required to redistribute the data and results and to guarantee completeness and consistency of the local databases.

One way to reduce the amount of communication and synchronization is to loosen the requirement that nodes always produce complete and accurate results. Instead, each node produces tentative results which may be incomplete, incorrect, or inconsistent with the tentative results produced by other nodes. For example, a node may produce a set of alternative partial results based on reasonable expectations of what the missing data might be. This type of node processing requires a distributed problem solving structure in which the nodes cooperatively converge to acceptable answers in the face of incorrect, inaccurate, and inconsistent intermediate results. This is accomplished using an iterative, coroutine type of node interaction, in which nodes' tentative partial results are iteratively revised and extended through interaction with other nodes. A network with this problem solving structure is called <u>functionally accurate, cooperative</u> [LESS81]. "Functionally accurate" refers to the generation of acceptably accurate solutions without the requirement that all shared intermediate results be correct and consistent (as distinct from completely accurate networks). "Cooperative" refers to the iterative style of node interaction in the network.

The functionally accurate, cooperative style of processing can be characterized as problem solving in the presence of uncertainty. A node may be uncertain as to what input data it is missing, the missing values of the data, and the correctness, completeness, and consistency of the results of its processing and of the processing results received from other nodes. In order to resolve this uncertainty, a node must be able to:

- o detect inconsistencies between its tentative partial results and those received from other nodes;
- o integrate into its local database those portions of other nodes' results which are consistent with its results;
- o use the newly integrated results to make up for its missing input data so that its tentative partial results can be revised and enlarged.

Because consistency checking is such an important part of the functionally accurate, cooperative approach, errors resulting from hardware, communication, or processing can be handled as an integral part of the network problem solving process -- without the need for other error correcting mechanisms. Work by Baudet, Brooks, Brooks and Lesser, Fennel and Lesser, and Lesser and Erman has demonstrated such error correcting capability [BAUD78, BROO79, BROO83, FENN77, LESS80b].

The work by Fennel and Lesser, and Lesser and Erman is particularly interesting because of its historical relationship to functionally accurate, cooperative distributed problem solving networks. Fennel and Lesser describe a simulated parallel processing version of the Hearsay-II speech understanding system in which knowledge sources were executed in parallel, communicating via the blackboard. Normally regions of the blackboard were locked to avoid the inconsistencies associated with interleaved accesses and modifications by multiple knowledge sources. However in experiments without this locking, the system produced approximately the same results as with locking. This was an early indicator of Hearsay-II's potential for functionally accurate, cooperative problem solving.

Lesser and Erman report the results of the first distributed problem solving application of the Hearsay-II architecture. Again the speech understanding system was used, this time in a simulated three node distributed network. In their experiments, the network was able to arrive at the correct solution with as much as fifty percent of internode communication lost at random.

The quality of the knowledge used by each node to distinguish between consistent and inconsistent data plays a major role in the success of a functionally accurate, cooperative approach. A network using low quality knowledge is unable to detect subtle inconsistencies among tentative partial results and may be unable to arrive at an acceptable solution. As the quality of knowledge used in the network is improved, the network should generate an answer with greater accuracy in less time.

Unfortunately, high quality knowledge for complex problem solving situations can be difficult to obtain and encode. Many months can be spent developing the knowledge base for a single "expert system". Given the elusiveness of quality knowledge, what level of knowledge is sufficient for acceptable performance in a functionally accurate, cooperative distributed problem solving network? The next section describes the distributed vehicle monitoring testbed, a research tool developed, in part, to address this question.

2.3 The Distributed Vehicle Monitoring Testbed

This section introduces the distributed vehicle monitoring testbed, a flexible and fully-instrumented research environment constructed for the empirical evaluation of alternative designs for functionally accurate, cooperative distributed problem solving networks. Here, the motivation for the testbed, its basic structure, and its parameterization and measurement capabilities are described. The design and implementation of the coordination components of the testbed (which serve as the computational framework for organizational structuring) are presented in Chapter IV.

2.3.1 Motivation.

Distributed problem solving networks are highly complex beasts, difficult to analyze formally and expensive to construct and modify for empirical evaluation. To reduce these problems, an intermediate approach of simulating the network at a detailed level was taken. The result of this still substantial implementation effort is the distributed vehicle monitoring testbed.[1]

1. Construction of the testbed was far beyond the capabilities of a single researcher and became itself a distributed problem solving activity. The ongoing, cooperative efforts of these all-too-human "nodes" are summarized in the acknowledgments.

An important decision in the design of the testbed was the level at which the network would be simulated. An abstract modeling level, such as the one used by Fox [FOX79], that represents the activities of nodes as average or probabilistic values accumulated over time would not capture the changing intermediate processing states of the nodes. It is precisely those intermediate states that are so important in evaluating different network coordination strategies. Instead, the testbed duplicates (as closely as possible) the data that would be generated in an actual distributed vehicle monitoring network as well as the effect of knowledge and control strategies on that data.

A second design decision was to fully instrument the testbed. It is important to know how well a node is doing with respect to its data and organizational responsibilities as it develops its portion of the overall solution. As will be discussed at the end of this chapter, the testbed includes dynamic measures that indicate the quality of the developing solution at each node in the network, the quality of the developing solution in the network as a whole, and the potential effect of each transmitted message on the solution of the receiving node.

A third decision in the design of the testbed was to make it parameterized. Experience with complex artificial intelligence systems demonstrated the difficulty of experimenting with alternative knowledge and control strategies. As a result, potential experimentation with the system is often not performed. Incorporated into the testbed are capabilities for varying:

- o the knowledge sources available at each node, permitting the study of different problem solving decompositions;

- o the accuracy of individual knowledge sources, permitting the study of how different control and communication policies perform with different levels of system expertise;
- o vehicle and sensor characteristics, permitting control of the spatial distribution of ambiguity and error in the task input data;
- o node configurations and communication channel characteristics, permitting experimentation with different network architectures.

The result is a highly flexible research tool which can be used to empirically explore a large design space of possible network and environmental combinations.

One last consideration in the design of the testbed was a desire to avoid the substantial knowledge engineering effort characteristic of large knowledge-based artificial intelligence systems by keeping the basic problem solving task simple. We were eager to experiment with the problems of cooperative distributed problem solving -- not to develop a knowledge base for distributed vehicle monitoring nor to develop the definitive distributed vehicle monitoring algorithm. The goal of our simplifications was to reduce the processing complexity and knowledge engineering effort required in the testbed without significantly changing the basic network coordination characteristics of the distributed vehicle monitoring task.

The distributed vehicle monitoring testbed embodies these design criteria. In the next section, the simplified version of the distributed vehicle monitoring task used in the testbed is presented. In subsequent sections, the basic architecture of the testbed will be described.

2.3.2 The simplified vehicle monitoring task.

Since the purpose of the testbed is the evaluation of alternative distributed problem solving network designs rather than the construction of an actual distributed vehicle monitoring network, a number of simplifications of the vehicle monitoring task were made. The goal of these simplifications was to reduce the processing complexity and knowledge engineering effort required in the testbed without significantly changing the basic character of the distributed interpretation task.

The major simplifications include:

- The monitoring area is represented as a two-dimensional square grid, with a maximum spatial resolution of one unit square.
- The environment is not sensed continuously. Instead, it is sampled at discrete time intervals called time frames.
- Frequency is represented as a small number of frequency classes.
- Communication from sensor to node uses a different channel than internode communication.
- Internode communication is subject to random loss, but if a message is received by a node it is received without error.
- Sensor to node communication errors are treated as sensor errors.
- Signal propagation times from source to sensor are processed by the (simulated) low-level signal processing hardware of the sensor;
- Sensors can make three types of errors: failure to detect a signal; detection of a non-existent signal; and incorrect determination of the location or frequency of a signal.
- Sensors output signal events which include the location of the event (resolved to a unit square), time frame, frequency (resolved to a single frequency class), and belief (based on signal strength).

- o Incompletely resolved location or frequency of a signal is represented by the generation of multiple signal events rather than a single event with a range of values.
- o Nodes, sensors, and internode communication channels can temporarily or permanently fail without warning.

2.3.3 Basic node architecture.

The distributed vehicle monitoring testbed simulates a network of Hearsay-II nodes.[2] Each node is an architecturally complete Hearsay-II system, capable of functioning as a centralized vehicle monitoring system if it was given all of the sensory data and made use of all its knowledge. This flexibility permits any subset of the network's knowledge to be used at a node and allows the simulation of a single node (centralized) system to provide a benchmark for various distributed networks monitoring the same environment.

The selection of Hearsay-II as the basic architecture of a testbed node was based on several considerations. First, the multilevel, diverse knowledge source structure of Hearsay-II seemed appropriate for the distributed vehicle monitoring problem and had, in fact, been previously used in a centralized multisensor interpretation problem [NII78]. Second, experiments with the Hearsay-II speech understanding system had indicated its potential for functionally accurate,

2. "Hearsay-II" refers to the general problem solving architecture developed as part of the Speech Understanding Project at Carnegie-Mellon University [ERMA80]. In addition to speech understanding, the Hearsay-II architecture has been used in such diverse applications as protein-crystallographic analysis, image understanding, a cognitive model of planning, dialogue comprehension, multisensor interpretation of acoustic signals, a model of human reading, and a learning system [ENGE77, HANS78, HAYE79, MANN79, NII78, RUME76, SOLO77].

cooperative problem solving [FENN77, LESS80b]. Third, we had experience with the Hearsay-II architecture.

Problem solving in Hearsay-II.

The basic problem solving components of the Hearsay-II architecture at each testbed node are illustrated in Figure 6.[3] The major databases are: the blackboard, the event-to-knowledge-sources table, the focus-of-control database, and the scheduling queue. The major processing modules are: knowledge sources, the blackboard monitor, and the scheduler.

In a Hearsay-II system, domain knowledge is partitioned into a set of diverse and independent programs called knowledge sources. Each knowledge source contains knowledge about one aspect of the overall problem solving task. In the testbed, there are knowledge sources which know how to form signal groups from acoustic signals, vehicles from signal groups, and patterns from vehicles.

Knowledge sources do not call one another directly. Instead, they interact through a shared database called the blackboard. The blackboard is subdivided into a set of distinct information levels, each representing a different view of the overall problem. In the testbed, there are blackboard levels for signals, signal groups, vehicles, and patterns.

3. Additional coordination components of the testbed are presented in Chapter IV.

Figure 6: Testbed Node Architecture.

Each node in the distributed vehicle monitoring testbed is a structurally complete Hearsay-II system. The basic execution cycle of a Hearsay-II system begins with the execution of a knowledge source that makes changes to the blackboard. These changes are detected by the blackboard monitor which determines what additional knowledge sources should be executed in response to the changes. These knowledge sources are placed on the scheduling queue which is ordered by the scheduler based on the progress of problem solving in the system. When the currently executing knowledge source has completed, the highest rated knowledge source on the queue is executed, and the cycle repeats.

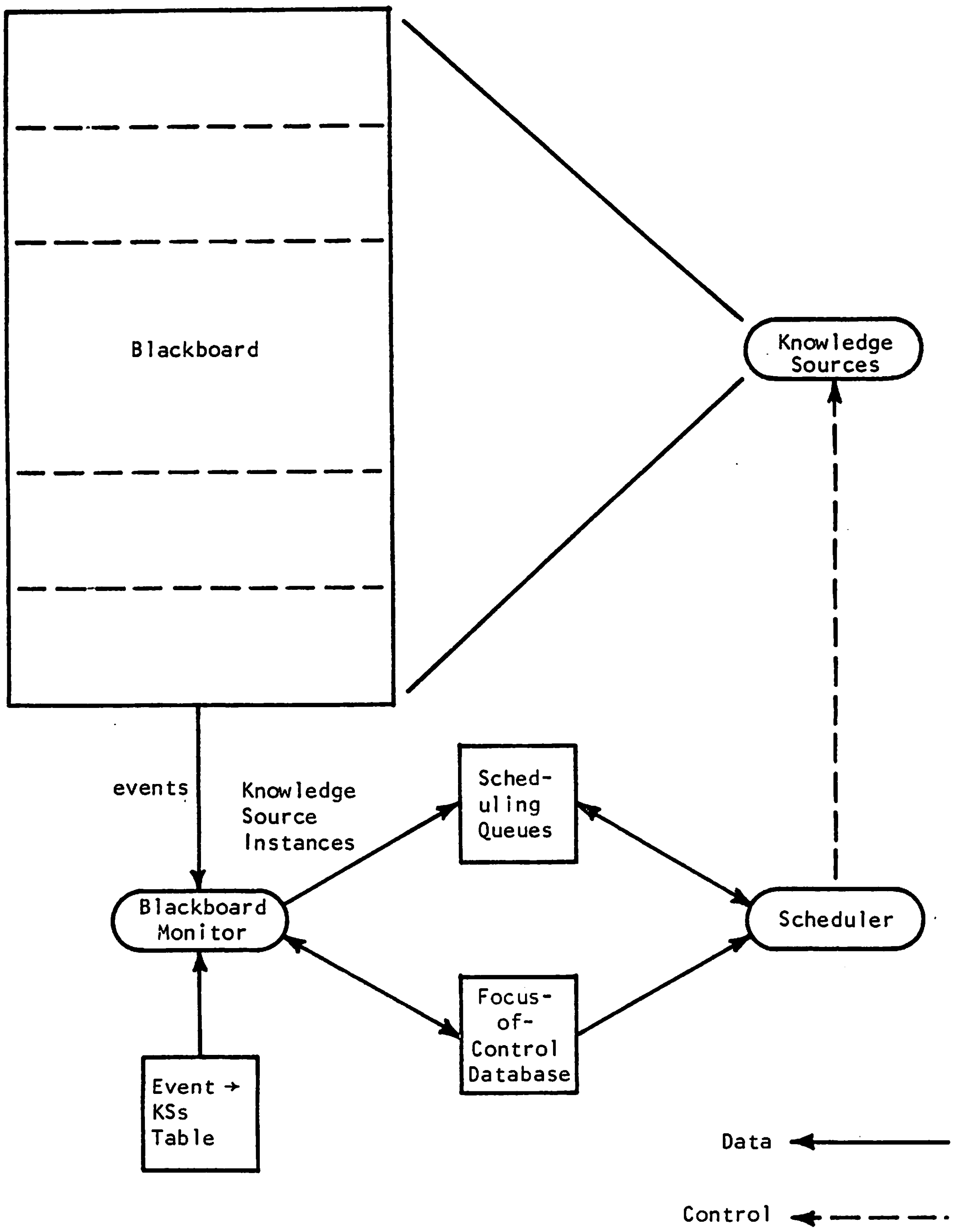


Figure 6: Testbed Node Architecture.

The basic data unit of the blackboard is the <u>hypothesis</u>. A hypothesis represents a partial solution to the overall problem expressed at one of the information levels of the blackboard. The possible hypotheses at a level represent the search space at that level. Each hypothesis includes a <u>belief</u> value indicating its consistency with supporting data and the likelihood that it is part of the overall solution. Relationships among hypotheses at different levels on the blackboard are represented by links, which allow a partial solution at one level to constrain the search at another level.

Knowledge sources are invoked in response to particular kinds of changes on the blackboard, called <u>blackboard events</u>. The <u>event-to-knowledge-source table</u> specifies which events are of interest to each knowledge source. This table is used by the <u>blackboard monitor</u> to create a <u>knowledge source instantiation</u> when a knowledge source's triggering events occur on the blackboard.

The newly-created knowledge source instantiation is added to the <u>scheduling queue</u> which is managed by the <u>scheduler</u> using the focus-of-control database. The <u>focus-of-control database</u> is updated by the blackboard monitor to reflect the global state of problem solving activity on the blackboard. The scheduler calculates a priority rating for each knowledge source instantiation on the scheduling queue, selecting for execution the one with the highest rating. Execution of the knowledge source instantiation causes changes to the blackboard which trigger additional blackboard events and the process continues.

Internode communication is added to the basic Hearsay-II architecture by the inclusion of communication knowledge sources. These knowledge sources allow the exchange of hypotheses among nodes in the same independent and asynchronous style used by the other knowledge sources.

The structure of the blackboard in a testbed node.

The blackboard at each node in the testbed is partitioned into four of the task processing levels described in Section 2.1. In order of increasing abstraction these are signal, group, vehicle, and pattern. Each of these levels is further divided into two levels, one containing location hypotheses and one containing track hypotheses. A location hypothesis represents a single event at a particular time frame. A track hypothesis represents a connected sequence of events over a number of contiguous time frames.

These orthogonal partitionings result in eight blackboard levels:

signal location (SL)
signal track (ST)
group location (GL)
group track (GT)
vehicle location (VL)
vehicle track (VT)
pattern location (PL)
pattern track (PT).

The relationships between these levels is shown in Figure 7. Location hypotheses are formed from location hypotheses at the next lower abstraction level. Track hypotheses can be formed from location hypotheses at the same abstraction level or from track hypotheses at the next lower level. This means that there are four possible blackboard paths for synthesizing pattern tracks from signal locations:

Figure 7: Blackboard Levels in the Testbed.

The eight blackboard levels in the testbed are:

signal location (SL)
signal track (ST)
group location (GL)
group track (GT)
vehicle location (VL)
vehicle track (VT)
pattern location (PL)
pattern track (PT).

The arrows indicate the four possible synthesis paths from sensory data to generation of the answer map. The task processing level most appropriate for shifting from location hypotheses to track hypotheses is dependent on the problem solving situation.

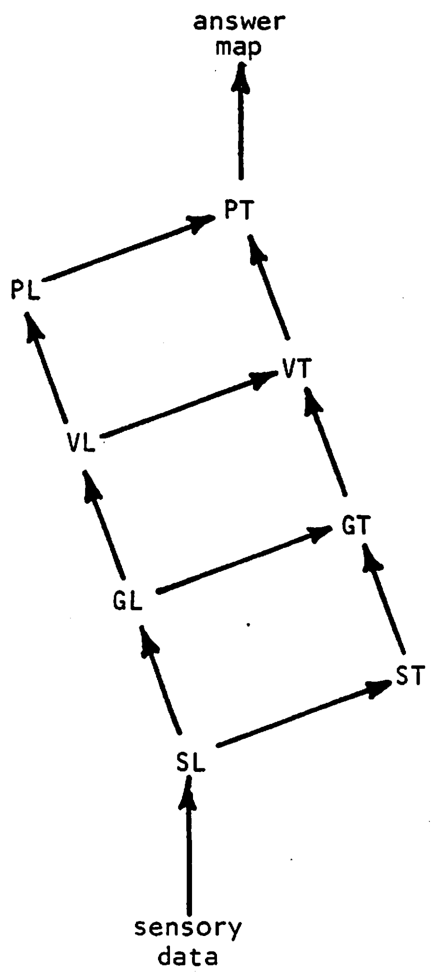


Figure 7: Blackboard Levels in the Testbed.

```
SL -> ST -> GT -> VT -> PT
SL -> GL -> GT -> VT -> PT
SL -> GL -> VL -> VT -> PT
SL -> GL -> VL -> PL -> PT.
```

The multiple synthesis paths allow track formation and extension at all data levels. This flexibility leads to a number of possible node activities which, as we will see in Chapter III, must be restricted and coordinated.

In addition to their level on the blackboard, hypotheses are indexed by the spatial (x,y) coordinates, time, and event class of the hypothesized event. The <u>time-location list</u> attribute[4] of a hypothesis is a list of time/location pairs indicating the hypothesized location at the indicated time frames. The time-location-list attribute of a location hypothesis contains only one such time/location pair; track hypotheses contain two or more time/location pairs. The <u>event class</u> attribute of a hypothesis indicates the kind of event the hypothesis represents, and its interpretation is level dependent. On signal levels an event class represents a particular signal frequency, resolved to a single frequency class. On group levels an event class represents the particular fundamental frequency class of the underlying harmonically related signal events. Above the group levels, the event class represents a classification. At the vehicle level an event class indicates a specific vehicle type, and at the pattern level a specific set of vehicle types with particular relative positions (a pattern).

4. A complete list of hypothesis attributes is given in Appendix B.

<u>The testbed grammar</u>.

The relationships among the event classes at each level is supplied to the testbed as part of a testbed <u>grammar</u>. The relationship among event classes in a simple grammar is illustrated in Figure 8. This grammar contains nine signal frequency classes which can be combined to form four signal group classes. The number of the group class corresponds to the fundamental frequency class of the group. For example, group 14 consists of fundamental frequency class 14 and harmonic classes 10 and 18. Notice that group classes 14 and 22 share a common harmonic signal class, 18. If only this harmonic is detected, the network cannot discriminate between these two groups (and without detection of other groups, between the two vehicles types at the next level in the grammar).

The grammar also specifies three possible pattern types which can be formed from the two vehicle types. Pattern type 1 is a single vehicle of type 1. Similarly, pattern type 2 is a single vehicle of type 2. Pattern type 3 requires a type 1 and a type 2 vehicle with a particular spatial relationship between them. (The spatial relationship is not illustrated in Figure 8.)

By increasing the connectivity of the grammar, the interpretation task can be made more difficult. For example, if vehicle class 1 included signal classes 2, 14, and 22 and vehicle class 2 included signal classes 14, 22, and 34, discriminating between vehicle class 1 and 2 would require the detection of either group class 2 or 34.

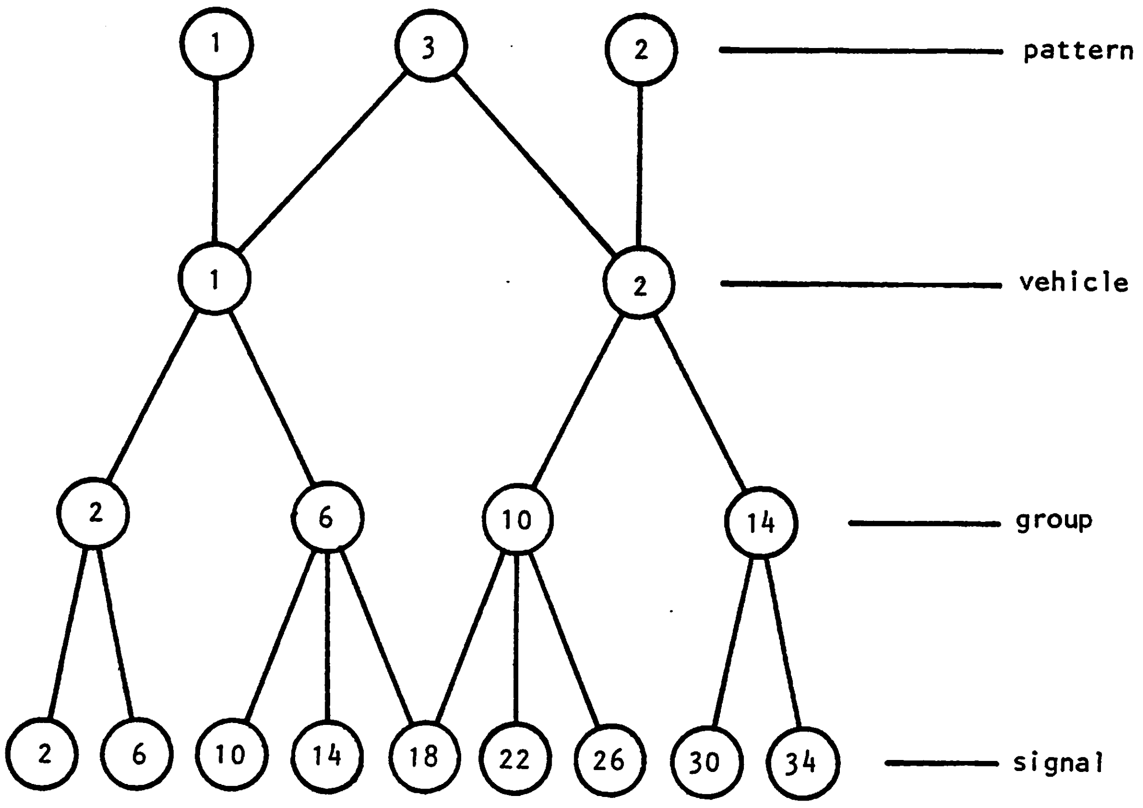


Figure 8: A Simple Testbed Grammar.

A simple, three pattern grammar specifying the relationship among the event classes at the various levels in the data hierarchy. There are two different vehicle types (1 and 2) each formed from two different signal groups. Each vehicle can form a single vehicle pattern (1 and 2, respectively) or both together (at a specific distance not illustrated here) can form a two-vehicle pattern (3). Signal class 18 is included among the supports of both vehicle types and adds minor confusion in this grammar.

There are several additional aspects to a testbed grammar which also increase the difficulty of the interpretation task. These aspects are best illustrated by overviewing the basic problem solving activities in the testbed. That is the subject of the next section.

2.3.4 Basic problem solving.

This section surveys the six basic problem solving activities in the testbed. They are: location synthesis, track synthesis, track formation, track extension, location-to-track joining, and track merging.

Location synthesis.

Location synthesis involves combining one or more location hypotheses at one level of the blackboard into a new location hypothesis at the next higher location level (Figure 9). The new location hypothesis is linked to its lower level <u>supporting hypotheses</u>. The possible supporting hypotheses are defined by the grammar. The belief of the created hypothesis is lowered if only a subset of the possible supports exist on the lower level.

For a group location hypothesis to be formed, the locations of the individual signal hypotheses should be the same. However, sensors do not always accurately locate the source of the signal. In the testbed, a signal event can be shifted by a sensor one location unit in either the x or y direction (Figure 10a). Similarly, sensors do not always accurately determine the frequency of the signal. In the testbed, the frequency class of a signal event can be shifted up or down by one class (Figure 10b). This means there are twenty-seven possible

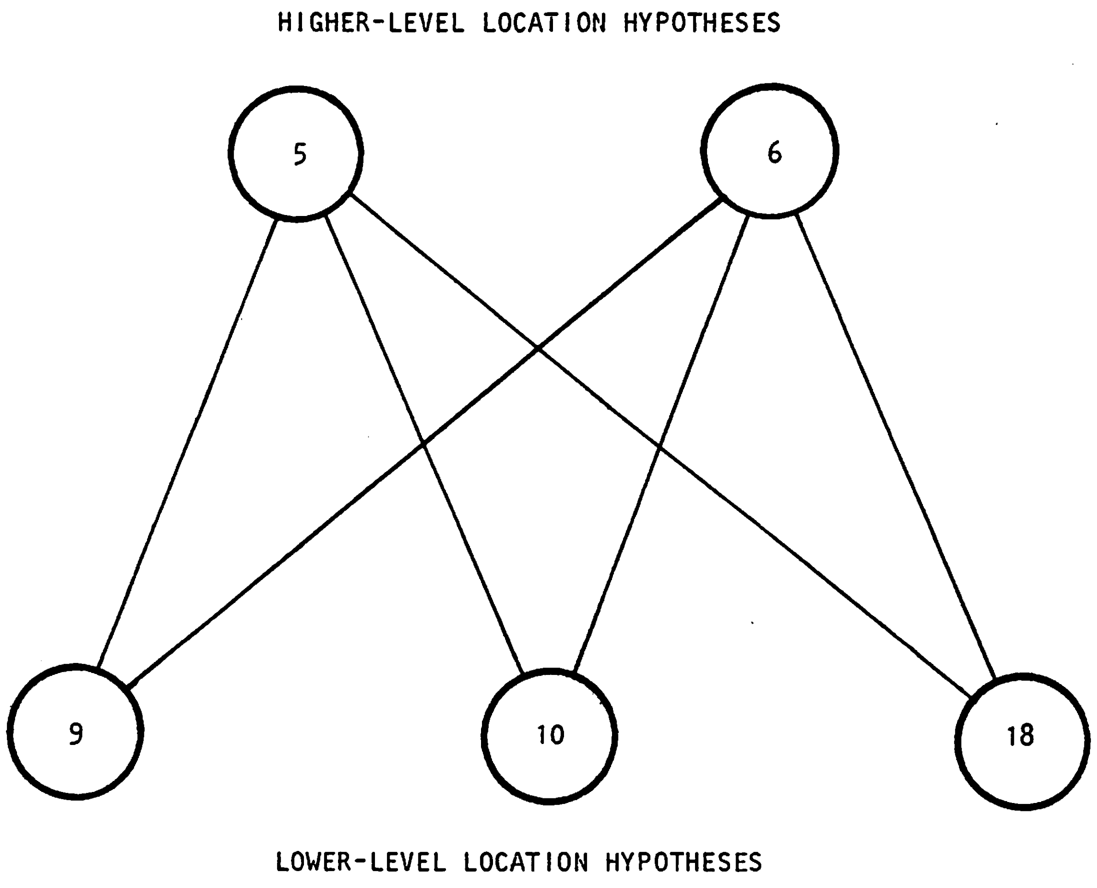

HIGHER-LEVEL LOCATION HYPOTHESES
5
6
9
10
18
LOWER-LEVEL LOCATION HYPOTHESES

Figure 9: Location Synthesis.

Combining location hypotheses at one level of the blackboard into new location hypotheses at the next higher location level.

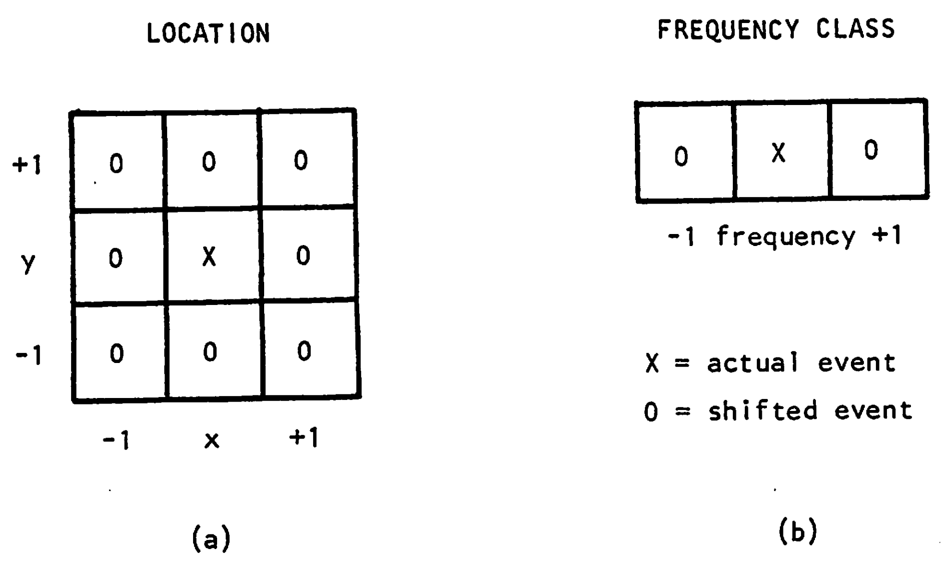


Figure 10: Inaccuracies in Signal Event Location and Frequency.

The sensed location of an actual event can be shifted in either the x or y direction (a) and the sensed frequency can be shifted up or down by one frequency class (b).

location/frequency class pairs for each signal event listed in the grammar. Depending on the sensor, up to sixteen of these possible pairs may be generated by a single sensor from a single "actual" signal source. The correct pair is not necessarily generated.

If the signal hypotheses are shifted in location or frequency class, so are the synthesized group hypotheses. For example, if signal location hypotheses with frequency classes 1 and 5 were processed using the grammar of Figure 8, a group location hypothesis of class 1 would be produced. This means that group hypotheses have the same twenty-seven possible location/frequency class pairs as signal location hypotheses. The knowledge sources in the testbed lower the belief of group location hypotheses synthesized from signal location hypotheses which are mismatched, but within the single unit tolerance, in location or frequency class.

Synthesis of a vehicle location hypothesis should ideally involve group location hypotheses with identical locations and with the frequencies listed in the grammar.[5] However, the potential for shifted group events requires the vehicle location synthesis knowledge source to consider group location hypotheses whose frequencies are shifted one unit to each side of the grammar frequency and which are within one unit location of one another. (If the locations are separated by two unit locations, vehicle location hypotheses with different locations are generated.) The belief of vehicle location hypotheses are lowered if formed from group location hypotheses with frequencies shifted from the grammar. The beliefs are further lowered if the group location hypotheses are mismatched in location or frequency with respect to one another.

<u>Track synthesis</u>.

Track synthesis is the combining of one or more track hypotheses at one level of the blackboard into a new track hypothesis at the next higher track level (Figure 11).

Track synthesis involves the same tolerance issues as location synthesis. Signal and group track hypotheses may be shifted in frequency class, and each location contained in any track hypothesis may

5. The possibility of gargantuan vehicles with widely separated acoustic sources is not considered.

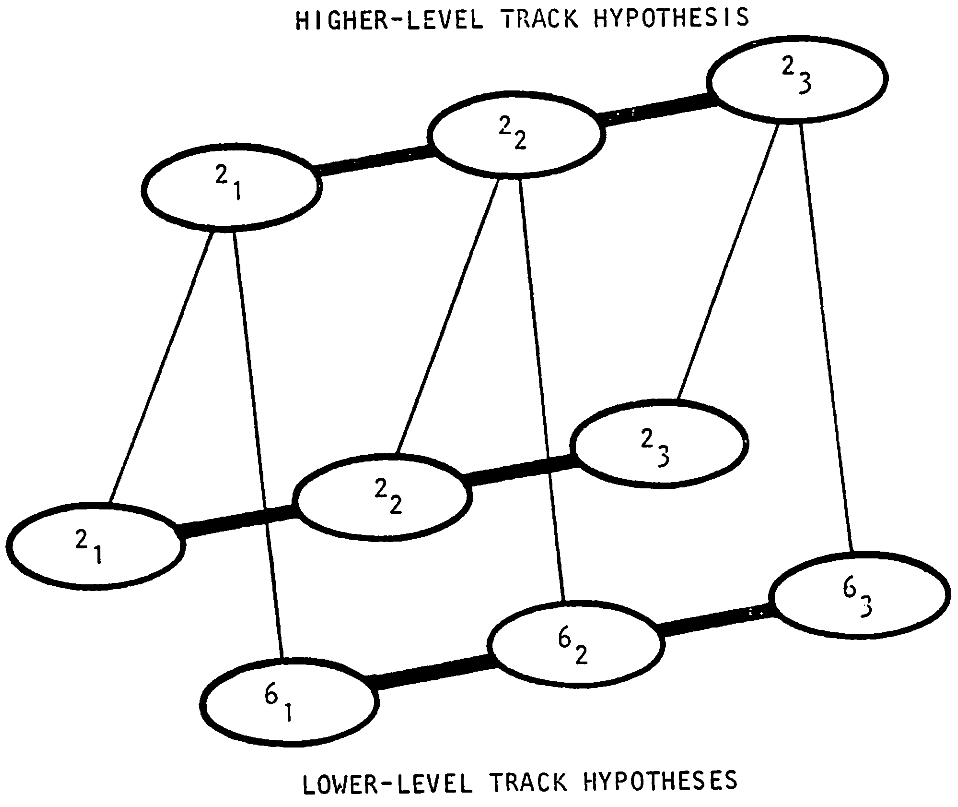


Figure 11: Track Synthesis.

Combining track hypotheses at one level of the blackboard into a new track hypothesis at the next higher track level.

be shifted. The belief of the new track hypothesis is lowered according to the number of missing support hypotheses, shifted frequency classes, and mismatched location hypotheses.

The tracking component of a grammar.

The tracking component of a a testbed grammar specifies the limitations on vehicle movement. It contains two values: the maximum velocity of a vehicle (and implicitly, events at all levels) and the maximum acceleration of a vehicle.[6] These values are used in three activities involving the creation of track hypotheses: forming a track hypothesis from individual location hypotheses, extending a track hypothesis with an additional location hypothesis, joining a location hypothesis to a track hypothesis (the dual to track extension), and merging two abutting or overlapping track hypotheses into a single track hypothesis. We first consider track formation.

Track formation.

To form a track from two location hypotheses, a hypothesis in one time frame is selected and combined with a matching hypothesis in the next time frame (or in the preceding time frame, if working backward in time). Here "matching" includes hypotheses within event class tolerance of hypotheses at the signal and group levels. These matching hypotheses must be within the distance permitted by the maximum velocity for a vehicle (Figure 12). Because there can be a number of matching hypotheses in successive time frames that are within this distance, a large number of incorrect, short track hypotheses can be formed. Significant additional processing may be required to determine which of these short tracks can be extended into additional time frames. For

6. Currently the maximum velocity and acceleration values apply to all vehicle types in the network.

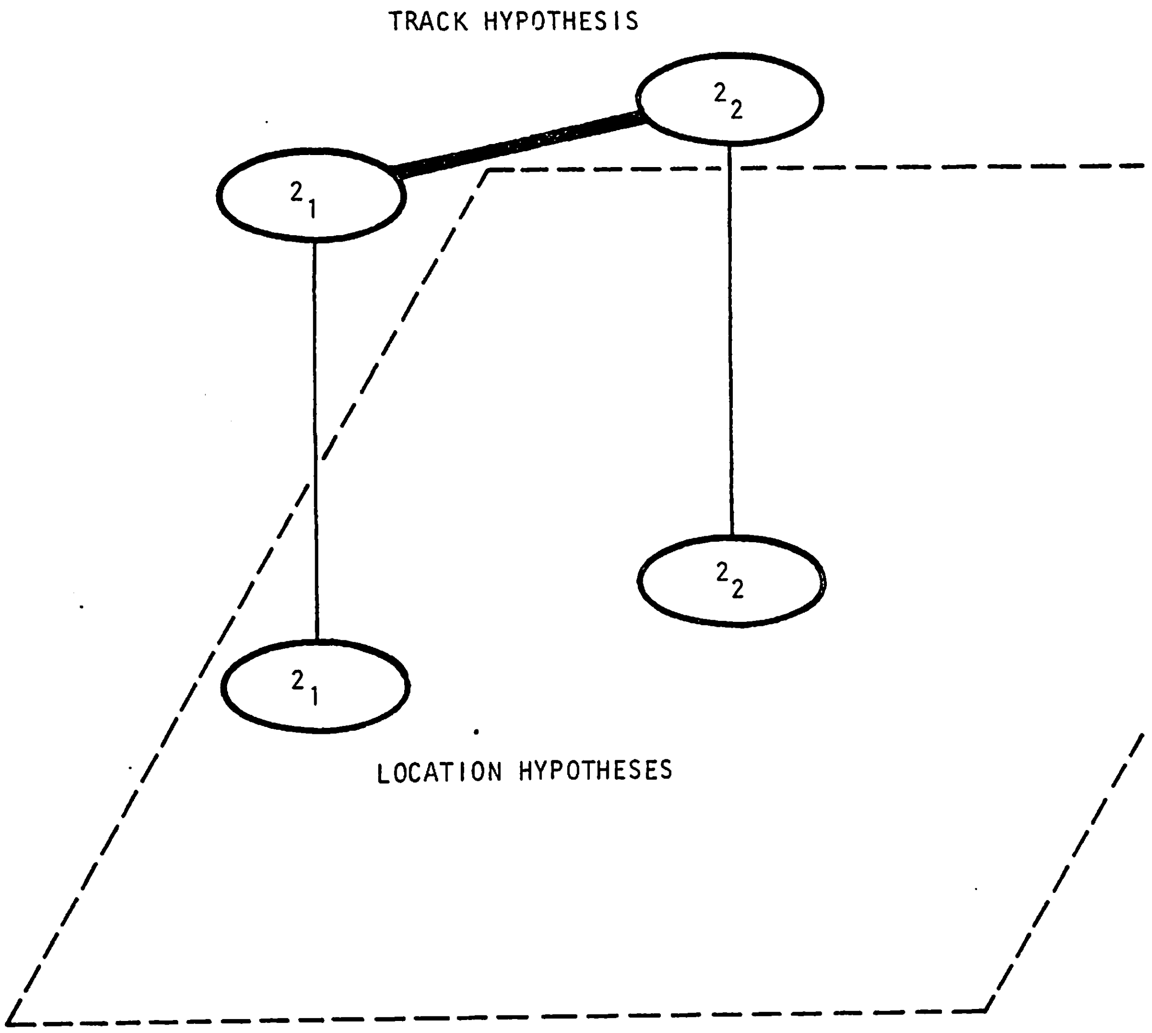


Figure 12: Track Formation.

Combining two location hypotheses in adjacent time frames into a track hypothesis. The location hypotheses must be within the distance permitted by the maximum velocity for a vehicle.

this reason, track formation is a relatively expensive operation.

Track extension.

Extending a track involves determining the spatial area in which the vehicle must be located in the next (or preceding) time frame, under the limitations imposed by the maximum velocity and acceleration of the vehicle (Figure 13). The distance covered during the last (or first) time frame of the track hypothesis determines the current velocity of the vehicle. The track hypothesis can be extended using each location hypothesis in this area which has an event class within tolerance of the event class of the track. Extensions requiring high accelerations are given lower beliefs than those requiring low accelerations.

Location-to-track joining.

Joining a location hypothesis to a track hypothesis is similar to track extension. Instead of a track hypothesis, however, we begin with a location hypothesis. All track hypotheses that are within the event class tolerance of the location hypothesis, that end (or begin) in the previous (or next) time frame, and that are within the maximum velocity are located (Figure 14). These candidate track hypotheses are then checked to ensure that the acceleration resulting from joining the location hypothesis to them does not exceed the maximum acceleration for a vehicle. As with track extension, joins requiring high accelerations are given lower beliefs than those requiring low accelerations. This knowledge source is more expensive than track extension because the acceleration of the vehicle cannot be used to filter the possible tracks until after the candidate track hypotheses have been found.

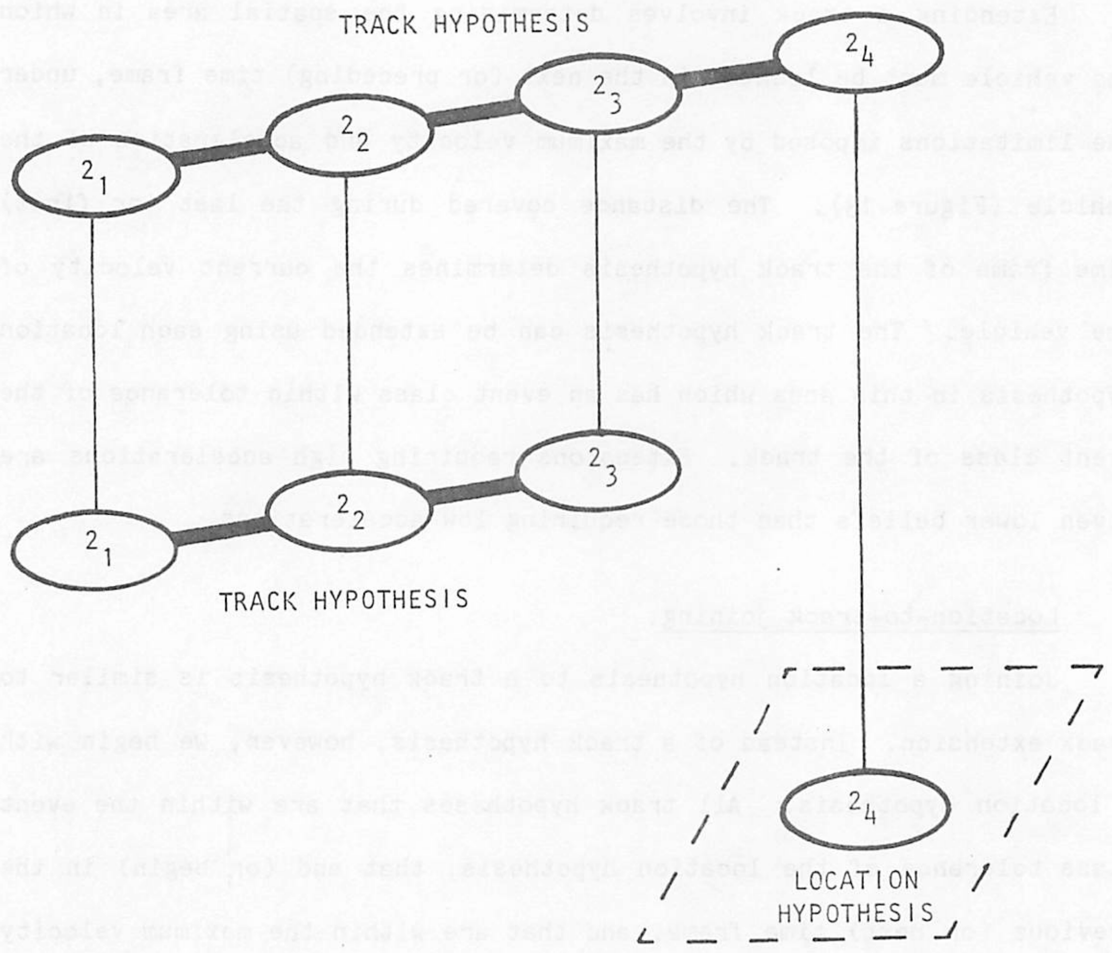


Figure 13: Track Extension.

Extending a track hypothesis with a location hypothesis in the next (or preceding, if extending backward in time) time frame.

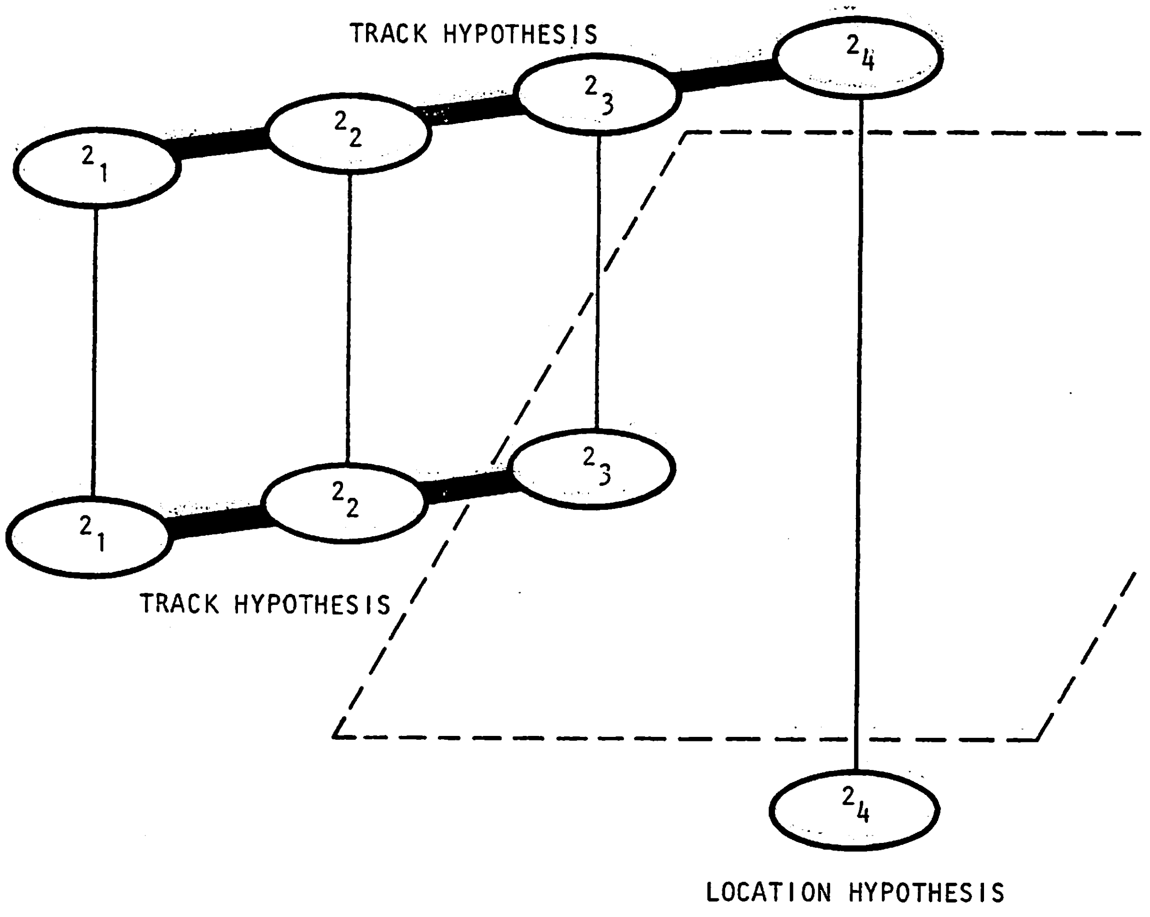


Figure 14: Location-to-Track Joining.

Joining a location hypothesis to a track hypothesis ending (or beginning) at an adjacent time frame.

Track merging.

Merging a track hypothesis with a second track hypothesis involves finding a track hypothesis matching the first track's event class which overlaps or abuts the last (or first) time frame (Figure 15). The overlapping or abutting locations must be within both event class and location tolerance for the merge to occur.

Extending a track hypothesis is the preferred way of monitoring an event. Although it is possible to track an event by repeated track formation followed by track merging, extension is much cheaper due to the additional context supplied by the track hypothesis. The intended application of each activity is:

track formation	-- initiation of new track hypotheses from existing location hypotheses;
location-to-track joining	-- joining an internally generated or externally communicated location hypothesis with existing track hypotheses (preferred over track formation);
track extension	-- extension of an internally generated or externally communicated track hypothesis using existing location hypotheses;
track merging	-- integration of existing track hypotheses with those received from other nodes (without using lower level hypotheses).

2.3.5 Knowledge sources.

The fifty-two testbed knowledge sources are listed in Table 1.[7] In addition to knowledge sources which perform the five basic problem solving activities discussed above, there are communication knowledge

7. Additional communication knowledge sources used in the goal-directed implementation of the testbed are discussed in Chapter IV.

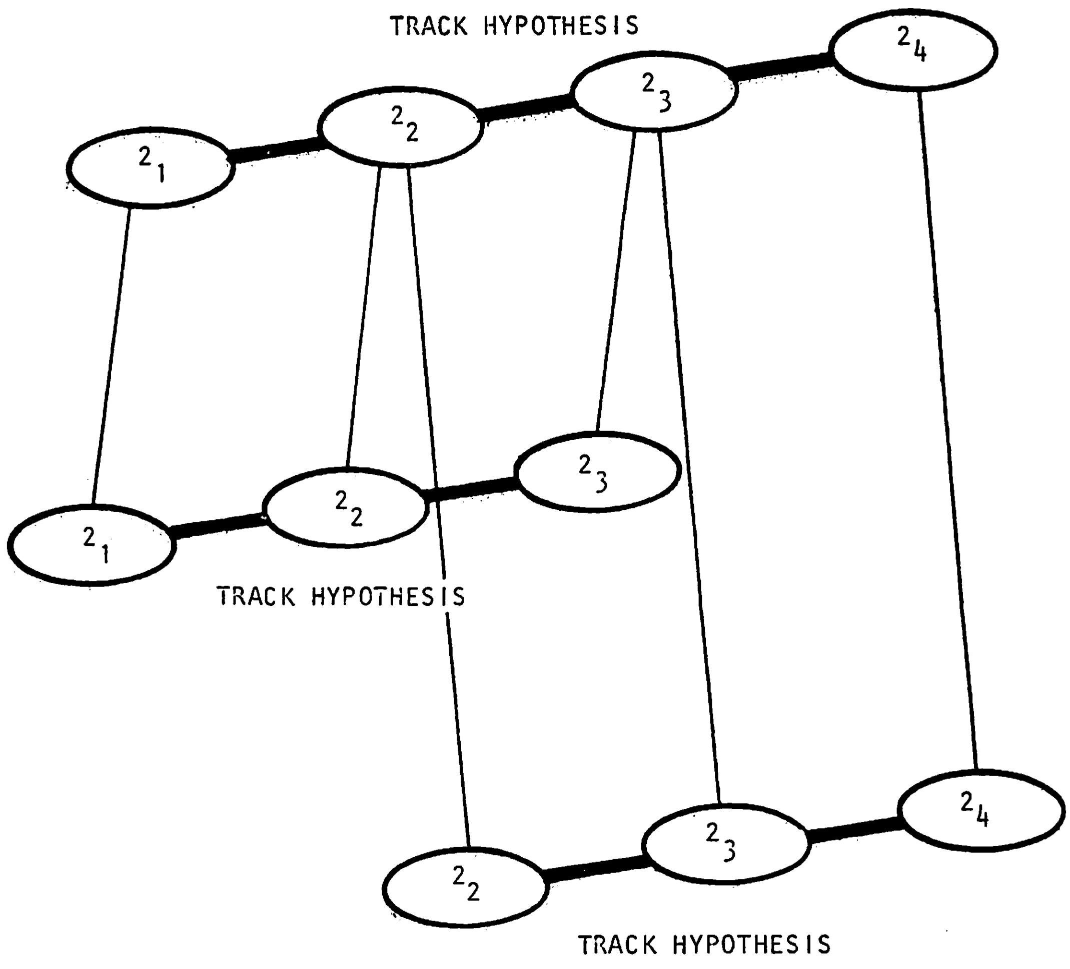


Figure 15: Track Merging.

Merging two overlapping track hypotheses into a single track hypothesis.

KNOWLEDGE SOURCES

Location Synthesis:

S:SL:GL
S:GL:VL
S:VL:PL

Forward Extension:

EF:ST/SL:ST
EF:GT/GL:GT
EF:VT/VL:VT
EF:PT/PL:PT

Forward Merging:

MF:ST:ST
MF:GT:GT
MF:VT:VT
MF:PT:PT

Forward Location-to-Track Joining:

JF:SL/ST:ST
JF:GL/GT:GT
JF:VL/VT:VT
JF:PL/PT:PT

Track Formation:

FT:SL:ST
FT:GL:GT
FT:VL:VT
FT:PL:PT

Backward Extension:

EB:ST/SL:ST
EB:GT/GL:GT
EB:VT/VL:VT
EB:PT/PL:PT

Backward Merging:

MB:ST:ST
MB:GT:GT
MB:VT:VT
MB:PT:PT

Backward Location-to-Track Joining:

JB:SL/ST:ST
JB:GL/GT:GT
JB:VL/VT:VT
JB:PL/PT:PT

Track Synthesis:

S:ST:GT
S:GT:VT
S:VT:PT

Miscellaneous:

FRONTEND
SENSORS

Hypothesis Transmission:

HYP-SEND:SL:SL
HYP-SEND:ST:ST
HYP-SEND:GL:GL
HYP-SEND:GT:GT
HYP-SEND:VL:VL
HYP-SEND:VT:VT
HYP-SEND:PL:PL
HYP-SEND:PT:PT

Hypothesis Reception:

HYP-RECEIVE:SL:SL
HYP-RECEIVE:ST:ST
HYP-RECEIVE:GL:GL
HYP-RECEIVE:GT:GT
HYP-RECEIVE:VL:VL
HYP-RECEIVE:VT:VT
HYP-RECEIVE:PL:PL
HYP-RECEIVE:PT:PT

Table 1: Distributed Vehicle Monitoring Testbed Knowledge Sources.

The fifty-two testbed knowledge sources. Each knowledge source can be individually selected and weighted at each testbed node (except for the FRONTEND, which runs at a special "simulation node" zero).

Except for the FRONTEND and SENSORS knowledge sources, the name of each knowledge source has the form:

type : input-level(s) : output-level.

sources and the FRONTEND and SENSORS knowledge sources.

Communication.

There are two types of communication knowledge sources at each node: hypothesis send and hypothesis receive.

A hypothesis send knowledge source transmits hypotheses created on the blackboard to other nodes based on the level, time frame, event class, location, and belief of the hypothesis. The send knowledge sources use a simple model of the hypotheses that have been seen by each node and of the availability of the communication channel to decide whether or not to send a particular hypothesis.

A hypothesis receive knowledge source places hypotheses received from other nodes onto the node's blackboard. Incoming hypotheses are filtered according to the level, time frame, event class, location, and belief of the received hypothesis to ensure that the node is truly interested in the information. (Specification of the interest areas of a node will be discussed in Chapter IV.) Hypothesis receive knowledge sources also use a simple model of the credibility of the sending node to possibly lower the belief of the received hypothesis before it is placed on the blackboard.

The FRONTEND and SENSORS knowledge sources.

The FRONTEND knowledge source is a special, simulation-level knowledge source used to initialize the testbed network. It is always the first knowledge source executed in a run. The FRONTEND reads a complete specification of the run from an input file called the environment file. An environment file includes: the configuration of the nodes and sensors, the grammar, and the movement of vehicle patterns through the monitoring area. (A description of an environment file is contained in Appendix A.) The FRONTEND creates all signal location hypotheses for the run, but does not insert them onto the blackboard. The FRONTEND also generates parameterization and measurement information (described in the next section).

The SENSORS knowledge source is executed once each time frame at each node. SENSORS inserts onto the node's blackboard the signal location hypotheses for the time frame which were previously created by the FRONTEND. A node can delay the receipt of sensory information by postponing the execution of the SENSORS knowledge source.

A note regarding prediction.

The thoughtful reader may wonder why top-down, prediction knowledge sources are not included in the testbed. As will be discussed in Section 4.1, prediction of important activities need not involve the top-down elaboration of hypotheses, and there are no prediction knowledge sources in the testbed.

2.3.6 Measurement capabilities.

A testbed experiment typically involves a series of runs in which a limited number of characteristics are independently varied. Often such gross measures of the testbed's performance as the time to generate a solution, the number of created hypotheses, the number of transmitted and received hypotheses, and the belief and accuracy of the solution can be used to compare runs. (This is the case with the experiments described in Chapter V.) However, understanding exactly why one run is significantly better or worse than another can require detailed measurement of the changing intermediate state of processing at each node in the network. Measures of the immediate and long term effects of executing a knowledge source and sending or receiving a message as well as the context in which these activities were performed are needed to truly understand what has occurred during a run. Even a slight delay in performing a critical activity can have a significant impact on the gross behavior of the network.

The need for such detailed measures led to the development of a model for analyzing how a Hearsay-II system constructs a solution and resolves the uncertainty and error in its input data. The basic approach was developed by Lesser, Pavlin, and Reed [LESS80a] and involves a measure for system performance which increases as the belief of "correct" hypotheses increases and as the belief of "incorrect" hypotheses decreases. The "correctness" of hypotheses is obtained from a hidden data structure called the consistency blackboard, which is precomputed by the FRONTEND from the simulation input data. This blackboard holds what the interpretation would be at each information level if the system worked with perfect knowledge. This blackboard is not part of the basic problem solving architecture of a node but rather is used to measure problem solving performance from the perspective of the simulation input data. The consistency blackboard is also used to mark consistent and false hypotheses (and the activities associated with them) in system output.

The present form of these measures allows for sophisticated analysis of locally consistent hypotheses which can only be viewed as incorrect by knowledge sources working above a particular blackboard abstraction level, of the instantaneous and potential effects of hypothesis communication, and of the distribution of uncertainty within the blackboard of each node as well as throughout the network.

2.3.7 Modifying knowledge source power.

One parameter that can have a significant effect on the performance of the network is the problem solving expertise of the nodes. The ability of each knowledge source to detect local consistencies and inconsistencies among its input hypotheses and to generate appropriate output hypotheses is called the power of the knowledge source. Knowledge source power ranges from a perfect knowledge source able to create output hypotheses with beliefs that reflect even the most subtle consistencies among its input hypotheses down to a knowledge source which creates syntactically legitimate output hypotheses without regard to local consistency and with beliefs generated at random. Note that a perfect knowledge source is not the same as an omniscient one. A perfect knowledge source can still generate an incorrect output hypothesis if supplied with incorrect, but completely consistent, input hypotheses.

The testbed can modify the power of a knowledge source to be anywhere along this range. This is achieved by separating each knowledge source into two stages: a candidate generator and a resolver. The candidate generator stage produces plausible hypotheses for the output of the knowledge source and assigns each hypothesis a tentative belief value. The candidate generator stage for each knowledge source in the testbed incorporates relatively simple domain knowledge. The next stage, the resolver, uses information provided by the consistency blackboard to minimally alter the initial belief values of these plausible hypotheses to achieve, on the average, a knowledge source of the desired power. The hypotheses with the highest altered beliefs are then used by the resolver stage as the actual output hypotheses of the knowledge source.

The alteration of hypothesis belief values by the resolver stage can be used to simulate the detection of more subtle forms of local consistency than is provided by the candidate generator's knowledge (and thereby increase the apparent power of the knowledge source). Hypothesis belief alteration can also be used to degrade the performance of the candidate generator (and thereby reduce the apparent power of the knowledge source).

2.3.8 Still to come.

The preceding sections discussed the basic structure, processing, and measurement capabilities of the distributed vehicle monitoring testbed. One major component of the testbed remains to be described. This component is the local planner at each node and is the subject of Chapter IV. First, however, the next chapter develops the problem of coordinating a functionally accurate, cooperative distributed problem solving network and discusses the need for sophisticated local planning by each node to implement network coordination.

Let not thy left hand know what thy right hand doeth.

-- St. Matthew

CHAPTER III
COORDINATING NODE ACTIVITY

This chapter looks at the problem of coordinating node activity in the distributed vehicle monitoring testbed. The basic issue is having each node make reasonable local activity decisions with incomplete, inaccurate, and inconsistent local information regarding both the developing answer map and the current and planned activities of other nodes in the network. The first section considers the problem of identifying and ordering the activities within a single network node. The second section presents the network coordination problem through a highly simplified example. Subsequent sections describe the contract network and self-directed approaches to the network coordination problem, develop a framework which integrates them both, and explain why even this integrated framework is insufficient to effectively deal with the network coordination problem. The last sections introduce the application of the functionally accurate, cooperative approach to the network coordination problem and the use of meta-level coordination through organizational design as a means of augmenting the integrated coordination framework.

3.1 The Internal Coordination Problem

It is unreasonable to expect coordinated network activity from nodes that cannot effectively control their own behavior. Assuming the general activities for each node in the network have been decided, we would at least like the nodes to carry out those activities in an effective manner. In this section we look at the internal coordination involved in generating an answer map within a single node of the vehicle monitoring testbed.

In a Hearsay-II system (used as our testbed node), internal coordination involves deciding which knowledge source instantiation should be executed next, taking into account the state of the developing solution. Because knowledge sources are non-interruptible tasks in a Hearsay-II system, this decision is made only at the completion of each executing knowledge source instantiation. The difficulty in deciding which knowledge source instantiation should be executed stems from four sources:

1. The space of possible interpretations (answer maps) is very large due to the amount of error in the input sensory data. It is therefore important to prune the search by executing only those knowledge source instantiations which develop interpretations that are best supported by the data.

2. The interpretation has to be driven upward through a number of blackboard abstraction levels. A highly-believed partial solution at a high blackboard level should focus knowledge source activity at lower levels on data with the potential for adding to that solution.

3. There are multiple synthesis paths through different blackboard levels involving different knowledge sources with different expected costs. It is important to evaluate these alternative paths and, once a decision to attempt a particular path is made, inhibit work on the other paths. The node should not waste its processing resources redundantly deriving the same

partial solutions in multiple ways.

4. There are communication knowledge sources (in multinode networks) which can be executed in place of a sequence of other knowledge sources (such as asking a neighboring node for a hypothesis at a particular level rather than deriving it locally). Of course, the neighboring node needs to have derived the requested hypothesis and to have sufficient communication capability to receive the request and transmit the hypothesis.

These considerations make it important that the node be aware of what it is doing and why it is doing it. The node must be able to determine what it needs to be doing and then to develop and implement a _plan_ for doing it. As will be shown in Chapter IV, this requires more sophisticated control capabilities than exist in the basic Hearsay-II architecture.

An important remaining consideration is when to stop executing knowledge sources that are working on "old" data. Stated differently, when has the space of possible interpretations been searched sufficiently to be confident of the accuracy of the answer? In the testbed, the first answer map that is generated is not necessarily the highest answer map that would be found if processing were allowed to continue. The distributed vehicle monitoring application is further complicated because vehicles do not necessarily cross the entire monitoring area (making it difficult to determine if a particular track died out naturally or was the result of correlating sensory noise), because there are an unknown number of vehicles present in the environment (making it difficult to decide if the answer map is complete), and because the network may be operating in a continuous, real-time manner (resulting in a conflict between increasing the accuracy of the answer map versus extending it forward in time). The knowledge source stopping question is even more difficult in a multinode network due to the lack of a complete and accurate view of the developing solution in the network. For the present, we will ignore the problem of deciding when a solution has been found, but will return to these issues in Chapter IV.

It is interesting to note that if each node contained but one knowledge source (in the distributed experts tradition) the internal coordination problem becomes trivial. If there is something for a node's knowledge source to do, the node should do it. While this may initially appear to be an advantage of the distributed experts approach, the coordination problem has merely been shifted from a local issue to the more difficult one of coordinating all activities at the network level. To what node(s) should the results of the node's knowledge source execution be sent? From what node(s) should a node accept hypotheses? What should a node do if it finds its knowledge source in great demand? What should the node do if it finds a node or communication link has failed? These questions must be addressed at the network coordination level.

In the next section, we look at the problem of coordinating the local node activities throughout the network. We will address the above questions, not from the one-node/one-knowledge-source structure of a distributed experts network, but from the more general perspective of full-functioned nodes.

3.2 The Network Coordination Problem

Suppose every node in the distributed vehicle monitoring testbed is supplied with the entire complement of knowledge sources, giving each node the potential for performing any portion of the problem solving activity needed to generate an answer map. What portions should each node perform to most effectively generate the map?

One way of approaching this question is to consider the distributed vehicle monitoring task as the growing of a hypothesis structure between the signal location hypotheses distributed among the nodes and the pattern track hypotheses which are to be generated at the appropriate answer site nodes. Note that only the initial node locations of the signal location hypotheses generated by the sensors and the eventual node location of the pattern track hypotheses used in the answer map have predefined node locations. The remainder of the hypothesis structure can be distributed (and duplicated) among any of the nodes in the network, subject only to communication and computation restrictions.

This flexibility in hypothesis node location makes determination of node activity a difficult problem, even when the required problem solving activities are completely known.

3.2.1 A simple (but extended) example.

Consider a simple two node distributed vehicle monitoring network in which the vehicle is located in an area covered only by sensors reporting to Node:1 and in which Node:2 is to produce the answer map (Figure 16). To further simplify the example, only accurate signal location hypotheses are reported to Node:1, the grammar used contains

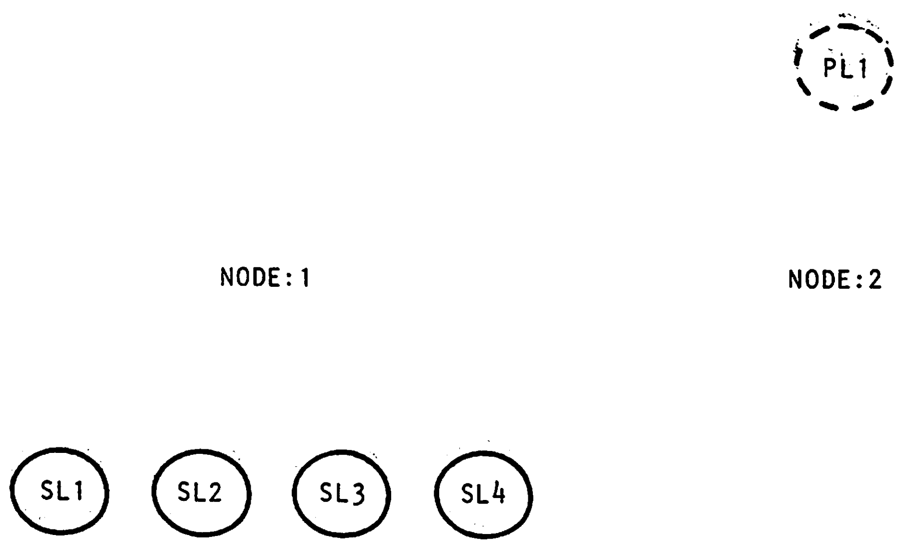


Figure 16: The Simple Two Node Problem.

All sensory signals are reported to Node:1 and Node:2 is to produce the answer map (indicated by the dashed pattern location hypothesis).

only one pattern, and only one time frame is considered: the answer map to consist of the correct pattern location hypothesis. One decomposition of the problem is to have Node:1 perform the entire problem solving task and transmit the answer pattern location hypothesis to Node:2 which supplies it as the answer map (Figure 17). Given that pattern location hypotheses are the most abstract of all location level hypotheses, this approach requires the minimal amount of internode communication. However, it does not exploit Node:2's processing capacity, since Node:2 is idle while Node:1 solves the problem.

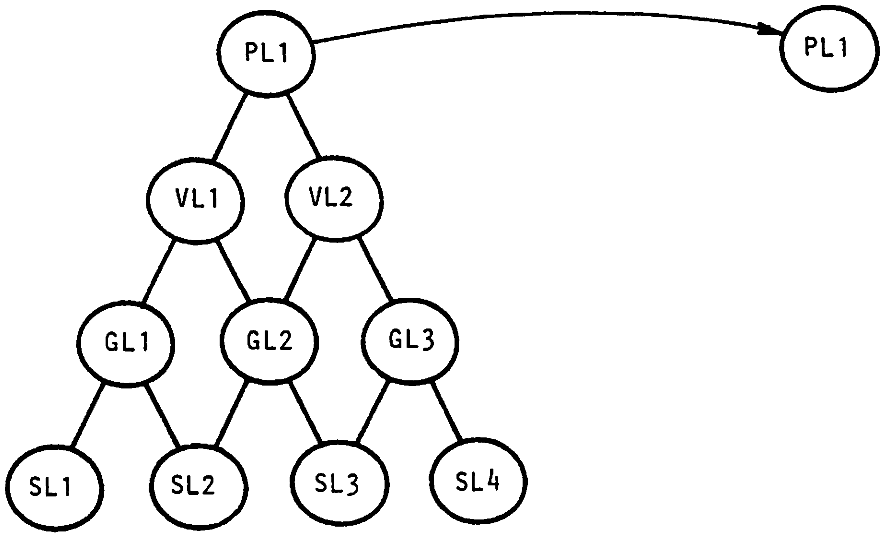


Figure 17: Node:1 Solves the Entire Problem.

In this decomposition, Node:1 performs the entire problem solving task and transmits the answer pattern location hypothesis to Node:2 which supplies it as the answer map.

To maximize the amount of parallelism in our two node network, Node:1 must immediately send Node:2 some of its sensory hypotheses, leaving Node:2 idle only during the time required to communicate enough signal location hypotheses to allow it to begin processing. But how many signal location hypotheses should Node:1 send to Node:2 and which hypotheses should they be? The decision can have significant impact on the effectiveness of problem solving in the network.

A very simple metric can be placed on our example by counting the number of support links created at each node (a measure of processing) and the number of hypotheses transmitted between nodes (a measure of communication). If Node:1 performs all the processing and sends the answer map to Node:2, then:

```
Node:1 Supports = 12 links
Node:2 Supports =  0 links
Communication   =  1 hypothesis.
```

Now suppose Node:1 sends hypothesis SL4 to Node:2 and processes only hypotheses SL1 through SL3 (Figure 18). In this case:

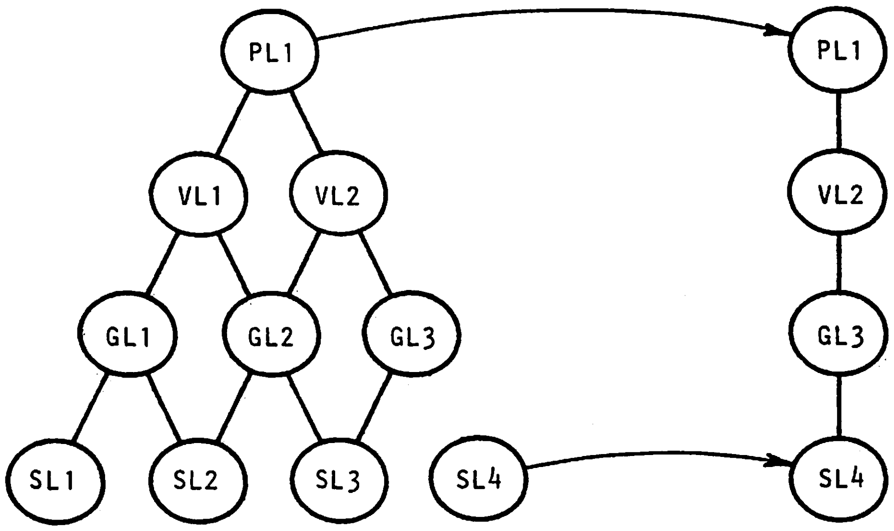


Figure 18: Node:1 Transmits One Signal Location Hypothesis to Node:2.

In this decomposition, Node:1 sends hypothesis SL4 to Node:2 and processes only hypotheses SL1 through SL3.

Node:1 Supports = 11 links
Node:2 Supports = 3 links
Communication = 2 hypotheses.

Obviously this is not much of an improvement.

Suppose Node:1 sends hypotheses SL3 and SL4 to Node:2 and processes only SL1 and SL2 (Figure 19):

Node:1 Supports = 8 links
Node:2 Supports = 8 links
Communication = 3 hypotheses.

This balances processing activity in the network at the cost of two additional communicated hypotheses (and processing accuracy, see below).

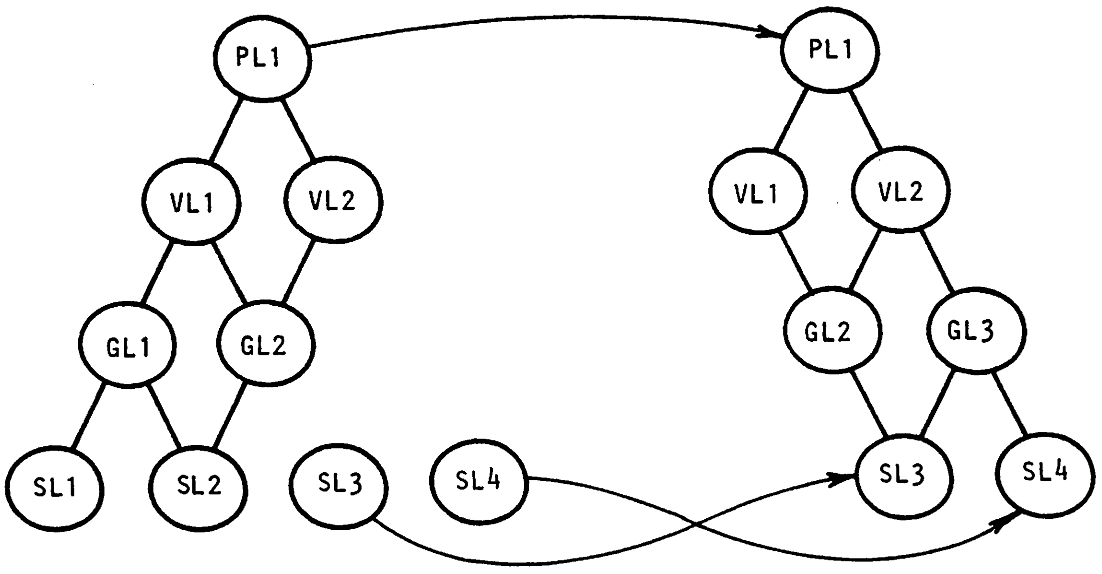


Figure 19: Node:1 Transmits Two Signal Location Hypotheses to Node:2.

In this decomposition, Node:1 sends hypotheses SL3 and SL4 to Node:2 and processes only hypotheses SL1 and SL2.

Would a different combination of transferred signal location hypotheses work equally well? Suppose SL2 and SL3 are sent to Node:2 (Figure 20):

```
Node:1 Supports =  6 links
Node:2 Supports = 10 links
Communication   =  3 hypotheses.
```

In this case the processing balance is not so uniform. The reason is that this decomposition does not match the connectivity of the grammar as well as the decomposition of Figure 19.

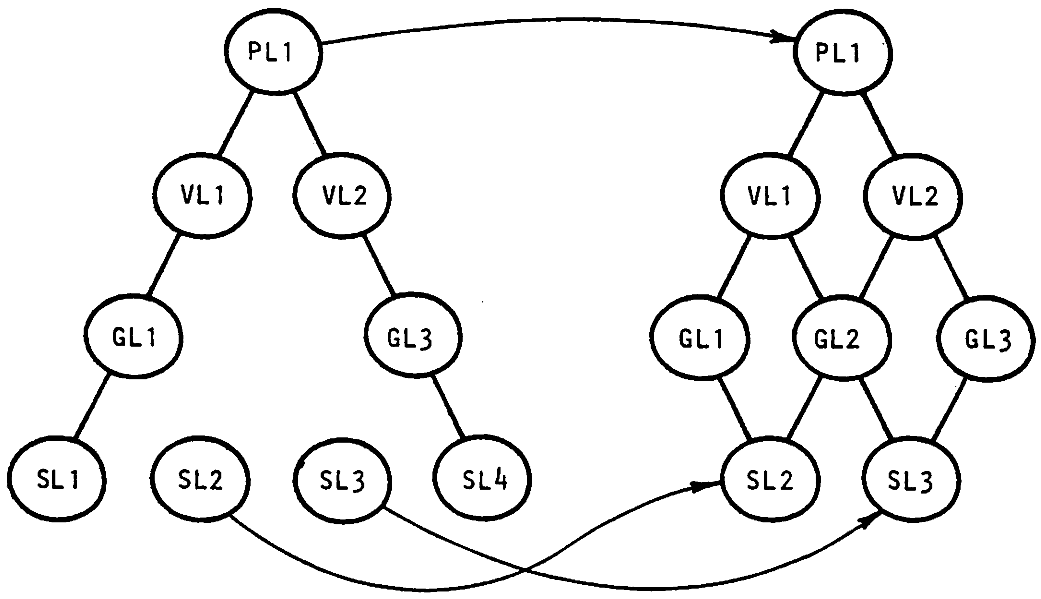


Figure 20: Node:1 Transmits a Different Two Signal Location Hypothesis to Node:2.

In this decomposition, Node:1 sends hypothesis SL2 and SL3 to Node:2 and processes only hypotheses SL1 and SL4.

In our simple measures, the effect that decomposing the problem had on the belief of the eventual answer was ignored. Recall from Section 2.3.4 that the belief of a created hypothesis increases with the number of hypotheses which support it. If we count the effect of each support link equally, the PL1 hypothesis created by each node in the Figure 19 decomposition is generated using four fewer supports than the PL1 hypothesis generated when Node:1 performed all the processing. This means that the load-balanced decomposition generates an answer with lower belief due to the use of partial sensory information. Although Node:2 eventually receives the PL1 hypothesis generated by Node:1, it does not know how the hypothesis was formed. Without the underlying support structure of the received PL1 hypothesis, Node:2 does not know if that hypothesis was generated using the same sensory data it used locally to created its PL1 hypothesis or was generated by Node:1 using additional or independent sensory data.[1] This is one example of the tradeoff between processing speed and the accuracy of the answer.

A similar tradeoff occurs between the amount of communication and the accuracy of the answer. (In fact, accuracy, speed, and communication are all interrelated.) Suppose we allow group location hypothesis communication in the Figure 19 decomposition. If Node:1 sends GL1 to Node:2 and Node:2 sends GL3 to Node:1 (Figure 21):

1. Determining independent versus redundant hypotheses generated at different nodes is a very difficult problem, requiring sophisticated models of the processing occurring at other nodes if nodes are to avoid communicating the underlying support structures. In the current version of the testbed, the maximum belief of the received hypothesis (which includes the credibility of the sending node) and the existing local hypothesis is used for the resulting hypothesis.

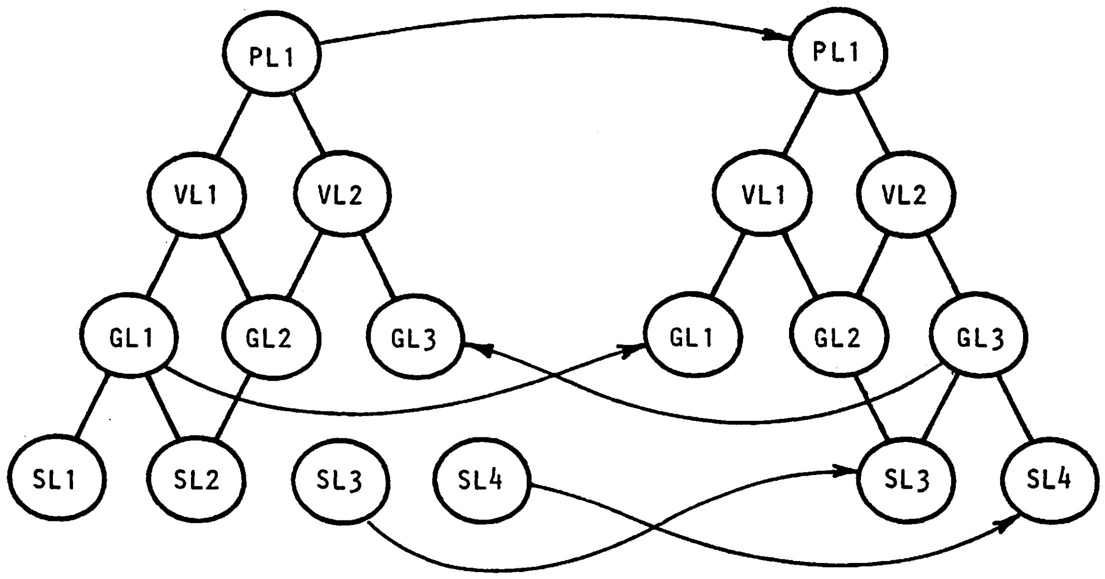


Figure 21: Adding Group Location Communication to the Figure 19 Decomposition.

In this decomposition, Node:1 sends hypotheses SL3 and SL4 to Node:2 and processes only hypotheses SL1 and SL2. Node:1 also sends hypothesis GL1 to Node:2 and Node:2 sends hypothesis GL3 to Node:1.

```
Node:1 Supports =  9 links
Node:2 Supports =  9 links
Communication   =  5 hypotheses
```

and the number of supports used to generate the pattern location hypotheses is only one fewer than the single node processing decomposition of Figure 17. Again the choice of appropriate hypotheses to exchange is important.

A closer look at Figure 21 shows some redundant activities in the network. Node:1 sends Node:2 the same PL1 hypothesis (with the same belief) as Node:2 generates locally. This processing can be eliminated if Node:1 sends Node:2 vehicle location hypothesis VL1 instead. Similarly, there is no need for Node:2 to send Node:1 hypothesis GL3. By carefully tailoring the problem solving activities, a decomposition with a good balance between speed and communication can be obtained which has no loss in accuracy with respect to the single node processing case (Figure 22):

Node:1 Supports	=	6 links
Node:2 Supports	=	6 links
Communication	=	4 hypotheses.

Although removing the redundant activity does improve the speed and communication requirements of the network, it also leaves the solution more vulnerable if Node:1 or the communication link should fail during the course of problem solving. In a distributed problem solving network, redundant activity is not necessarily bad. Part of the coordination problem is being intelligent about what redundant activities should be performed in the network.

The coordination problem does not end with problem decomposition. In the discussion thus far, we have ignored the timing of the particular activities at each node. Figures 17 through 22 only depict the processing activity in the network statically -- after the answer has been generated.

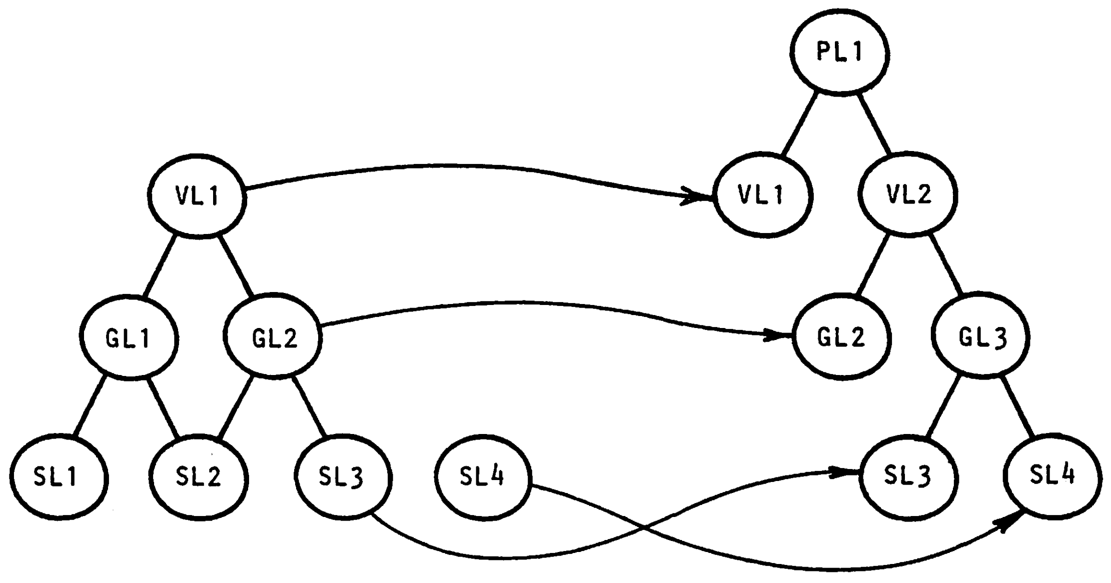


Figure 22: .Communication at the Signal, Group, and Vehicle Location Levels.

In this decomposition, Node:1 sends hypotheses SL3 and SL4 to Node:2 and processes hypotheses SL1 through SL3. Node:1 also sends hypothesis GL2 and VL1 to Node:2.

Suppose each knowledge source instantiation requires one time unit to execute plus one time unit for each support link found. In addition, suppose each communicated hypothesis requires two time units to send, five time units in transmission and two time units to receive. Table 2 shows a possible trace of activity for the Figure 22 decomposition. It illustrates several interesting situations.

First, the activities in Table 2 were ordered to generate the PL1 hypothesis in the minimum number of time units. Resequencing the activities at a node (such as Node:1 generating GL1 before GL2) can

Time	Node:1	Channel	Node:2
1	Send SL3	Idle	Idle
2	▼	▼	¦
3	SL2 & SL3 => GL2	SL3	¦
4	¦	¦	¦
5	▼	¦	¦
6	Send SL4	¦	¦
7	▼	▼	▼
8	SL1 & SL2 => GL1	SL4	Receive SL3
9	¦	¦	▼
10	▼	¦	Idle
11	Send GL2	¦	¦
12	▼	▼	▼
13	GL1 & GL2 => VL1	GL2	Receive SL4
14	¦	¦	▼
15	▼	¦	SL3 & SL4 => GL3
16	Send VL1	¦	¦
17	▼	▼	▼
18	Idle	VL1	Receive GL2
19	¦	¦	▼
20	¦	¦	GL2 & GL3 => VL2
21	¦	¦	¦
22	¦	▼	▼
23	¦	Idle	Receive VL1
24	¦	¦	▼
25	¦	¦	VL1 & VL2 => PL1
26	¦	¦	¦
27	▼	▼	▼

Table 2: Execution Trace for Figure 22.

Timing diagram showing the optimum execution ordering for our simple (but extended) example given that each knowledge source instantiation requires one time unit to execute plus one time unit for each support link found and that each communicated hypothesis requires two time units to send, five time units in transmission and two time units to receive.

increase the time needed to generate PL1.

Second, all knowledge source instantiations were executed only after all the possible supporting hypotheses were available. Recall from Section 2.3.4 that a knowledge source can synthesize a hypothesis using only a subset of the possible supporting hypotheses. A knowledge source that is executed before all the supporting hypotheses are generated would have to be reinvoked in order to incorporate these new supports into an output hypothesis of higher belief. Reexecuting knowledge sources would also increase the time needed to generate PL1. (A mechanism for delaying the execution of a knowledge source instantiation will be presented in Chapter IV.)

Third, the total time required in Table 2 is twenty-seven time units. This is the identical time required for the single node decomposition of Figure 17, given the above timing specifications. What has happened to our load-balancing? It has been eaten away by the time needed to communicate the low-level hypotheses from Node:1 to Node:2. The timing specifications have been chosen so that the two decompositions balance. If the timing tradeoff between communication and computation is shifted toward communication, the Figure 22 decomposition will be faster. If the timing tradeoff is shifted toward computation, the Figure 17 decomposition will be faster.

A closer look at Table 2 shows that the communication channel is the bottleneck. Increasing the channel capacity to allow two concurrent messages also favors the Figure 22 decomposition.

3.2.2 Some additional considerations.

The above example illustrates how sensitive a distributed problem solving network is to the use of a particular task decomposition, knowledge source processing requirements, and the characteristics of node interaction. The example also shows that by careful tailoring, effective and balanced problem solving activity can be achieved. However, the example was very simple and the tailoring was performed using complete, post hoc information about the activities required to generate the answer. How can an effective decomposition be determined when the grammar contains many patterns, the sensory data is noisy, and the actual pattern of vehicle movement in the environment is not known beforehand?

The above example also dealt only with location hypotheses. Recall that there are actually four possible synthesis paths from signal location hypotheses to pattern track hypotheses, each with potentially different resource requirements and coordination characteristics. An appropriate choice of synthesis paths depends upon the grammar, the sensory data, the communication channel capacity, and the anticipated activities at each node.

In networks composed of even a small number of nodes, a complete analysis to determine the detailed activities at each node is impractical. The computation and communication costs of determining the activities far outweigh the improvement in problem solving performance. Instead, coordination in distributed problem solving networks must sacrifice some potential improvement for a less complex coordination problem. What is desired is a balance between problem solving and coordination so that the combined cost of both activities are acceptable. Sproull incorporates a similar balance in his planning and solution optimal (p-optimal) planning strategy [SPRO77]. The emphasis is shifted from optimizing the activities in the network to achieving an acceptable performance level of the network as a whole.

As noted by March and Simon, most human problem solving (both individual and organizational) is similarly concerned with achieving a satisfactory performance level rather than an optimal one. Termed <u>satisficing</u>, this level of performance can be significantly less complex than optimizing. Determining if the activities in the network are optimal requires:

- a set of criteria permitting all alternative sequences of network activities to be compared;
- using these criteria to decide whether the particular sequence of network activities is preferred to all the alternatives;

while determining if the activities are satisfactory requires:

- a set of criteria describing minimally satisfactory performance levels;
- using these criteria to decide whether the particular sequence of network activities is minimally satisfactory.

March and Simon compare optimizing to "searching a haystack to find the <u>sharpest</u> needle" and satisficing to "searching the haystack to find a needle sharp enough to sew with" [MARC58].

In the next sections, two approaches to the network coordination problem are presented. These approaches do not attempt to optimize network activity by precisely coordinating the detailed activities of nodes but, instead, attempt to determine larger units of node activity with limited node interaction.

3.3 The Contract Net Approach

One approach to the coordination problem is the work of Smith and Davis on the contract net formalism [SMIT81]. The contract net formalism incorporates two major ideas.

The first idea is the use of negotiation between willing entities as a means of obtaining coherent behavior. Negotiation involves a multidirectional exchange of information between the interested parties, an evaluation of the information by each member from its own perspective, and final agreement by mutual selection. Negotiation differs from voting in that dissident members are free to exit the negotiation rather than being bound by the decision of the majority.

The second idea is the use of negotiation to establish a network of contracting control relationships between nodes in the distributed problem solving network. In the contract net formalism, nodes coordinate their activities through contracts to accomplish specific goals. These contracts are elaborated in a top-down manner; at each stage, a manager node decomposes its contracts into sub-contracts to be accomplished by other contractor nodes. This process involves a bidding protocol based on a two-way transfer of information to establish the nature of the sub-contracts and which node will perform a particular sub-contract. The elaboration procedure continues until a node can complete a contract without assistance.

The result of the contract elaboration process is a network of manager/contractor relationships distributed throughout the network. Each node knows for which nodes it is performing contracts and which nodes are performing contracts for it. However, a node does not know about the past, present, or future activities of other nodes outside its local contracting relationships (see below).

Smith and Davis have used a model of distributed problem solving in which the network passes through three phases as it solves a problem (Figure 23) [SMIT80].

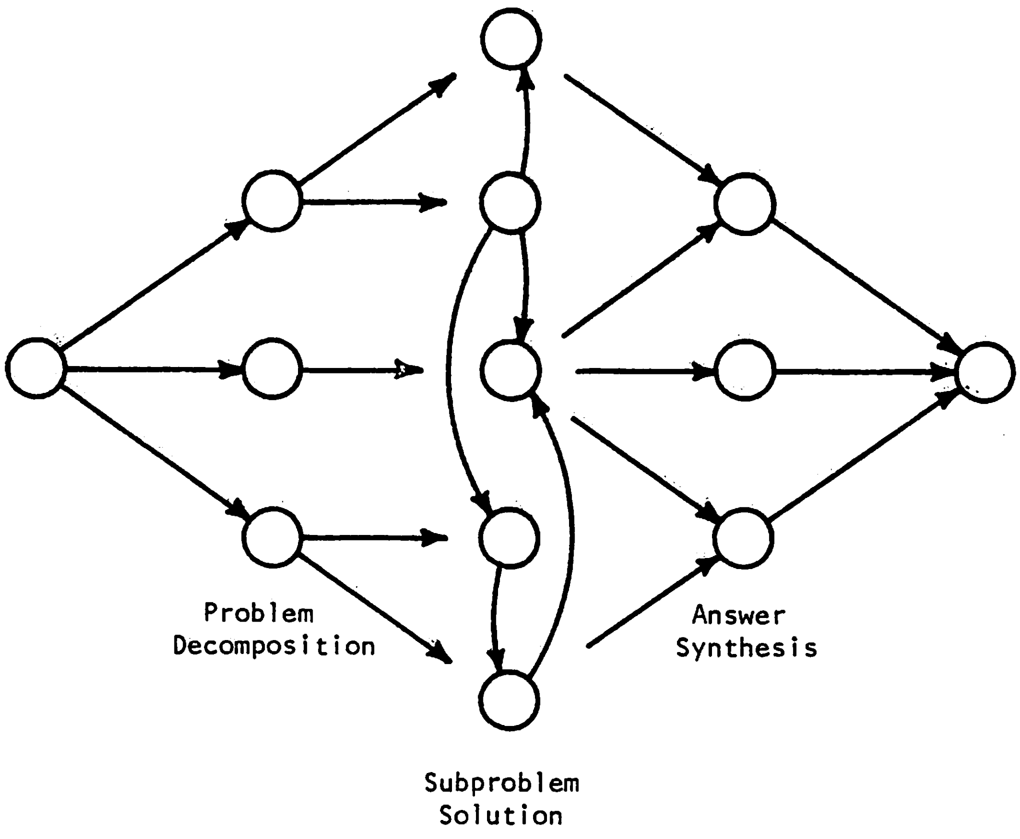


Figure 23: Distributed Problem Solving Phases (Smith and Davis).

The model of problem solving used by Smith and Davis first decomposes the problem into subproblems assigned to each node, then the distributed subproblems are solved, and the results are synthesized into an answer.

The first phase is problem decomposition. The problem solving task is recursively partitioned into increasingly smaller subtasks until atomic (non-decomposable) tasks remain. Part of this decomposition process is assignment of the subtasks to individual nodes. Smith calls this the connection problem [SMIT78].) Node assignment is particularly intertwined with problem decomposition. Different assignments may be best suited to different possible decompositions, and vice versa. This node assignment aspect of problem decomposition was made explicit by the inclusion of a distinct phase, subproblem distribution, in a later report by Davis and Smith [DAVI81].

The second phase in their model is the coordinated solution of the individual subproblems. Potential interactions with other nodes during the subproblem solution phase are specified by the elaborating nodes.

The third phase is answer synthesis using the results produced by the second phase. Part of the answer synthesis phase is assignment of synthesis activity to particular nodes. It should be noted that more than one node can have a solution to a particular subproblem and that not all such solutions are equally good. If the best subproblem solutions are to be used in the answer synthesis phase, the synthesizing nodes must locate and acquire these superior solutions. Therefore, the inclusion of a another phase, solution collection, would seem to be appropriate, given the inclusion of a distinct subproblem distribution phase.

The Smith and Davis problem solving model embodies the classic "divide-and-conquer" problem solving strategy, and the contract net formalism fits nicely into their model's problem decomposition phase.

However, their model simplifies some important aspects of problem solving in many distributed settings.

First, in applying their model to the contract net approach Smith and Davis imply that answer synthesis is the structural inverse of problem decomposition. This need not be the case. For example, in distributed vehicle monitoring a node might be given the task of generating a map at a second, distant node. Simple inversion of the decomposition structure would result in the generation of the map at the first node. But there is a subtler issue here. Even if the node given the task description is also to generate the map, is it necessarily the case that the answer is best synthesized by the same nodes that decomposed the task? If not, then control of the answer synthesis phase is not simply a matter of unwinding the problem decomposition and node assignment structure. Instead, it is a distinct control activity requiring additional contracting to establish the answer synthesis and result collection structure.

Second, their model suggests that the phases of problem solving are sequential.[2] Subproblem solution awaits problem decomposition and answer synthesis awaits subproblem solution. As illustrated by the example in Section 3.2, the control information required to perform these three phases does not always present itself in such a timely fashion. How can an appropriate decomposition of the problem be performed without prior knowledge of the distribution of vehicular

2. In the later report, mention is made that the activities could be performed in parallel, but the possibility is not elaborated [DAVI81].

activity (which is to be the result of the answer synthesis phase)? Unless supplied with knowledge beforehand, a particular decomposition can be very inappropriate to the actual problem solving situation.

Third, even if an appropriate initial decomposition is obtained, what happens if the environment or network capabilities change? In distributed vehicle monitoring, task assignments that work well at one point in the problem solving phase may work very badly at another, as patterns of vehicles move in the environment. The network may be able to cope with some environmental changes by reassigning subtasks to different nodes. However, the problem solving requirements may change to such a degree that the decomposition itself needs to be modified, and problem redecomposition is a much harder task than subproblem-to-node reassignment. Even modification of a portion of the problem decomposition can be difficult, because the manager/contractor relationships are distributed throughout the distributed problem solving network and each node knows only about its own manager/contractor activities.

Fourth, their model does not address the issue of controlling the three phases themselves. Each phase is, in fact, a distributed problem solving task in its own right and competes for network resources with the other phases. Some mechanism for balancing the effort expended among the phases is needed.

With these points in mind, the strengths and weaknesses of the contract network formalism become more apparent. The contract network formalism does not really address the issue of controlling the communication and processing activities of the problem solving and answer synthesis phases, but focuses instead on the problem decomposition phase. The idea is to use the hierarchical decomposition structure developed through the contracting protocols as a framework for coordination decisions made during the problem solving phase.

Contract nets are ideally suited to distributed problem solving situations where:

- o the information required to hierarchically decompose the task is available (or at least can be estimated) before subproblem solution;
- o the subproblem solution structure does not change significantly during the course of the solution;
- o answer synthesis is the inverse of problem decomposition;
- o the amount of effort expended in the decomposition and answer synthesis phases is small compared to the amount of basic problem solving performed in the problem solving phase.

When these conditions do not exist, the contract network formalism is less appropriate. The difficulty comes, not from the basic philosophy of negotiation, but from the top-down, contracting character of the contract network control structure, its focus on the problem decomposition phase, and each node's lack of a general view of the overall activities in the network. In the next section, we look at a complementary approach to network coordination which is bottom-up in character and which focuses on the problem solving and answer synthesis phases.

3.4 The Self-Directed Approach

In the absence of information needed to decompose, assign, collect, and synthesize an answer, it becomes important to let the dynamics of the problem solving itself dictate the coordination policies. This is the approach taken in early work by Lesser and Erman and by Lesser and Corkill [LESS80b, LESS81]. The idea is to develop the decomposition and answer synthesis structures concurrently and implicitly while performing the problem solving phase.

This is accomplished by having node activity be self-directed. Each node is given the overall task to be performed and reacts to incoming sensory data and messages received from other nodes. Each node performs as much of the overall task as it can, given its information. Using its own estimate of the state of network problem solving, each node determines which portions of the overall task it should perform and what information it should exchange with other nodes. Instead of assigning node activity (as in the contract network approach), the issues with the self-directed approach are initiating appropriate node activity and eliminating inappropriate node activity.

The choice of self-directed activities in a functionally accurate, cooperative network is potentially quite large because a node is able to choose a processing direction for which all the necessary data may not be available or consistent with other nodes. For instance, if a node does not receive an appropriate partial result in a given amount of time, it has the option to continue processing, utilizing whatever data are available at that time, or to choose some other processing direction which appears to be more beneficial.

Because the self-directed control decisions made by each node are based on the information available to each node rather than being elaborated from a complete decomposition structure, the nodes lack the top-down imposed coherence which characterizes the contract network approach. No single node may possess enough information to initiate a critical task. Furthermore, differences in local perspectives can cause the nodes to perform unnecessary, redundant, or incorrect processing. If these differences are not severe the amount of additional communication and processing resulting from inconsistent local control decisions may not seriously degrade network performance. This is not unreasonable in a functionally accurate, cooperative system, given that the additional data uncertainty caused by incorrect local control decisions may be resolvable by the same mechanisms used to resolve data uncertainties caused by incomplete local databases. Self-directed control has the added benefit of increased network robustness in the face of communication and node failure and increased network responsiveness to unexpected events because a node can immediately undertake activities without interacting with other nodes.

One way to obtain increased global coherence in the self-directed approach is to provide each node with an enlarged and more accurate view of the state of problem solving in the network so that its self-directed control decisions are more informed and consistent with those of other nodes. This can be accomplished by having nodes exchange meta-information about the state of their local problem-solving and what they have learned about the states of other nodes.

For example, one form of meta-information is the generation and exchange of a problem "decomposition" structure synthesized from the primitive subproblems generated by the self-directed activities at each node. One could imagine standing the contract net approach on its head, having nodes announce contracts for integrating their primitive subproblems into the developing problem "decomposition" structure. In this case, negotiation would be used to decide which locally generated subproblems should, in fact, be worked on by the nodes. The generation of explicit relationships among the primitive subproblems could also be used to indicate crucial activities that are not being performed by any node. The difficulty with such an approach is that significant effort may be required to determine how subproblems relate to one another and to the overall problem solving activity in the network. (Another approach to the development of this meta-information is discussed in Section 3.8.)

Self-directed control can be overly influenced by the local perspective of a single node. A node whose data incorrectly indicates a particular processing direction can adversely effect other nodes through a process Lesser and Erman termed distraction [LESS80b]. By sending highly rated, but incorrect, hypotheses to other nodes, the distracting node forces those nodes to attempt an integration of incorrect hypotheses into their developing hypothesis structure. Hopefully, the distracted nodes will eventually discover the inconsistency between the received hypotheses and their own sensory data, but a significant amount of processing may have been lost. Note that distraction is not restricted to the exchange of hypotheses. It can also occur in the contract net approach if nodes announce high priority, but inappropriate, tasks to be performed.

The potential for distraction in the self-directed approach can be reduced by lowering the beliefs of hypotheses received from other nodes in relation to hypotheses generated directly from a node's sensory data. Unfortunately, such damping of internode cooperation also reduces the effect of helpful, correct hypotheses and can lead to slower development of a solution or failure to develop a solution at all [BROO79, BROO83]. We will return to the problem of ignoring distracting information while quickly integrating accurate information in Chapter IV.

In summary, the self-directed approach to network coordination is best suited to distributed problem solving situations where:

- o at least one node has sufficient information to initiate every crucial activity (ensuring coverage of the problem);
- o only a small number of nodes have sufficient information to initiate any particular activity (reducing the potential for redundant network activity);
- o an appropriate problem decomposition structure is not known or computable beforehand;
- o the problem solving environment is changing significantly during problem solving.

As fewer of these conditions exist, a self-directed approach to network coordination becomes less appropriate. The difficulty comes from the lack of an elaborated decomposition structure and the lack of a general view of the activities in the network.

3.5 Reconciling the Two Approaches

The contract net approach to network coordination is plagued by the need to make control decisions before sufficient information for these decisions is available. On the other hand, the self-directed approach is overly influenced by the local perspective of each node and lacks the coherence and continuity of purpose provided by the contract net approach. Is there a way of achieving the best of both approaches?

On closer inspection, there is a strong similarity between these two network coordination approaches and the classic data-directed/model-directed problem solving dichotomy. When viewed from the level of the network coordination task (recall the multilevel system discussion of Chapter I), the contract net approach can be viewed as a model-directed approach to network coordination and the self-directed approach as a data-directed approach.

The contract net approach uses a priori task decomposition and node assignment information to determine how to decompose the problem solving task in an appropriate manner. To the extent that this information is available and accurate, a suitable decomposition for the problem solving phase is developed. The danger is that the decomposition may not match the actual problem solving requirements given the data. The network is coordinated, but inappropriate.

The self-directed approach, on the other hand, assumes the task decomposition and node assignment information is not available. Instead, primitive subproblems are generated from the receipt of sensory data and incoming messages. These subproblems are then synthesized (either implicitly or explicitly) into increasingly higher-level tasks until the overall task structure is developed. The danger here is that the low level tasks may not merge into cohesive network activity and that crucial low level tasks may not be worked on by any node (because a node needs a larger context in order to recognize the overall importance of a task). The network is appropriate, but uncoordinated.

What is needed is the ability to use whatever approach is best suited to the current problem solving situation, including the simultaneous application of both approaches on different aspects of the problem. Integrating both approaches into a common network coordination framework has several implications:

- o A higher-level control mechanism is needed for deciding which approach to use in a given situation.
- o There is no guarantee that the top-down and bottom-up problem "decomposition" structures developed by the two approaches will be compatible. There is a significant possibility of the two structures passing one another by instead of meeting in the middle. (Nilsson discusses at length the problems associated with combined top-down and bottom-up problem solving [NILS79].)
- o To reduce the likelihood of incompatible structures, the higher-level control mechanism should be able to modify the activities of both approaches toward a common ground. The meta-control mechanism must also determine when portions of each structure should be merged (similar to Nilsson's CANCEL operator [NILS79]).
- o The problem solving component of a node must be sensitive to both coordination approaches. If there are conflicts between the two approaches, the node must be able to reconcile them.

In short, the ability to use both approaches requires a sophisticated coordination component. In Chapter IV, an extension to the basic Hearsay-II architecture which integrates top-down and bottom-up local node control will be described.

In addition to differing in the direction that network coordination is elaborated, the contract net and self-directed approaches also differ in who makes the coordination decisions and in how these decisions are exchanged among nodes. In the next section these differences are elaborated into a general categorization of network coordination regimes.

3.6 Forms of Network Coordination

An important aspect of network coordination is who makes the control decisions. From the perspective of an individual node, control can be self-directed or externally-directed. Externally-directed means that the node is required to perform some action in response to the receipt of a message. In an externally-directed approach the network coordination decisions of a node are made externally, by other nodes having a (hopefully) superior view of the state of network activity. In a self-directed approach, nodes make their own network coordination decisions.

Between the externally-directed and self-directed approaches is a continuum of control schemes in which both local and external criteria are used in determining the activities of a node. One means of combining local and external criteria is through a weighted sum of the received (externally-directed) activity evaluation and the local (self-directed) activity evaluation. In this approach the local and external node are not necessarily in agreement, but both contribute to the activity decision through the weighted sum. The weighting controls the degree of external- versus self-direction and applies uniformly to all the activity decisions of the node. This is approach taken in this research.

Another means of combining local and external criteria is through explicit <u>negotiation</u>, where nodes interact to mutually agree on a coordination decision. The contract network formalism exemplifies the negotiated approach to network coordination decisions: an external node specifies the contract, but the local node determines whether to make a bid on it. The entire deliberation is subject to modification by both parties. In this case the goal is to bring nodes into agreement, using information specific to the particular activity decision.

How network coordination decisions are exchanged among nodes is also an important aspect of network coordination. Coordination decisions can be exchanged through:

<u>data</u> --	sensory data and processing results (hypotheses);
<u>goals</u> --	requests for the generation of results with specified attributes to be achieved by another node (much more regarding goals is presented in Chapter IV);
<u>tasks</u> --	specific actions (knowledge source instantiations) to be performed by another node.

These alternative network coordination regimes are illustrated in Table 3. The more precise the message (tasks are more precise than goals and goals are more precise than data) and the more externally-directed a node is, the more explicit the form of control.[3] Each control regime can be appropriate in a particular situation, and a

3. Smith and Davis have termed control implemented as the exchange of goals and tasks as <u>task sharing</u> and control implemented as the exchange of data and results as <u>result sharing</u> [SMIT81].

Table 3: Forms of Network Coordination.

The amount of influence a particular node has on another node's coordination decisions depends on who makes the decision (the particular node, the other node, or both nodes -- with or without negotiation) and on the form that decision takes (performing a specified task, achieving a specified goal, or working with specified data). A node has the most control on another node's decisions in the coordination regime at the upper right and the least control in the regime at the lower left.

	Data-directed	Goal-directed	Task-directed
Externally-directed	* receive evaluated data * generate goals from the data * prioritize the goals * determine tasks to achieve the goals * prioritize the tasks	* receive prioritized goals * determine tasks to achieve the goals * prioritize the tasks	* receive prioritized tasks
Negotiated	* receive data * negotiate an evaluation of the data * generate goals from the data * prioritize the goals * determine tasks to achieve the goals * prioritize the tasks	* receive goals * negotiate a priority for the goals * determine tasks to achieve the goals * prioritize the tasks	* receive tasks * negotiate a priority for the tasks
Self-directed	* receive data * evaluate the data * generate goals from the data * prioritize the goals * determine tasks to achieve the goals * prioritize the tasks	* receive goals * prioritize the goals * determine tasks to achieve the goals * prioritize the tasks	* receive tasks * prioritize the tasks

Table 3: Forms of Network Coordination.

truly general control framework should support the entire range of control forms. In fact, the entire range of control forms can be useful during the course of solving a single problem.

3.7 Functionally Accurate, Cooperative Network Coordination

In Section 2.2, the functionally accurate, cooperative approach to problem solving was introduced. Since the problem of coordinating network activity is characterized by the lack of timely coordination information, can the functionally accurate, cooperative problem solving approach be applied to the network coordination task as well?

Instead of ensuring that the activity decisions made by each node are completely consistent with those made by other nodes, nodes are allowed to make tentative and possibly incorrect coordination decisions using incomplete and inconsistent local views of the state of problem solving in the network and of the proposed activities of other nodes. If coordination errors are made, such as a node undertaking an activity before receiving all input information from another node or a node selecting an activity which is globally inappropriate, the tentative nature of the functionally accurate, cooperative style of problem solving reduces their impact. The intent is that the additional communication and computation caused by these local coordination decisions is less than would be required to maintain complete and consistent views of network problem solving activity and is, to some degree, offset by increases in parallel node activity.

Because the generation of the answer map in the distributed vehicle monitoring testbed is commutative, the network has substantial leeway before an inaccurate tentative result must be corrected. While failure to quickly correct an inaccurate result may delay the recognition of an important additional constraint by the network and thereby increase the time needed to generate the answer map, the major concern is correcting incorrect results before they are included in the eventual answer map.

Network coordination decisions are not commutative, but must cope with the constraints of limited time and limited resources. If a crucial aspect of the answer map is not quickly undertaken by at least one node in the network, the network can fail to generate the map in the required time. In the determination of node activities, mistakes can cause the loss of unrecoverable problem solving time and, if enough mistakes are made, can the network can fail to generate a timely answer map. If the nodes and sensors are mobile, their placement adds another non-commutative aspect to the distributed vehicle monitoring task. A misplaced node or sensor can require substantial time to be repositioned.

These constraints suggest that there is less room for mistakes in network coordination decisions than exists in performing the underlying network problem solving activities themselves. How much less is an interesting (and open) research issue. Individual coordination errors may not be very damaging in isolation, it is the compound effect of many individual coordination errors that must be avoided.

Functionally accurate, cooperative network coordination applies the concept of satisficing (Section 3.2) at the problem solving control level. It substitutes a less than perfect network coordination structure for a less complex coordination problem, at the risk of degrading the overall problem solving performance in the network. Functionally accurate, cooperative coordination is applicable to both the contract net and the self-directed coordination techniques. However, in order to be successful, functionally accurate, cooperative coordination must achieve the following conditions:

<u>coverage</u>	-- any given portion of the overall problem must be included in the activities of at least one node;
<u>connectivity</u>	-- nodes must interact in a manner which permits the covering activities to be developed and integrated into an overall solution;
<u>capability</u>	-- coverage and connectivity must be achievable within the communication and computation resource limitations of the network.

These conditions are more likely to be achieved if each node has a high-level view of how the network is attempting to solve the problem and how the node fits into this structure. Such a shared high-level view is missing in both the contract net and the self-directed approaches to network coordination. Its development is the subject of the next section.

3.8 Organizational Design as a Framework for Network Coordination

3.8.1 The need for organizational design.

Even using functionally accurate, cooperative control there can be insufficient communication and computation resources to effectively deal with the network coordination problem. An organizational structure, developed and maintained by the network, can be used to limit the range of coordination decisions which must be considered by a node and can provide a framework for obtaining an acceptable level of coverage, connectivity, and capability.

The organizational design approach to network coordination splits the network coordination problem into two concurrent activities:

1. construction and maintenance of an organizational structure;
2. continuous local elaboration of this structure into precise activities by the nodes.

Included in the organizational structure are control decisions that are not quickly outdated and that pertain to a large number of nodes. The organizational structure represents general "ballpark" control decisions which are dynamically tailored by the local, functionally accurate, cooperative control decisions of the nodes. The idea is to use the organizational structure to reduce the impact of non-commutative network coordination errors and to provide a high-level view of problem solving in the network by establishing a high-level set of node responsibilities and node interaction patterns that is available to all nodes.

At the organizational structuring level, coordination decisions involve determining an appropriate organizational structure for addressing the problem, assigning general responsibilities to individual nodes, patterning the use communication media and other shared network resources, and monitoring and possibly revising the structure of the organization as problem solving progresses. This level of coordination is sometimes termed "strategic" planning [ANTH65].

Within the organizational structure developed at the organizational level, network coordination is performed. Network coordination involves the achievement of specific task goals and the coordination of activities within the organizational structure. This level of coordination is sometimes termed "tactical" planning or "management and operational control" [ANTH65].

If the coordination problem is complex enough, development and maintenance of the organizational structure may itself need to be coordinated by developing levels of meta-organizational structure. However, we will consider one level of organizational structuring to be sufficient.

It is important to have a clear demarcation between organizational structuring and explicit coordination among nodes. Organizational structuring decisions represent general responsibilities that are possibly inaccurate and are subject to revision by the nodes. On the other hand, explicit network coordination decisions made among nodes represent short-term agreements to perform specific tasks for a specific period of time. For example, specifying that a node should identify vehicles within a particular area and forward their movements to a second node could be either a organizational decision or an explicit coordination decision. As an organizational decision, the node is being told that it should generally favor working in that area. There may be other nodes with overlapping responsibilities, and since the decision is a general one, vehicles may not even be expected within that area in the immediate future. As an explicit coordination decision, the node is being told that the second node is now expecting vehicle movement information from the node's specified area. Typically, this type of decision is an elaboration of the organizational decision. If a node cannot distinguish one form from the other, it cannot decide whether a particular decision is an organizational guideline or a specific short-term elaboration.

3.8.2 Organizational design versus network design.

Organizational design is different from the underlying network or system design. Network design involves determining the particular resources available to the network and is generally performed prior to the use of the network. In the distributed vehicle monitoring network, network design includes:

- o the number of nodes available for the task;
- o the locations of the nodes (if the nodes are immobile);
- o the sensors reporting to each node and their sensory characteristics;
- o the processing capabilities of each node (computational speed, memory size, specialized hardware);
- o the communication capabilities among nodes (channel bandwidth, delay, reliability);

o the long-term expert knowledge available to the network.

Organizational design involves determining how these network resources are to be applied to solve the problem. In the distributed vehicle monitoring network, organizational design decisions define the general responsibilities of each node and the pattern of communication among nodes.

Design choices such as adding more nodes as the need arises, increasing the processing speed of a node that is saturated, or acquiring new, external knowledge are not reasonable organizational decisions in distributed problem solving networks. With the exception of possible node and communication failure (and conversely, the addition of an occasional node or communication channel -- outside the control of the organizational designer) the network must view its resources as given and attempt to do the best with those resources.

This view differs significantly from the scope of organizational design in the management literature. Business organizational design includes the acquisition of additional resources as a basic organizational design option. If the demands on a business organization are high, overtime is an option that can temporarily increase the capacities of a given sized work force. A second option is the hiring of additional temporary employees to increase the size of the work force to handle the increased demands.[4] Finally, subcontracting a portion of the work load to an outside organization can be used to reduce the internal overload [MARC58, GALB77].

4. The success of temporary office worker services such as Manpower and Kelly Services testifies to the frequent use of this strategy.

3.8.3 The need for organizational change.

To be effective, the organizational structure must be based on the dynamics of the problem solving situation and the internal characteristics of the network. As these change, the network may need to change its organizational structure in order to maintain its effectiveness.

But organizational change can have negative consequences. Change disrupts the progress of problem solving activities and the flow of information in the network. Information and partially completed tasks may have to be transferred among nodes, consuming valuable communication resources. Processing time can also be lost as nodes await relocation of tasks and information and as organizational activities override basic problem solving. Even small changes in one part of the organizational structure can have significant effects on the network as a whole.

In order to enhance the positive aspects of organizational change while reducing its negative consequences, the basic organizational framework must be capable of providing (adapted from Kast and Rosenzweig [KAST74]):

- o enough _stability_ to allow effective problem solving;
- o enough _continuity_ to ensure orderly change;
- o enough _adaptability_ to react appropriately to external demands as well as changing internal conditions;
- o enough _innovativeness_ to allow the organization to initiate change when conditions warrant.

Of course, the network still has the problem of developing and maintaining the organizational structure.

3.8.4 When is organizational design worth the effort?.

Organizational design is not without cost. In particular, determining and implementing a highly effective structure may be prohibitively expensive. So it is appropriate to determine if the possible increase in organizational effectiveness justifies the cost.

If the task demands are light enough that all nodes are relatively idle and all communication channels have significant unused capacity, then almost any organizational structure will perform well. At the other extreme are task demands which are so great that no organizational structure can perform any useful work unless network resources are increased.[5] Between these two extremes are the situations where only highly appropriate organizational structures can meet the task demands. It is here that organizational structuring has the most effect on network productivity. These situations are also the most subject to organizational change, since the use of network resources are so tightly coupled to the task requirements and there are very few slack resources. Unfortunately, it is also precisely these situations when the network can least afford to spend resources reorganizing itself.

So, what are the most appropriate situations for organizational design? They should require enough network resources that the identification and implementation of an appropriate organizational structure improves the performance of the network. They should also leave enough network resources unused to enable organizational performance evaluation and change without seriously degrading network

5. Such situations are the province of Stanislaw Lem's satirical random scheme.

performance.

3.9 Skeptical Nodes

It is reasonable to ask if the functionally accurate, cooperative approach can be applied at the organizational structuring level. The answer is yes -- if the nodes are sufficiently aware of their own activities and the activities of other nodes and if the nodes are allowed to override a locally inappropriate organizational structure.

The activities of a node can arise internally, externally, and organizationally. Sensory data generate potential work for the node to perform. Similarly, interactions with other nodes generates potential work to perform. An organizational structure generates potential work through the responsibilities assigned to the node. In order to favor an externally or organizationally specified activity over of an internally generated one, the node must consider the external or organizational activity more important. Nodes are basically egocentric, but all nodes share the goal of accomplishing the interpretation task in the most effective manner. A node is ready to change its activities if "convinced" that such a change will improve network performance.

The degree to which a node needs to be convinced in order to change its activities places that node at a particular point in the self-directed/externally-directed/organizationally-directed space of coordination regimes. A completely self-directed node pursues only those activities which can be justified by its own direct knowledge of the problem solving situation. A completely externally-directed node pursues only those activities which are deemed important by other nodes. A completely organizationally-directed node pursues only those activities deemed important by the node's organizational responsibilities.

Neither extreme is well-suited to the uncertain environment of a distributed problem solving network. The self-directed extreme does not allow node activity to be determined by what other nodes know about the problem solving situation (for example: "If you would do THIS, it would really help us out."). The externally-directed extreme does not allow a node to ignore requests for activities which it knows are not appropriate ("I know doing THIS for you won't help us at all, so I'll ignore it."). The organizationally-directed extreme does not allow a node to ignore its organizational responsibilities in favor of other work ("I have much better work I could be doing than THESE responsibilities, so I'll ignore them.").

A degree of skepticism on the part of a node allows it to continue work on activities which are highly justified on the basis of local knowledge despite external or organizational requests for activities for which there is negative local evidence. This skepticism can lead to an increase in the network's ability to tolerate some error in control at both the organizational and network coordination levels. Inappropriate control decisions can be ignored by skeptical nodes which possess information to the contrary: a node with a unique perspective is not necessarily stifled by an uninformed majority. The degree of skepticism exhibited by a node should dynamically change according the node's certainty as to the network importance of its own locally generated activities: as the certainty of a node's own activities decreases, it should become more receptive to externally or organizationally generated activities; as a node becomes convinced of its own approach, it should become more skeptical of external or organizational activities which are in conflict with that approach.

In distributed problem solving networks composed of large numbers of nodes, the existence of idiosyncratic degrees of skepticism can further increase the robustness of the network. In situations where there exist two competing approaches (one advocated by the organization and one apparent to some of its members) individual variances in node skepticism will ensure that both approaches are pursued by the organization. The approach apparent to a portion of the node population is implicitly pursued, without the cost of making an explicit organizational decision. Of course this robustness comes at the price of uncontrolled expenditure of resources by the skeptical nodes. Reed and Lesser describe an analogous thresholding behavior theory based on genetic variation for the selection of activity by honey bees [REED80].

The notion of skeptical nodes also has an analog in an approach to understanding the motivation of individuals in business organizations. Called expectancy theory, Nadler and Lawler list the four assumptions of this approach [NADL77]:

1. Behavior is determined by a combination of forces in the individual and forces in the work environment — different work environments tend to produce different behavior in similar individuals just as dissimilar individuals tend to behave differently in similar environments.

2. Individuals make decisions about their own behavior in organizations — most of the behavior of individuals in organizations is the result of individuals' conscious

decisions.

3. Different individuals have different needs, desires, and goals -- these differences are not random but are based on the kinds of outcomes desired by the individuals.

4. Individuals make decisions among alternative plans of behavior based on their perceptions (expectations) of the degree to which a given behavior will lead to desired outcomes.

If the business enterprise is going badly and morale is low there is, as expected, a large amount of skepticism and dissension within the organization. As the situation is perceived to deteriorate, each member begins to look out for her own well-being, and her goals tend to become completely disjoint with those of the organization.

It might seem that if the business is going extremely well that skepticism and dissension would be at their lowest. This does not appear to be the case. When business is good, minor differences between the goals of the organization and an individual's needs and aspirations tend to be viewed as reasonable, given the health (and wealth) of the organization, and unlikely to lead to the organization's demise. In this situation, minor grievances tend to be openly displayed, rather than withheld.

It is when the organization is perceived to be just barely functioning, but when its members still believe in its eventual success, that skepticism and dissension are at their lowest. Failure to place the organization's goals ahead of individual goals, failure to follow the company line, is viewed as a threat to the functioning of the organization. If the members view the existence of the organization to be in their own best interests, survival of the organization is viewed as more important than minor individual differences.

Crane, in discussing biological and societal systems, considers the appropriateness of <u>dual-directed control</u> in which a balance between top-down (externally-directed) processes and bottom-up (self-directed) processes are maintained [CRAN78]. Node skepticism is a mechanism which can provide such a balance. Interestingly, Crane also discusses dual-directed control in the context of the individual as a balance between conscious and subconscious processes. It may be reasonable to view top-down (conscious) and bottom-up (subconscious) behavior as analogous dual-directed control at the level of a single network node.

Beer discusses a similar balance between the performance of potential elemental actions (which he terms the <u>operational force</u>) and the need for system viability and coordination (which he terms the <u>coherence force</u>) [BEER81].

Historically, the notion of decisionmaking using a combination of locally perceived information and decisions received from others goes back at least as far as the Kilmer, McCulloch, and Blum model of the reticular formation (an important control system in the brains of vertebrates) [KILM69]. Their model consists of a voting system of small neural regions, each of which sees a small portion of the environment and receives the decisions of a limited number of other regions. The voting strength of each region varies with an evaluation of its decisionmaking performance. In their model, activity was selected only when a suitable majority of the regions arrived at a consensus.

Node skepticism allows node activity to occur without such a consensus. When there is no agreement, the organization pursues a number of activities implicitly and in parallel (similar to a breadth-first search). As the number of agreeing nodes increases, organizational activity becomes both more specific and more explicitly controlled.

3.10 The Need for Sophisticated Nodes

This chapter presented approaches to the network coordination problem that attempt to deal with the problem of obtaining coordinated node activities with decentralized and interaction-limited decisionmaking. A common theme throughout this chapter is that sophisticated local control capabilities are necessary to achieve effective network coordination in a distributed problem solving environment. It is unreasonable to expect sophisticated network behavior in dynamic and uncertain problem solving situations with nodes having little knowledge of their own potential activities or the activities of their neighbors.

Nilsson has stated that the challenges posed by distributed artificial intelligence will contribute to (and may even be a prerequisite for) progress in "ordinary" artificial intelligence [NILS80b]. This is especially true for the control component of knowledge-based artificial intelligence problem solving systems. In the next chapter we find that the control capabilities of our Hearsay-II testbed node must be expanded if a node is to become aware of its own activities and organizational responsibilities.

Though this be madness, yet there is method in't.

— Shakespeare

CHAPTER IV

A FRAMEWORK FOR ORGANIZATIONAL COORDINATION

This chapter discusses the design and implementation of a prototype internal and network coordination component for the distributed vehicle monitoring testbed that integrates planning at both the organizational and problem solving levels. This coordination component incorporates two major developments: an extended, goal-directed Hearsay-II architecture for planning the local activities of each node and an overall testbed node architecture which interfaces organizational structuring decisions with the goal-directed Hearsay-II architecture. Single node and network experiments using this coordination component are presented in Chapter V.

The following section develops the goal-directed approach to control and introduces some of the issues involved with it. Section 4.2 details the implementation of the goal-directed Hearsay-II architecture in the testbed. Section 4.3 discusses some additional issues involved with goal-directed control, and Section 4.4 presents the organizational structuring portion of the overall control framework.

4.1 A Goal-Directed Hearsay-II Architecture

Effective internal and network coordination in the distributed vehicle monitoring testbed requires each node to assess its own state of problem solving activity and to elaborate the organizational structure into plans for using the network's limited processing and communication resources to the best advantage. To do this each node must be able to represent and reason about:

- the relationships among competing and cooperating knowledge source instantiations (both past and potential) that are working on different aspects and levels of the problem;
- the activities requested of the node by other nodes;
- the activities requested of the node by the organizational structure of the network;
- the computational and communication resources available to the node.

Although the data-directed, multilevel, cooperating knowledge source model of problem solving used in the original Hearsay-II architecture has many advantages, it is incapable of this level of reasoning or of effectively planning its interpretation activities.

For example, the control component needs to develop and reason about sequences of knowledge source activities relating to a particular approach to one aspect of the problem. This allows these activities to be scheduled as a coherent unit and to be eliminated as a unit if the approach proves unproductive. A second example is the implementation of an opportunistic scheduling strategy where a high-level partial solution, possibly received from another node, is used to focus the node on low-level activities required to solve the remainder of the problem (focus of attention through subgoaling). Other examples include selecting a specialized knowledge source to resolve the node's confusion over competing partial solutions and instantiating knowledge source activities to produce input data necessary for performing an important activity (precondition-action backchaining). All of these examples rely on the control component's ability to evaluate the the potential effects of knowledge source activities from a non-local context.

To remedy these limitations, an augmented version of the Hearsay-II architecture was developed and implemented in the testbed. This augmented architecture integrates data- and goal-directed control of knowledge source activity via the generation of goals from blackboard events. This architecture allows the relationships among knowledge source instantiations to be naturally represented by viewing knowledge source activity simultaneously from both a data-directed and a goal-directed perspective. The next section outlines the limitations of the original data-directed Hearsay-II architecture. Subsequent sections develop the goal-directed approach to control. Implementation of the goal-directed Hearsay-II architecture in the testbed is described in Section 4.2.

4.1.1 Limitations in Hearsay-II's data-directed approach to control.

Figure 24 presents a high-level schematic for data-directed control in Hearsay-II. Knowledge sources are invoked in response to particular kinds of changes on the blackboard, called blackboard events.[1]

1. The particular blackboard events used in the distributed vehicle monitoring testbed are described in Section 4.2.1.

Figure 24: Data-Directed Hearsay-II Architecture.

The knowledge source execution cycle in the data-directed Hearsay-II architecture begins with the execution of a knowledge source that makes changes to the blackboard. These changes are detected by the blackboard monitor which determines what additional knowledge sources should be executed in response to the changes. The blackboard monitor executes the precondition procedure of each of these knowledge sources to determine if a knowledge source should be instantiated. Instantiated knowledge sources are placed on the scheduling queue which is ordered by the scheduler based on the progress of problem solving in the system. When the currently executing knowledge source has completed, the top knowledge source on the queue is executed, and the cycle repeats.

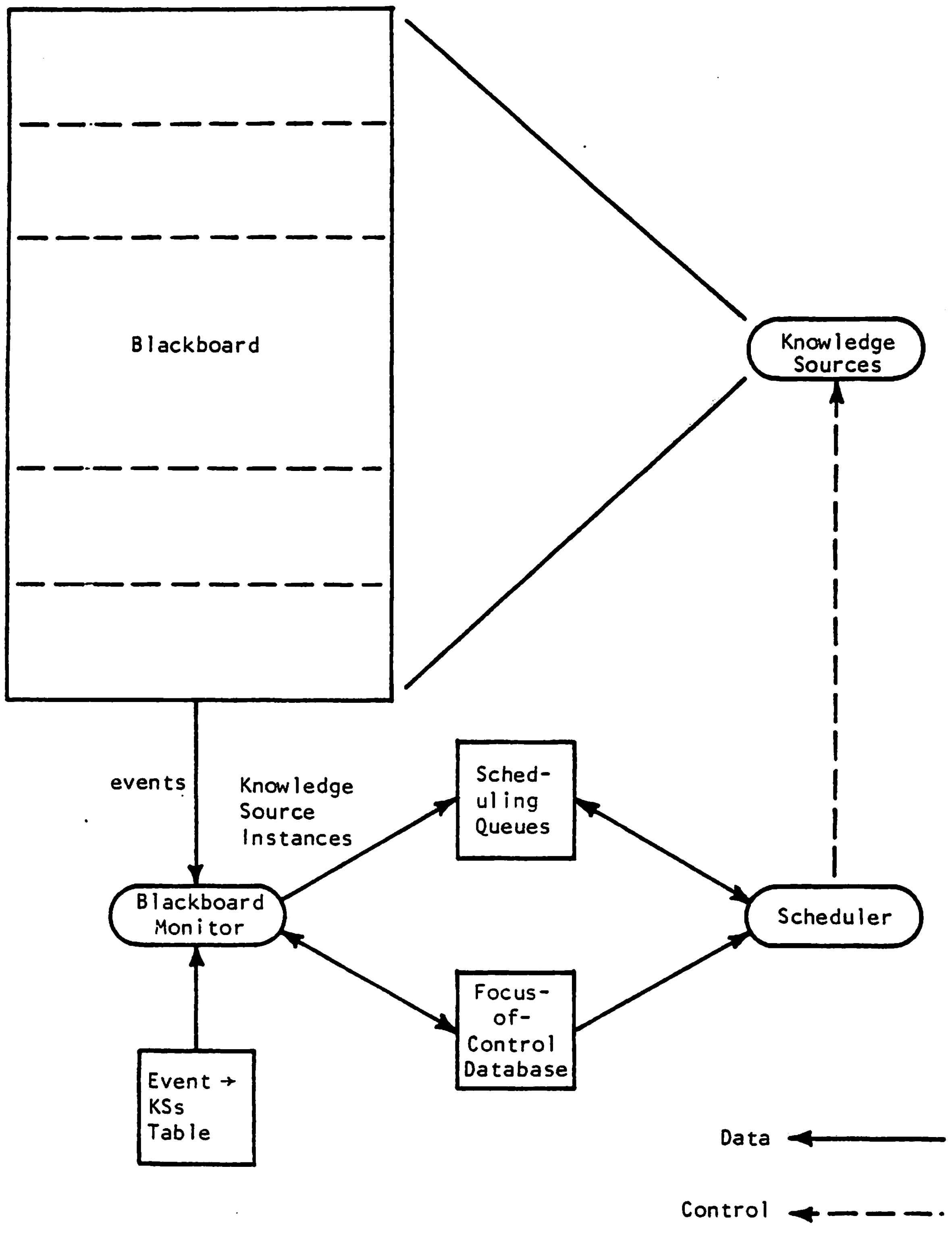


Figure 24: Data-Directed Hearsay-II Architecture.

Associated with each type of blackboard event is a set of knowledge sources that might be executable, based on the occurrence of that event type at a particular level on the blackboard. This information is contained in the blackboard event table. The blackboard monitor uses this table to determine which knowledge sources might be interested in the event. The occurrence of a blackboard event does not guarantee that there is, in fact, sufficient information on the blackboard for a knowledge source to be executed. To investigate further, the blackboard monitor executes a precondition procedure for each interested knowledge source to make a more detailed examination of the hypotheses on the blackboard. If sufficient information is found, a knowledge source instantiation is created and placed onto the scheduling queue.

The knowledge source instantiation includes the knowledge source's stimulus and response frames. The stimulus frame specifies the stimulus hypotheses responsible for the creation of the knowledge source instantiation and additional information located by the knowledge source's precondition process. This information is used as an input context when the knowledge source is executed, avoiding a second search of the blackboard. The response frame is an abstract description of the expected type of blackboard events that would result from executing the knowledge source (including each expected output hypothesis's blackboard level, the blackboard region in which the hypothesis is expected to reside, and its estimated belief). In planning terminology, the stimulus and response frames represent an operator model of the knowledge source instantiation.

The scheduler uses this stimulus/response frame characterization, together with measures of the overall state of problem solving contained in the focus-of-control database (kept current by the blackboard monitor), to calculate a priority rating for executing the knowledge source. The scheduler then selects the knowledge source instantiation with the highest priority rating from the scheduling queue to be executed.[2]

This approach to scheduling is instantaneous -- only the immediate effects on the state of problem-solving, as specified in the response frame, are considered by the scheduler in determining a priority rating for the knowledge source instantiation [HAYE77]. Although the rating is reevaluated if the knowledge source instantiation's stimulus hypotheses are modified or if the relationship between the response frame and the focus of control database changes, no attempt is made to determine the effects of executing a knowledge source instantiation beyond its immediate changes to the blackboard.

Another limitation in this type of scheduling occurs when a knowledge source's precondition procedure cannot find sufficient information for the knowledge source to be instantiated. No record of the missing information is made and there is no mechanism for reevaluating the priorities of a pending knowledge source instantiation or for adding a new knowledge source instantiation to the scheduling queue to potentially generate the missing information.

2. Erman, et al. describe in detail the knowledge source execution cycle used in the data-directed Hearsay-II speech understanding system [ERMA80].

Suppose, for example, that a vehicle track hypothesis has been created and the node would like to extend it by executing the EF:VL/VT:VT knowledge source. Extension requires the existence of a suitable vehicle location hypothesis before it can be released for execution. If such a <u>precondition hypothesis</u> does not exist on the blackboard, either the knowledge source instantiation must await data-directed activity to create the vehicle location hypothesis or the knowledge source instantiation is discarded to be recreated as a result of the eventual creation of the precondition hypothesis. The first approach was used in the "problems list" of SU/X and SU/P[3] [NII78] and the second approach was used in the Hearsay-II speech understanding system [LESS77].

The purely data-directed approach to control assumes that if the information is truly important enough to be of use in developing the solution it will eventually be generated based on normal scheduling considerations. This assumption is clearly invalid in a distributed network. If a node has one important stimulus hypothesis for a knowledge source and a second node has the other necessary input hypothesis, both nodes will wait forever for the "missing" information. What is needed is a more active approach to the acquisition of the information.

This data-directed, instantaneous approach to control was sufficient for the eventual (C2) knowledge source configuration used in the Hearsay-II speech understanding system, which was truly

3. This system has been renamed HASP/SIAP [NII82].

data-directed at a only a single level, had all sensory data available at the beginning of data-directed processing, had only a single knowledge source to perform a particular action given a particular blackboard event, and was not distributed among a number of nodes [ERMA80]. However, its limitations have become apparent as the architecture has been used in domains requiring more sophisticated control of knowledge source activities.

Engelmore and Nii with SU/P, Nii and Feigenbaum with SU/X, and Erman, et al., with Hearsay-III recognized these limitations and consequently have developed systems with enhanced control capabilities [ENGE77, NII78, ERMA81]. These enhancements permit more sophisticated control over scheduling by allowing the scheduling queue to be manipulated under program control. However, these enhancements do not go far enough for our needs in explicitly representing the relationships among knowledge source instantiations to provide an adequate framework for reasoning about control decisions in a general, domain-independent manner. (Work on reducing this deficiency in Hearsay-III is currently underway [BARN82].)

The premise behind the goal-directed Hearsay-II architecture is that the relationships among knowledge source instantiations can be naturally represented when knowledge source activity is viewed simultaneously from a data-directed and a goal-directed perspective. The next section describes how data- and goal-directed control can be integrated into a single, uniform framework.

4.1.2 Adding goal-directed control to Hearsay-II.

Figure 25 presents a high-level schematic of Hearsay-II as modified to accommodate goal-directed control. A second blackboard, the goal blackboard, is added that mirrors the original (data) blackboard in dimensionality. The goal blackboard contains goals, each representing a request to create a particular state of hypotheses on the data blackboard in the corresponding area covered by the goal. For example, a simple goal would be a request for the creation of a vehicle location hypothesis above a given belief in a specified area of the data blackboard.

The integration of data-directed and goal-directed control into a single framework is based on the following observation:

> The stimulation of a precondition process in the data-directed architecture not only indicates that it may be possible to execute the knowledge source, but that it is desirable to do so in order to achieve the goal implicit in the response frame.

In order to make these implicit goals explicit, the event-to-knowledge-sources mapping contained in the blackboard event table is split into two steps: event-to-goals and goal-to-knowledge-sources. The blackboard monitor watches for the occurrence of a data blackboard event, but instead of immediately placing knowledge source instantiations on the scheduling queue, it uses the event-to-goals mapping to determine the appropriate goals to generate from the event and inserts them onto the goal blackboard. These goals explicitly represent the intended effects of each knowledge source instantiation. These effects were implicitly represented in the event-to-knowledge-sources mapping used by the blackboard monitor in the

Figure 25: Goal-Directed Hearsay-II Architecture.

Hearsay-II as modified to accommodate goal-directed control includes a second, goal, blackboard that mirrors the original (data) blackboard in dimensionality and a full-fledged planner. The blackboard monitor responds to changed on the data blackboard by inserting goals onto the goal blackboard. The planner responds to these goals by developing plans for their achievement and instantiates knowledge sources to perform the plans. The scheduler uses the relationships between the knowledge source instantiations and the goals on the goal blackboard as a basis for its scheduling decisions.

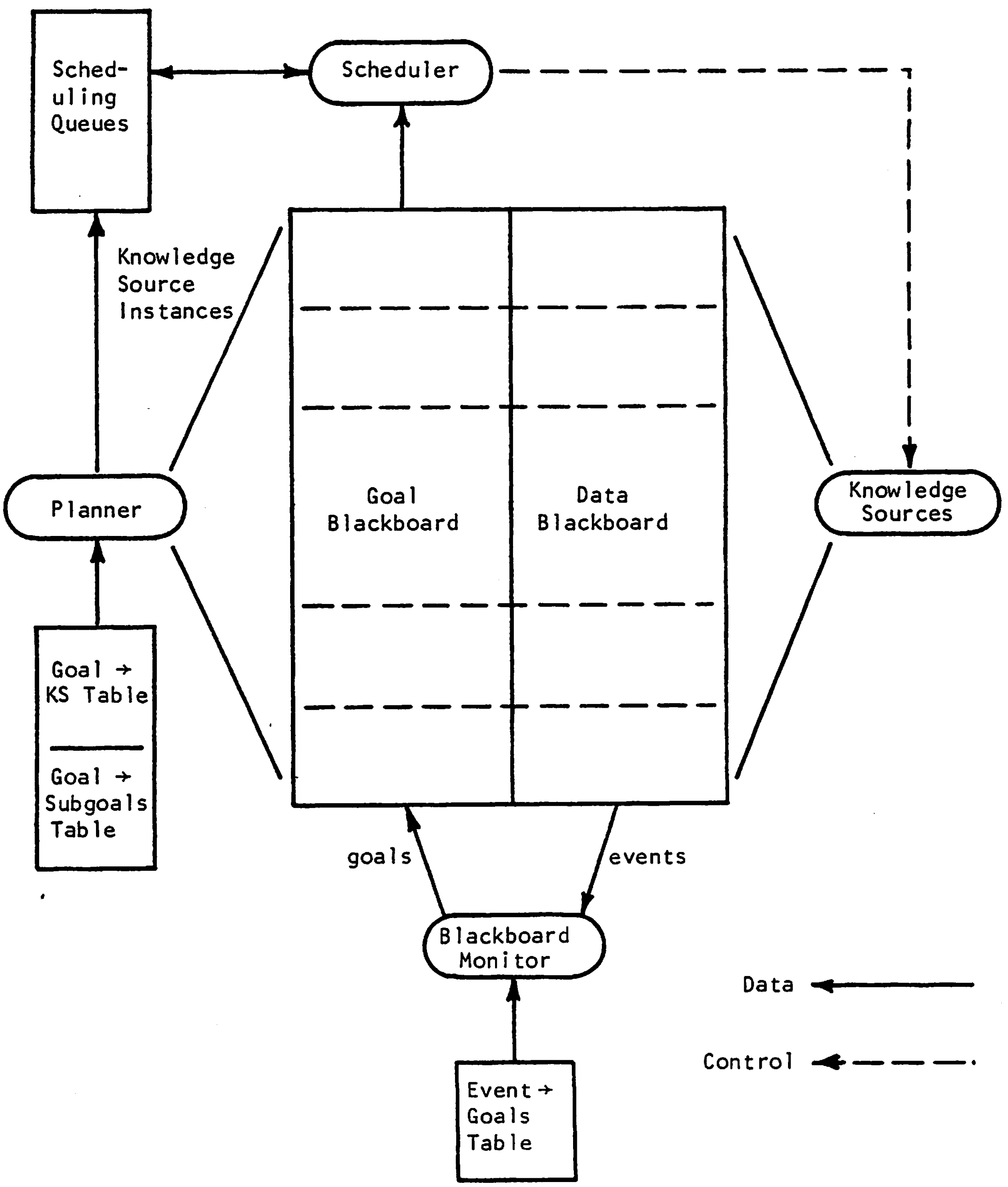


Figure 25: Goal-Directed Hearsay-II Architecture.

data-directed architecture.

Goals may also be placed on the goal blackboard from external sources. Placing a high-level goal onto the goal blackboard of a node can effectively bias the node toward developing a solution in a particular way. As we shall see, the exchange of goals between goal-directed Hearsay-II nodes provides an effective means of coordination (recall the forms of network coordination of Section 3.6).

A new control component, the planner, is also added to the architecture. The planner responds to the insertion of goals on the goal blackboard by developing plans for their achievement. The goal-to-knowledge-sources mapping is used by the planner to instantiate knowledge sources to potentially satisfy the goals. The scheduler then uses the relationships between the knowledge source instantiations and the goals on the goal blackboard as a basis for its scheduling decisions.

Bottom-up goal-directed processing.

We first look at how a simple, bottom-up, vehicle location synthesis sequence looks in the goal-directed architecture. For simplicity, assume that the grammar contains only one signal and group for the vehicle, that each blackboard event triggers only one goal, and that only one knowledge source can satisfy each goal.

Initially, signal location hypothesis H:SL is inserted onto the data blackboard on level DBB:SL (Figure 26a). That event causes the blackboard monitor to create goal G:GL on goal blackboard level GBB:GL (Figure 26b). G:GL is a request to create a hypothesis supported by

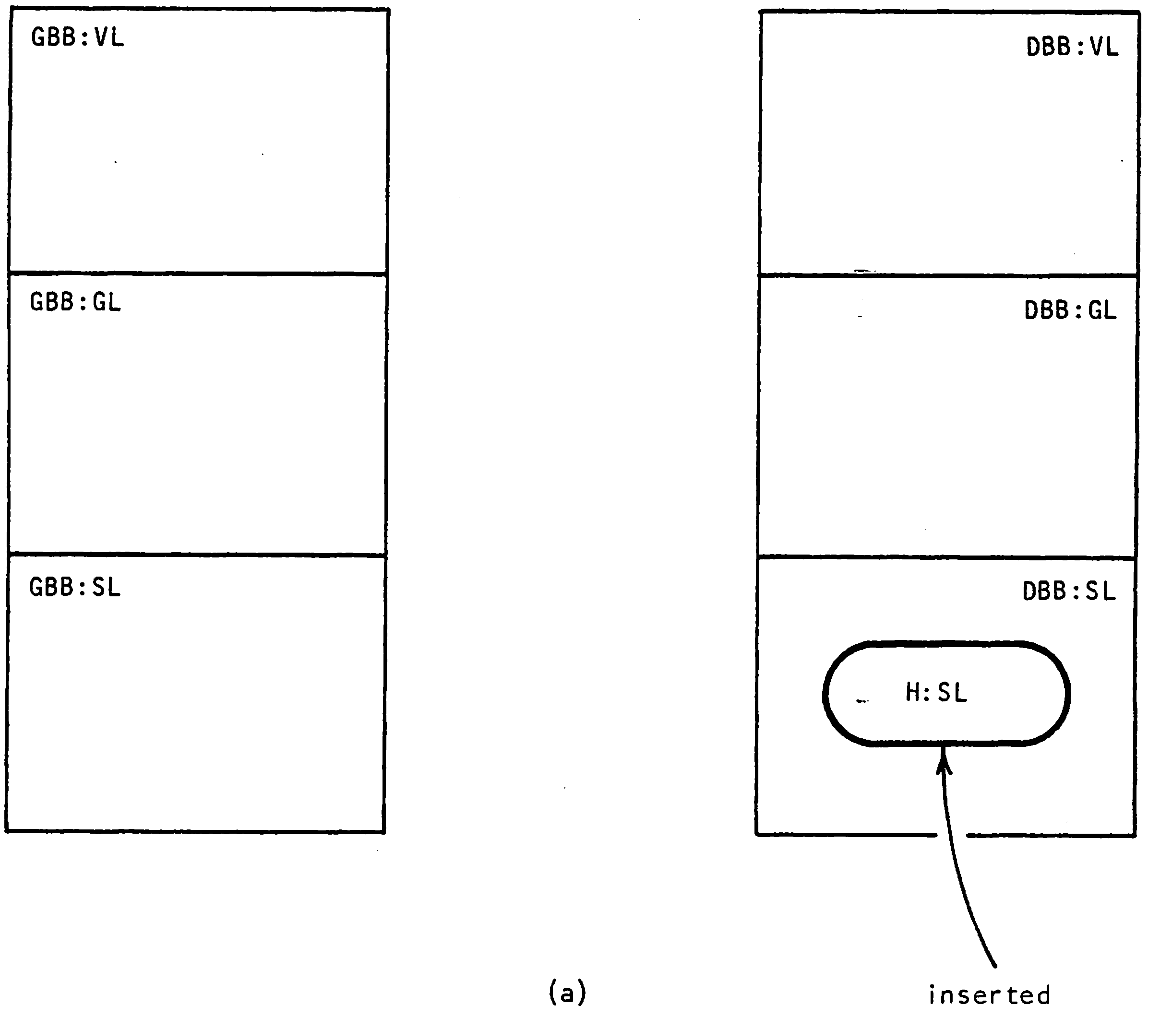


Figure 26: Bottom-Up Goal-Directed Processing.

The sequence of goal-directed activities used in the bottom-up synthesis of a vehicle location hypothesis from a signal location hypothesis.

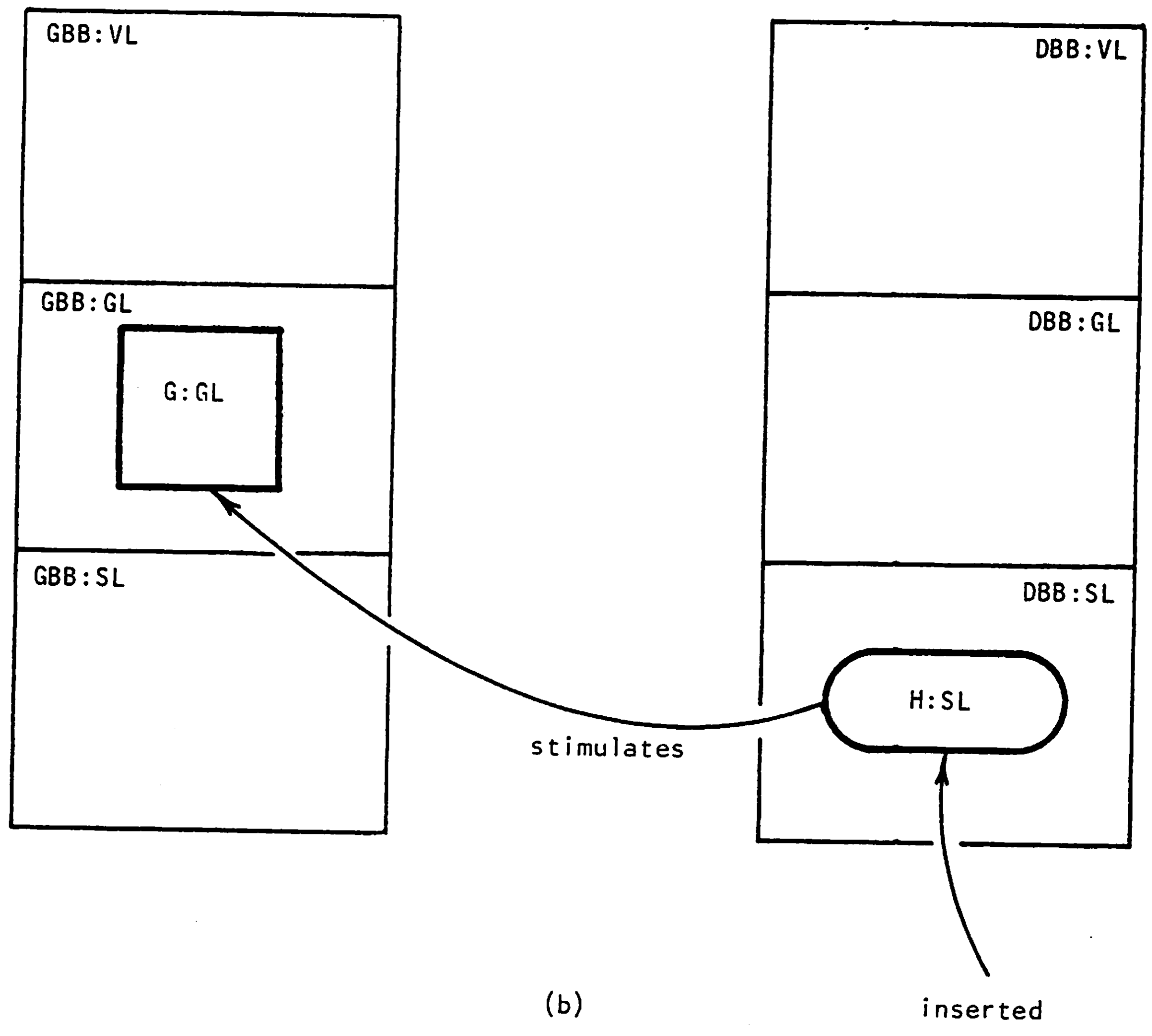


Figure 26: Continued.

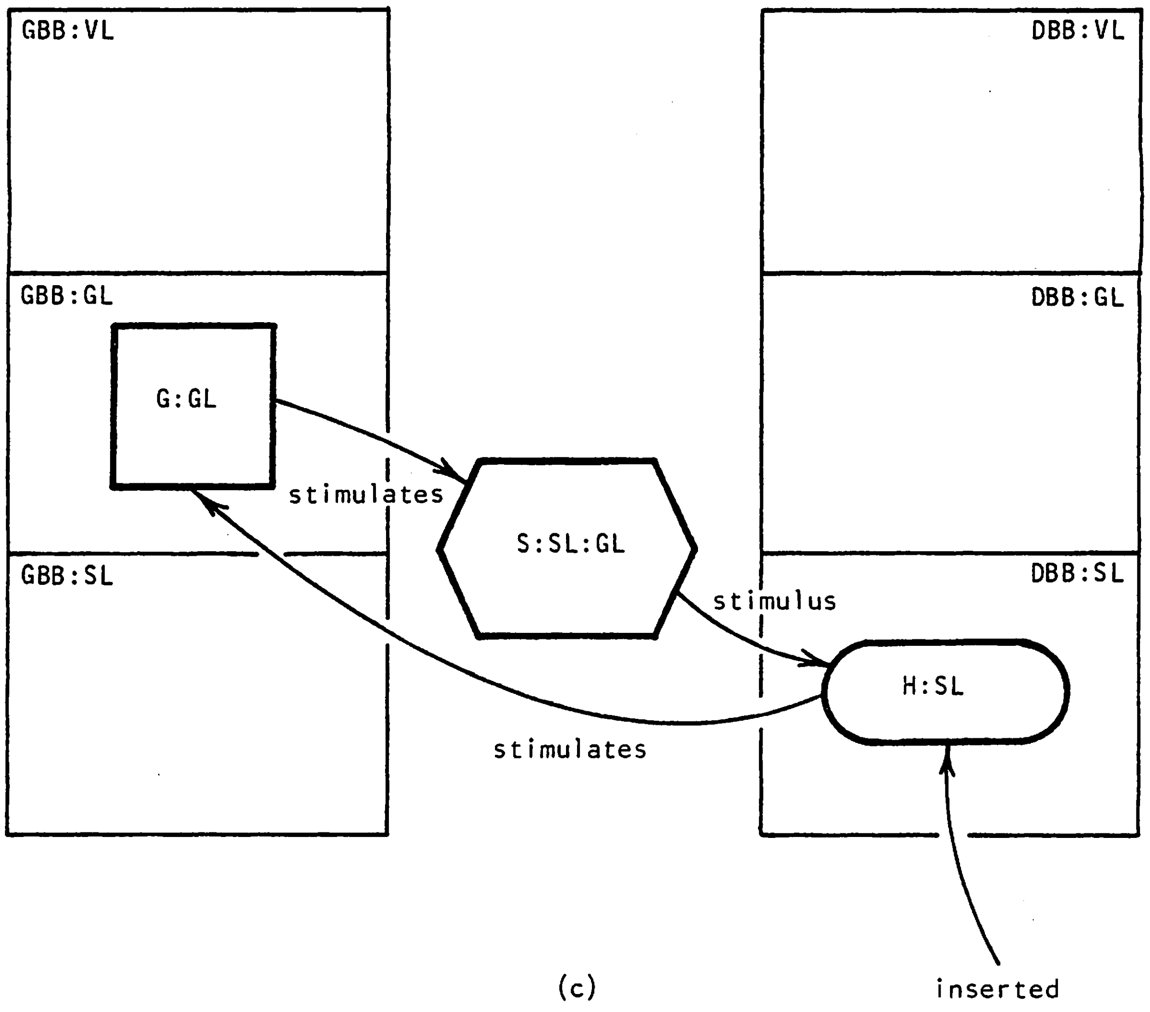


Figure 26: Continued.

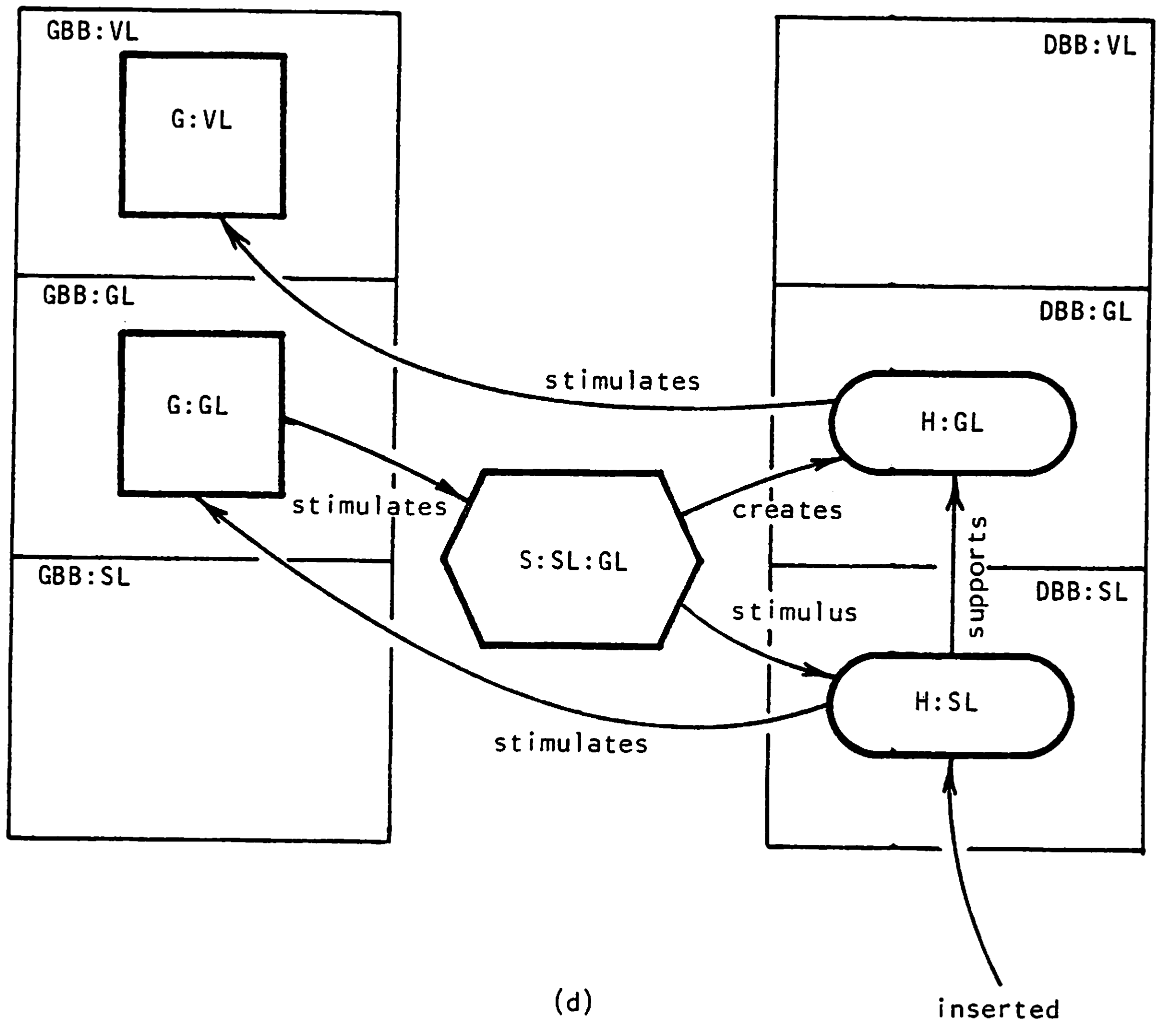


Figure 26: Continued.

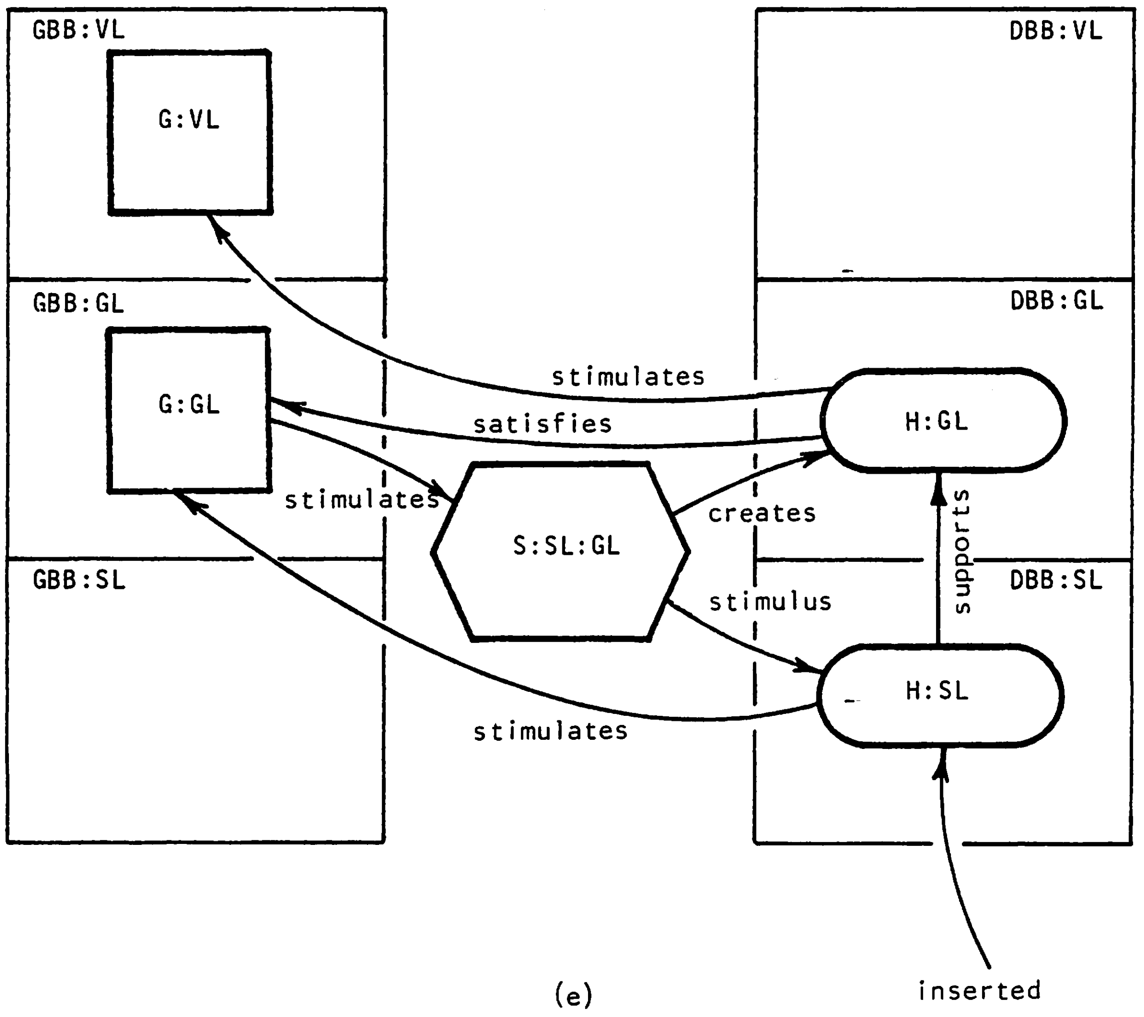


Figure 26: Continued.

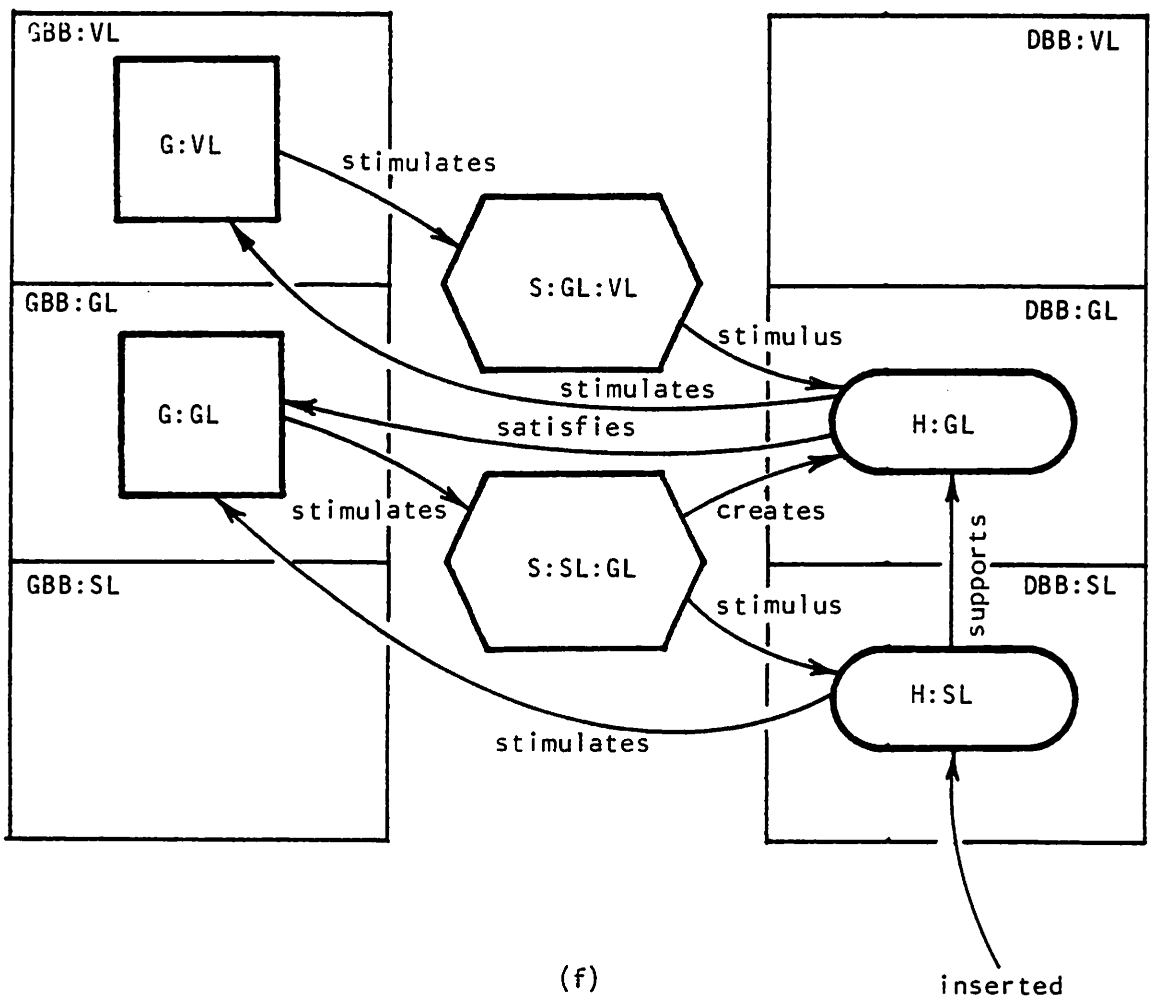


Figure 26: Continued.

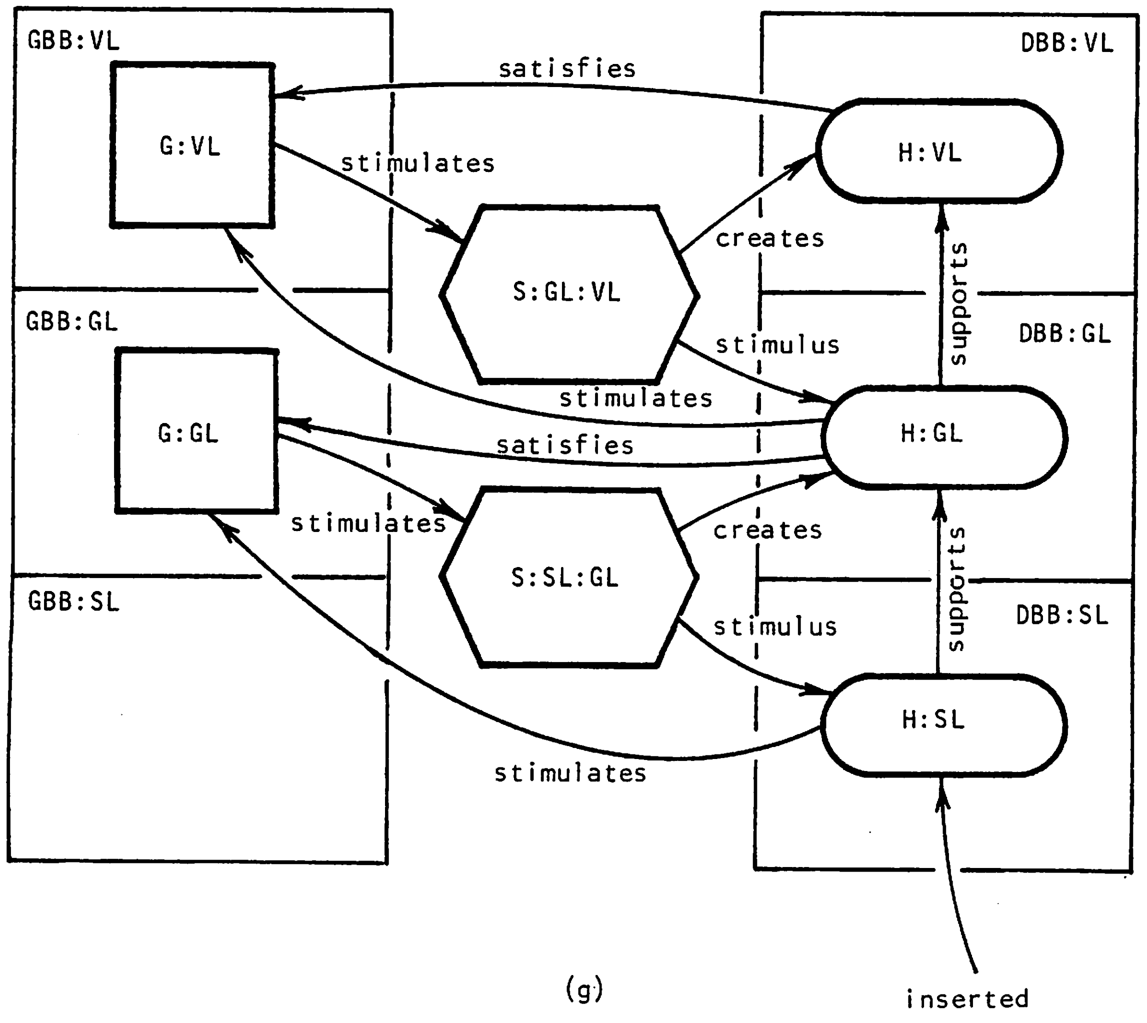


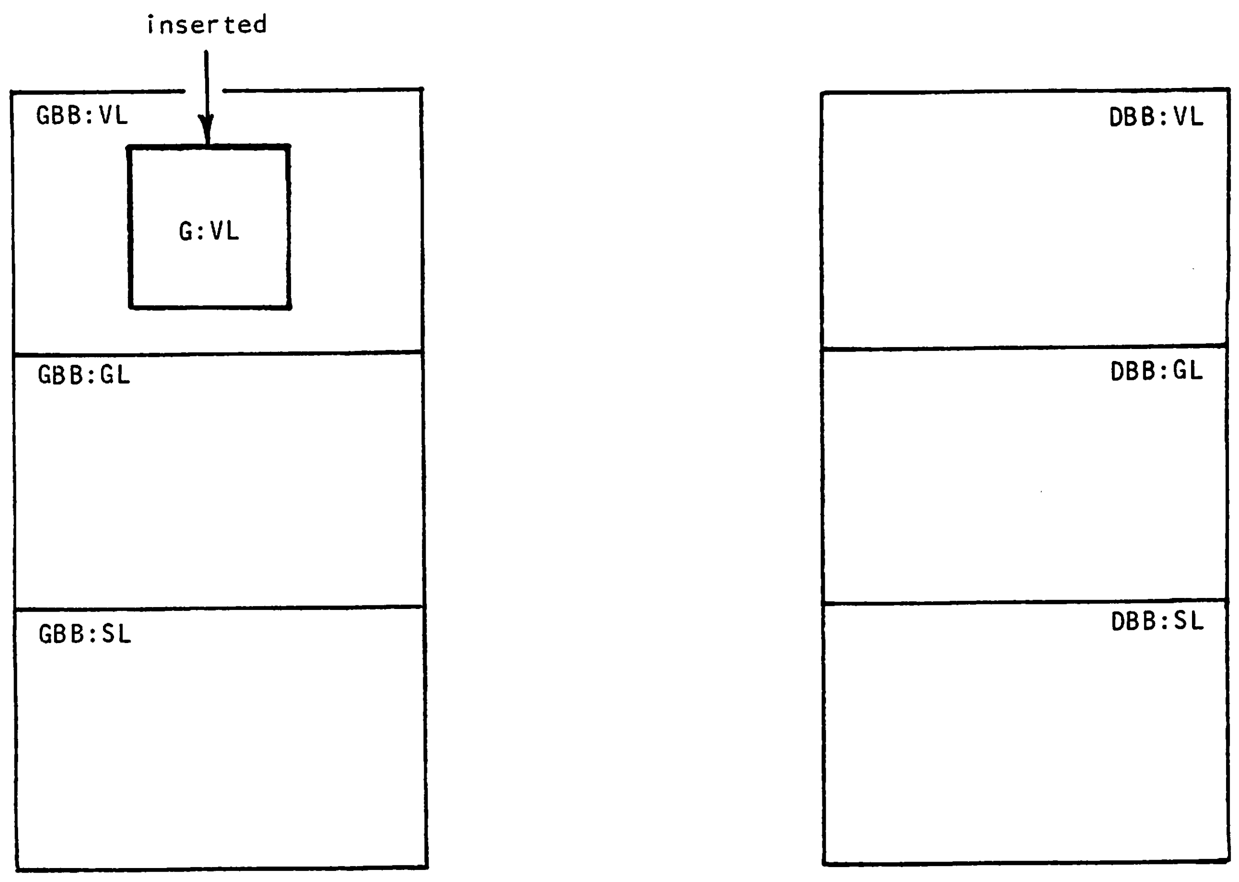
Figure 26: Continued.

H:SL on DBB:GL. The planner then looks at G:GL and instantiates knowledge source S:SL:GL with H:SL as its stimulus and places it onto the scheduling queue (Figure 26c). When S:SL:GL executes, it creates hypothesis H:GL. The blackboard monitor responds to this event and creates goal G:VL (Figure 26d). It also checks to see if hypothesis H:GL can help to satisfy any goals on the goal blackboard. In this case, H:GL completely satisfies goal G:GL (Figure 26e).[4] The planner next instantiates S:GL:VL to achieve G:VL and places it onto the scheduling queue (Figure 26f). When S:GL:VL executes it creates H:VL and the blackboard monitor marks G:VL as satisfied (Figure 26g).

Precondition-action backward chaining.

In the above example, all necessary low-level hypotheses exist when each knowledge source is instantiated. Instead, suppose the goal G:VL is inserted first before any hypotheses on DBB:SL (Figure 27a). Given only synthesis knowledge sources, a data-directed system must wait until low-level hypotheses are inserted on DBB:SL before instantiating knowledge sources. On the other hand, the planner in the goal-directed system can immediately proceed to instantiate S:GL:VL (Figure 27b). The planner is provided with an inverse operator model (knowledge-source-to-precondition-goals) of each knowledge source, which it uses to determine that precondition goal G:GL must be achieved before S:GL:VL can be executed (Figure 27c). To achieve G:GL, the planner instantiates S:SL:GL which requires G:SL to be satisfied before it can

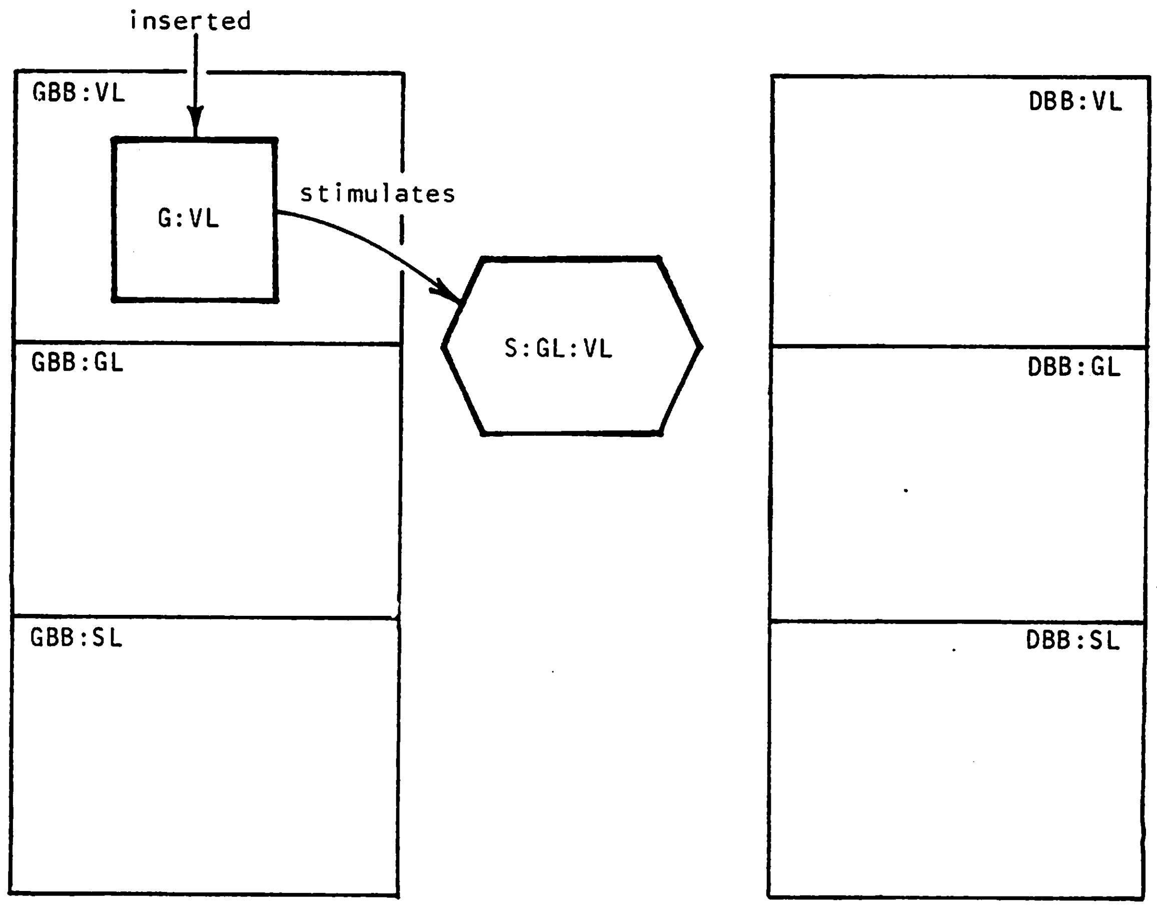
4. Goal satisfaction is a tricky issue, see Section 4.3.3.



(a)

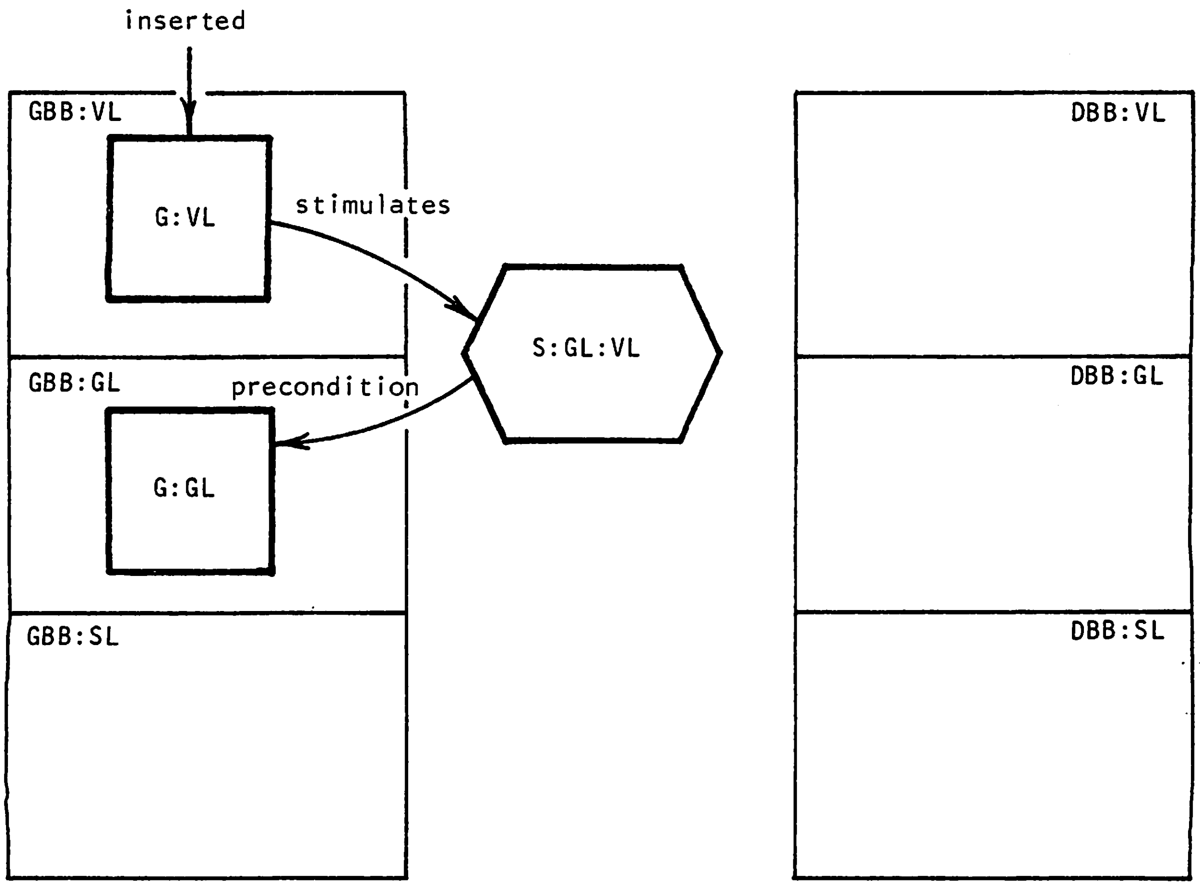
Figure 27: Precondition-Action Backward Chaining.

The sequence of activities when the preconditions necessary for synthesizing a vehicle location hypothesis are not initially satisfied on the data blackboard.



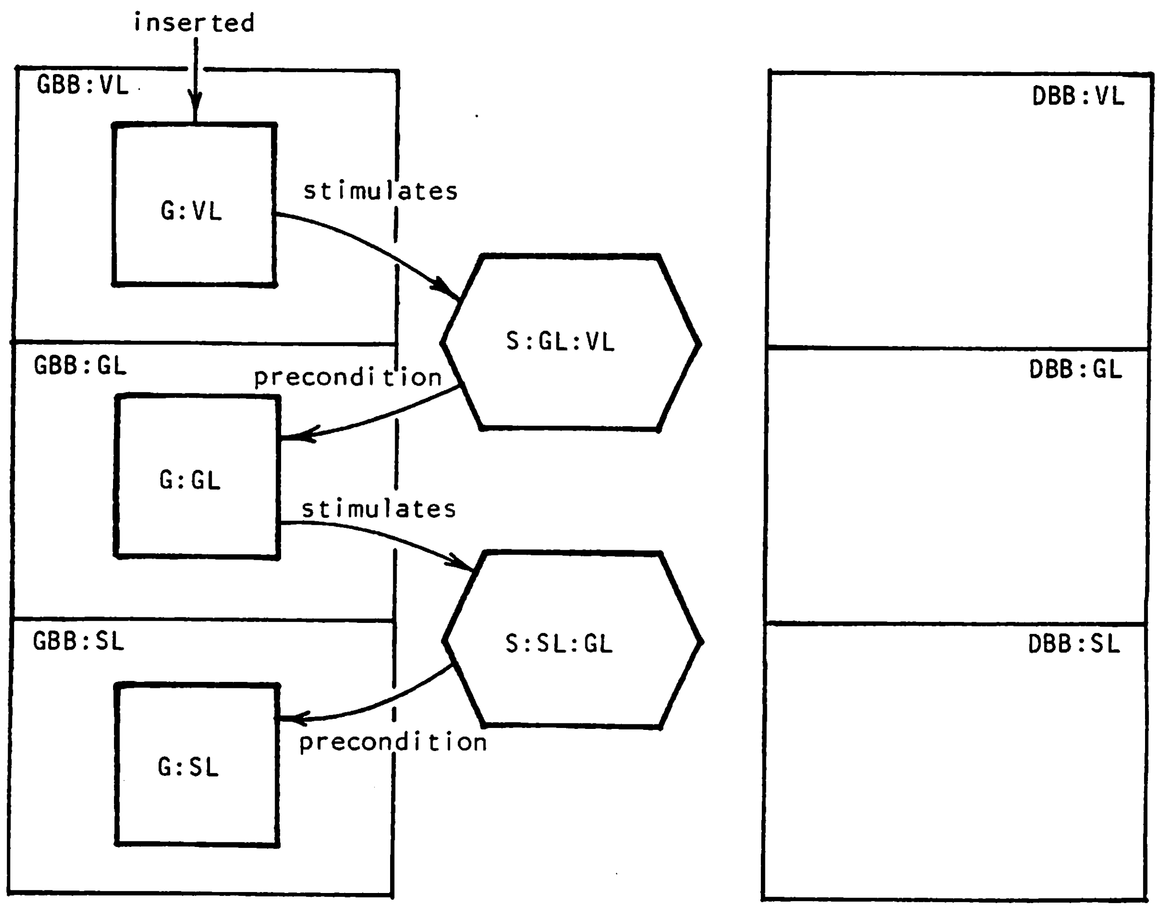
(b)

Figure 27: Continued.



(c)

Figure 27: Continued.



(d)

Figure 27: Continued.

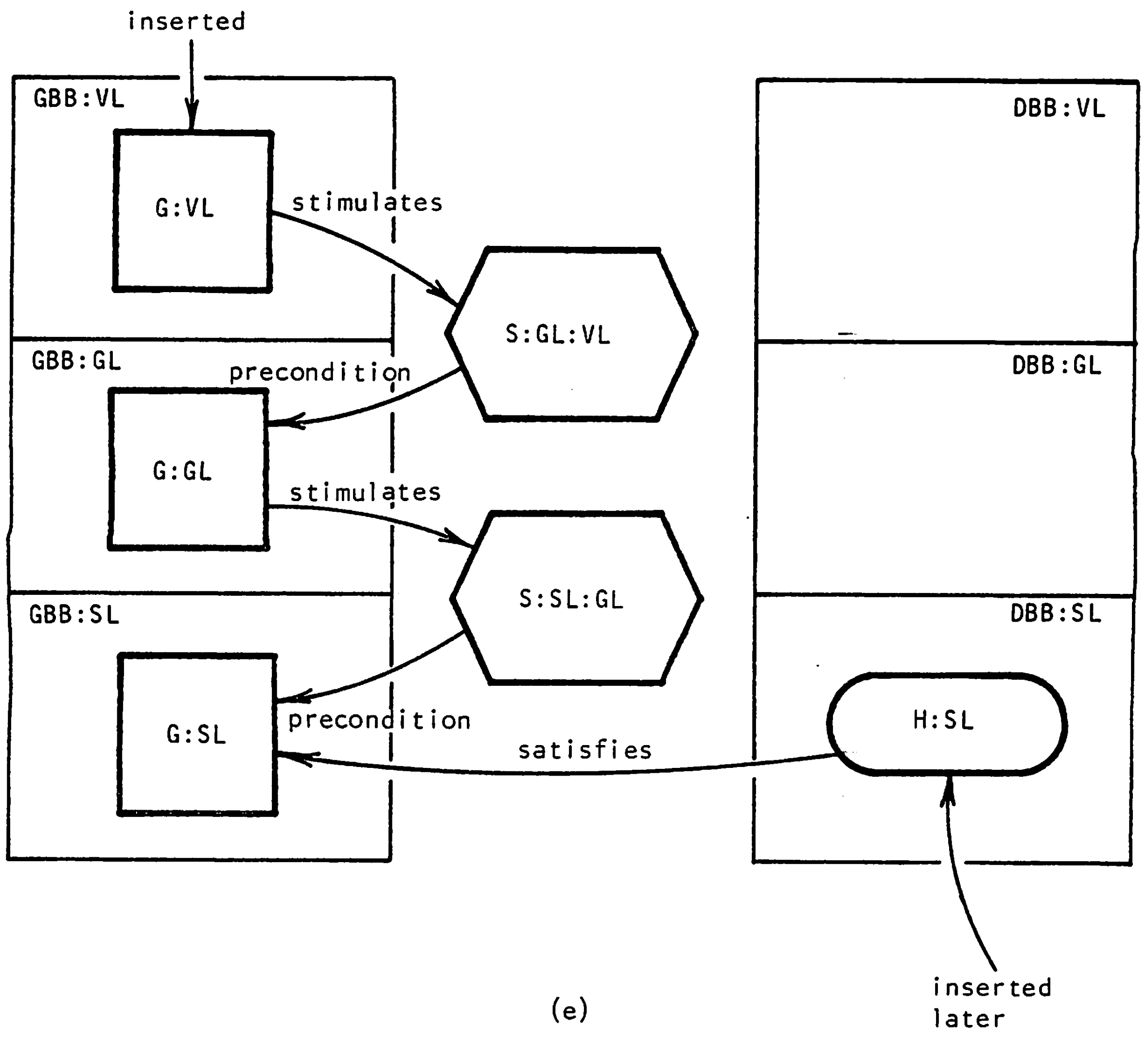


Figure 27: Continued.

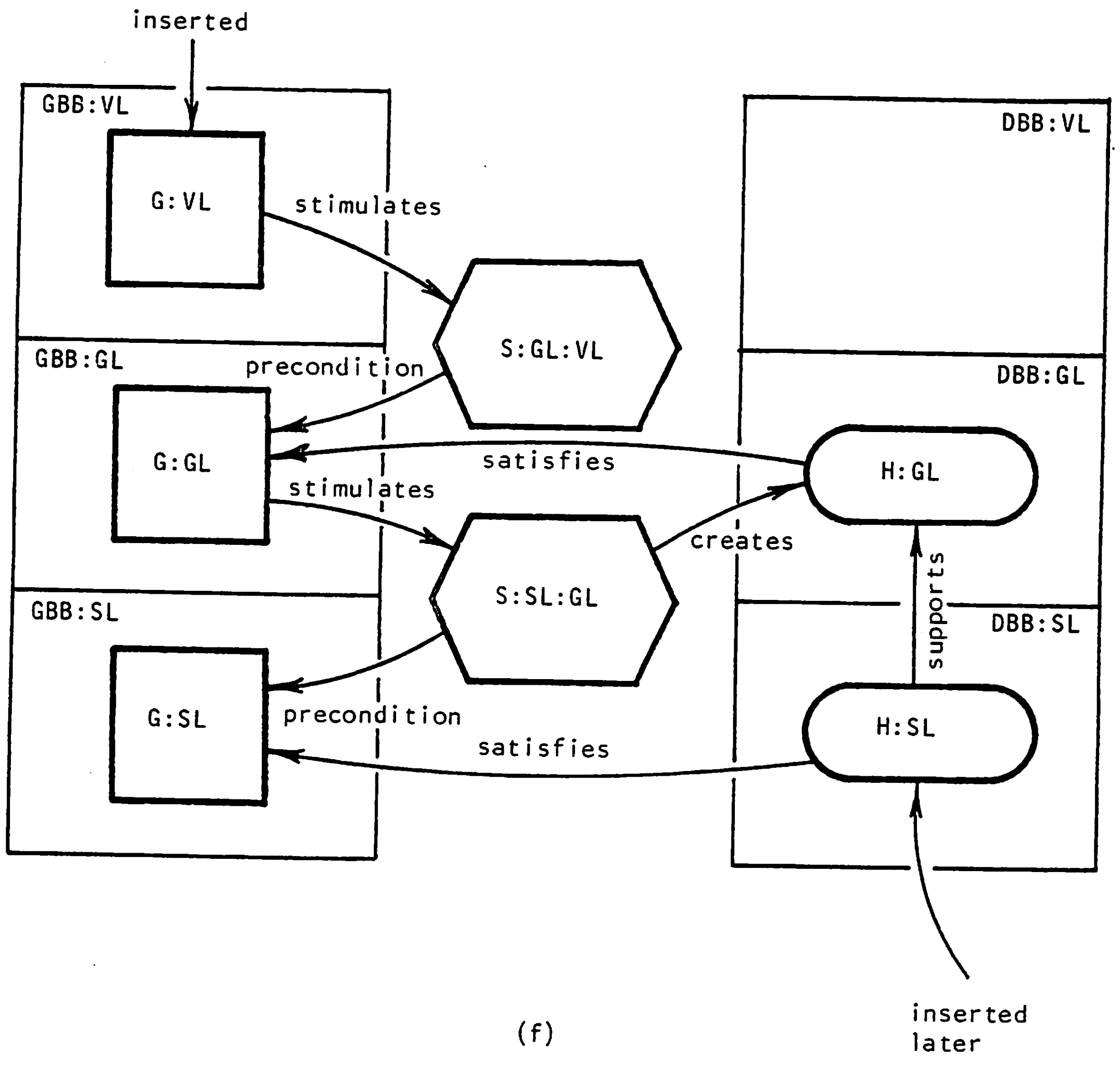
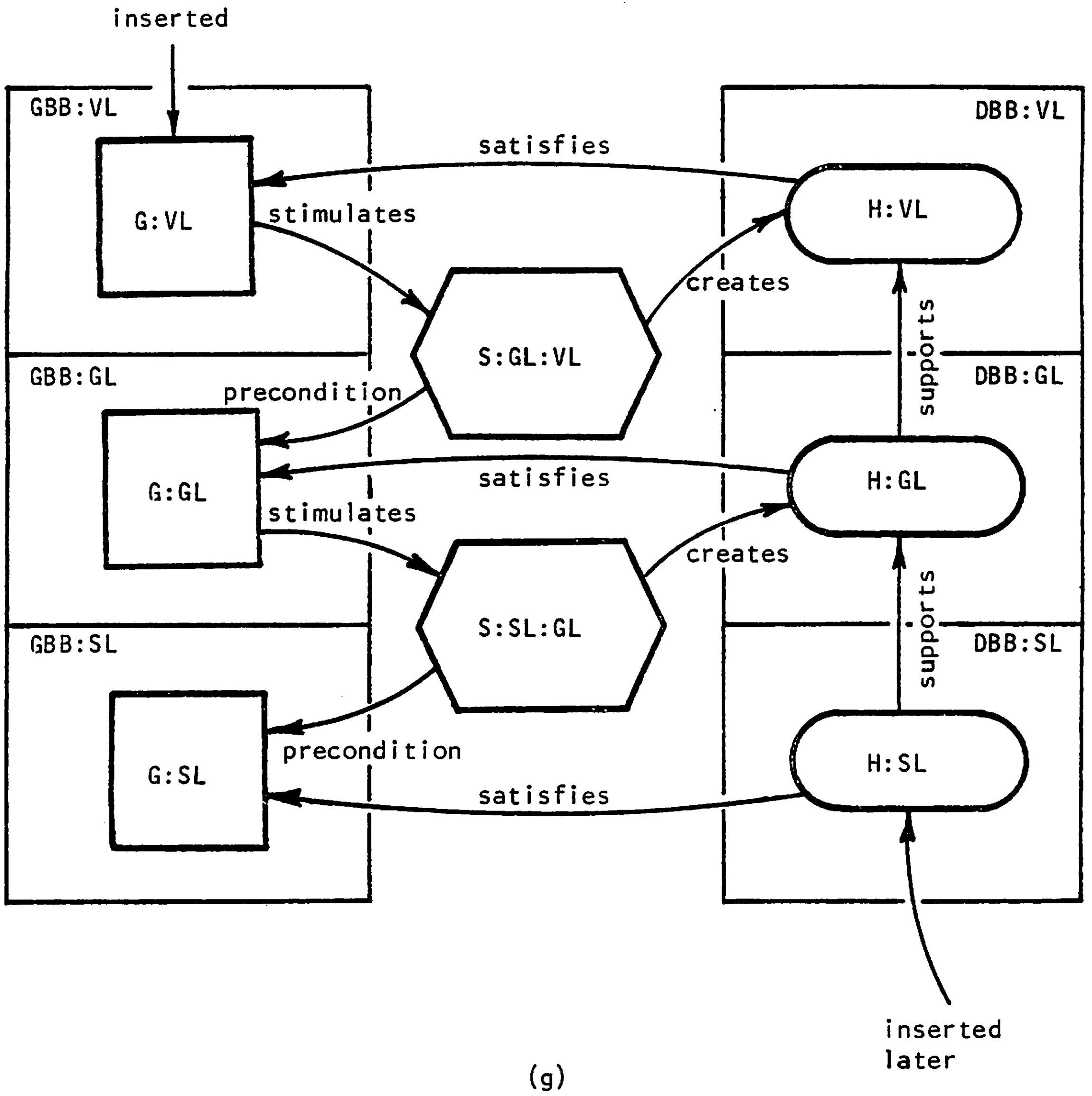


Figure 27: Continued.



(g)

Figure 27: Continued.

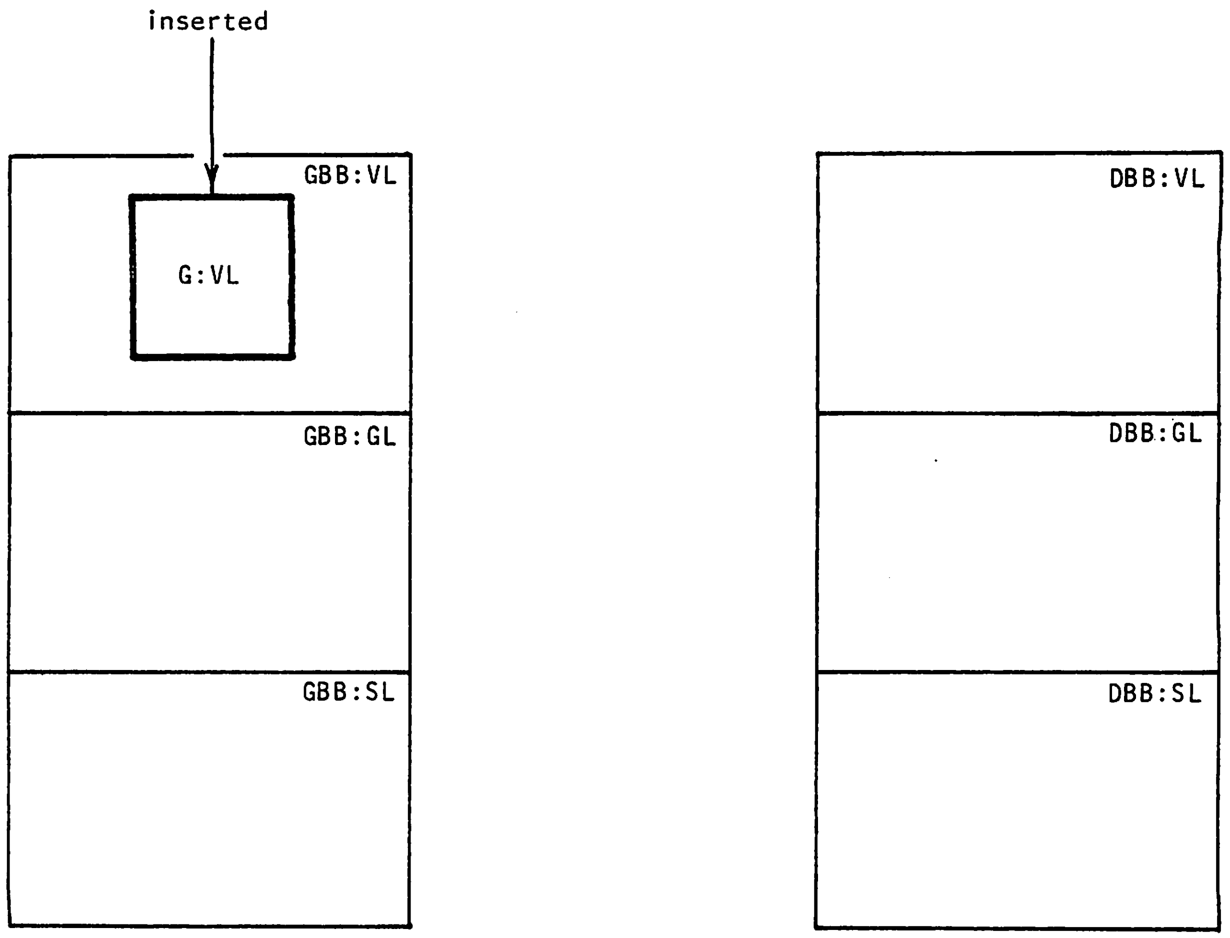
execute (Figure 27d).[5] When H:SL is eventually inserted (Figure 27e), the relationship among goals and knowledge source instantiations is used by the scheduler to execute S:SL:GL and S:GL:VL as a multiple knowledge source sequence to finally achieve G:VL (Figures 27f,g).

Subgoaling.

The planner is also provided with domain knowledge in the form of a goal-to-subgoal mapping for decomposing high-level goals into lower-level ones. This allows it to determine G:GL directly from G:VL and G:SL from G:GL (Figures 28a-c). In this case, S:SL:GL and S:GL:VL are not instantiated by the planner until their respective goals (G:GL and G:VL) have their subgoals satisfied (Figures 28d-h). In fact, the intermediate goal G:GL can be omitted; the planner can directly determine the lowest level subgoals of the high-level goal. Once G:SL is satisfied, the intervening goals can be created bottom-up, using the high-level to low-level subgoal relationship as a guide. In some domains such "level-hopping" could significantly reduce the number of goals generated on the goal blackboard. In the testbed implementation, however, relatively few subgoals are generated and the additional complexity required to support delayed generation of intermediate subgoals was inappropriate.

Subgoaling from high-level predictive goals can be used to greatly reduce the combinatorics associated with using the top-down elaboration of hypotheses as a focusing technique. Top-down elaboration of

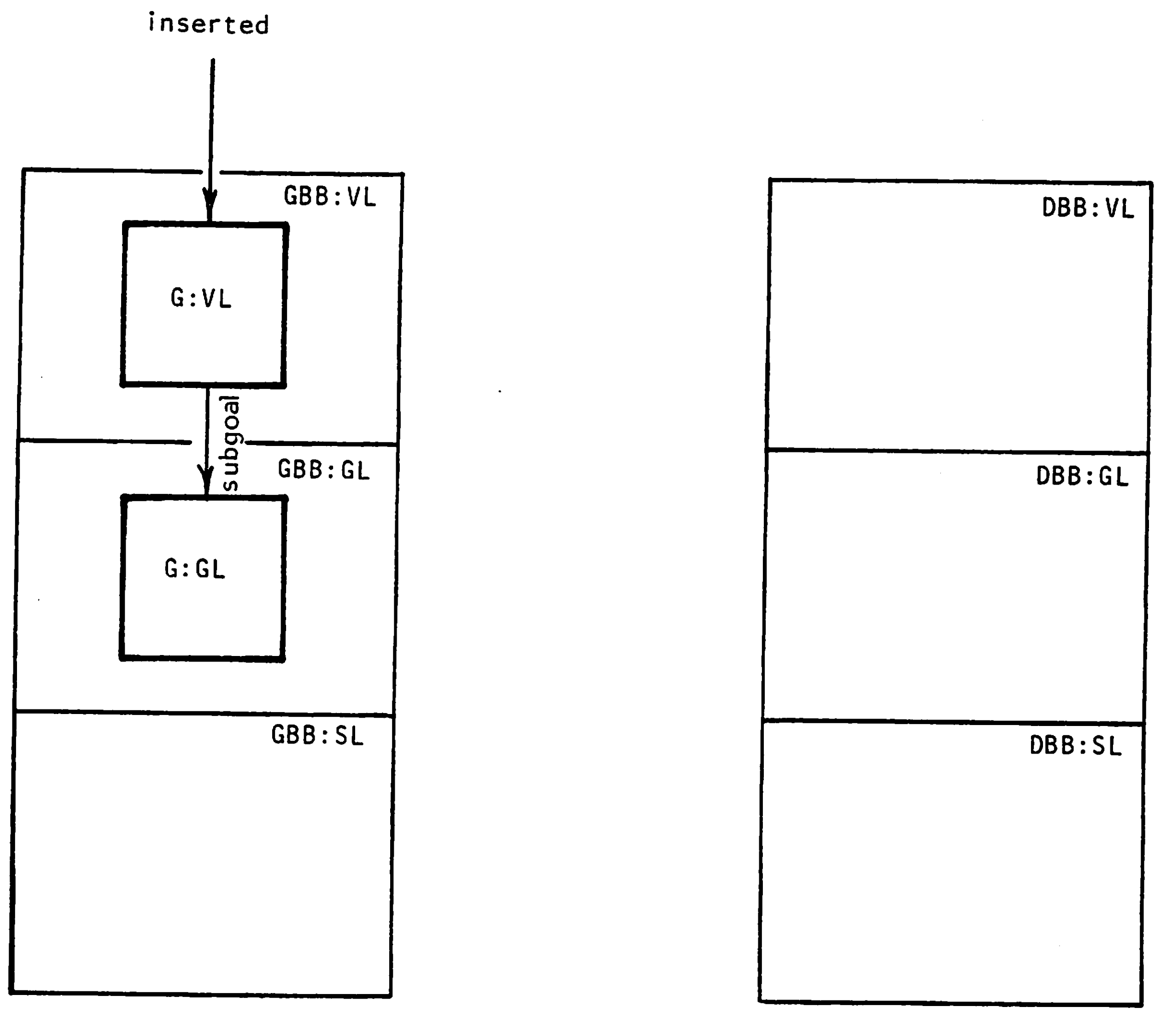
5. If level DBB:SL hypotheses are produced by a controllable sensor, G:SL can be used to request the insertion of hypotheses which satisfy G:SL.



(a)

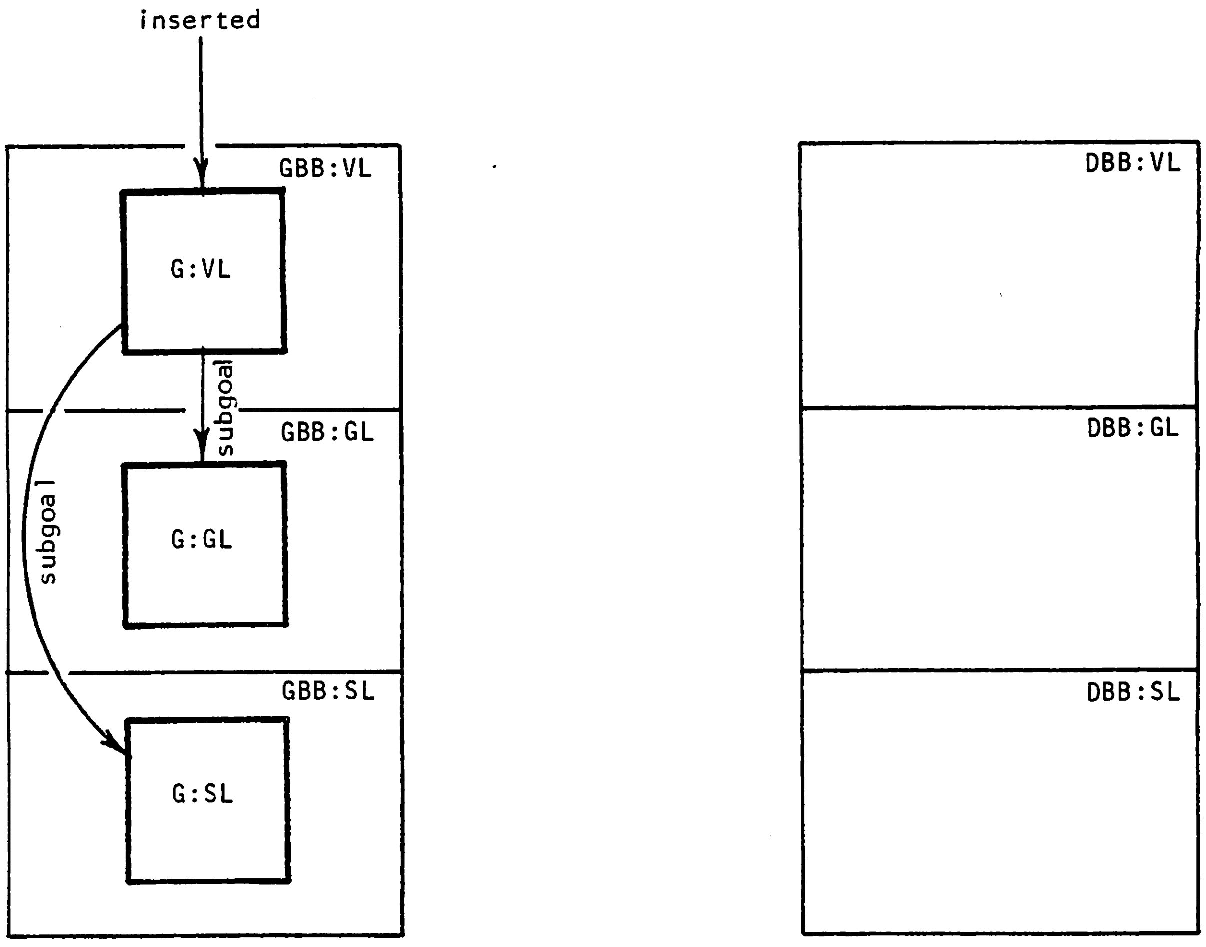
Figure 28: Subgoaling.

The sequence of activities involved in subgoaling a high-level goal.



(b)

Figure 28: Continued.



(c)

Figure 28: Continued.

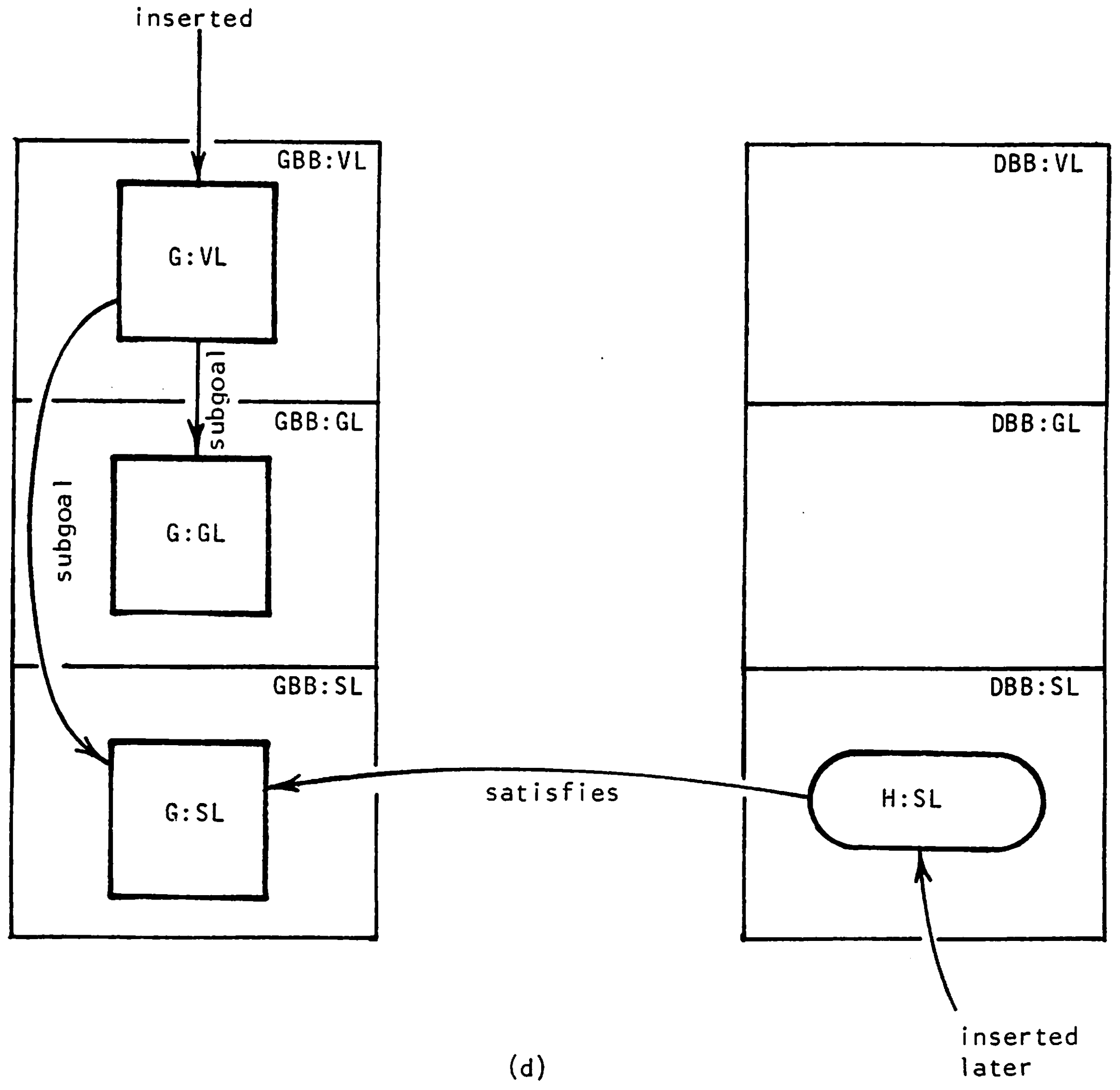


Figure 28: Continued.

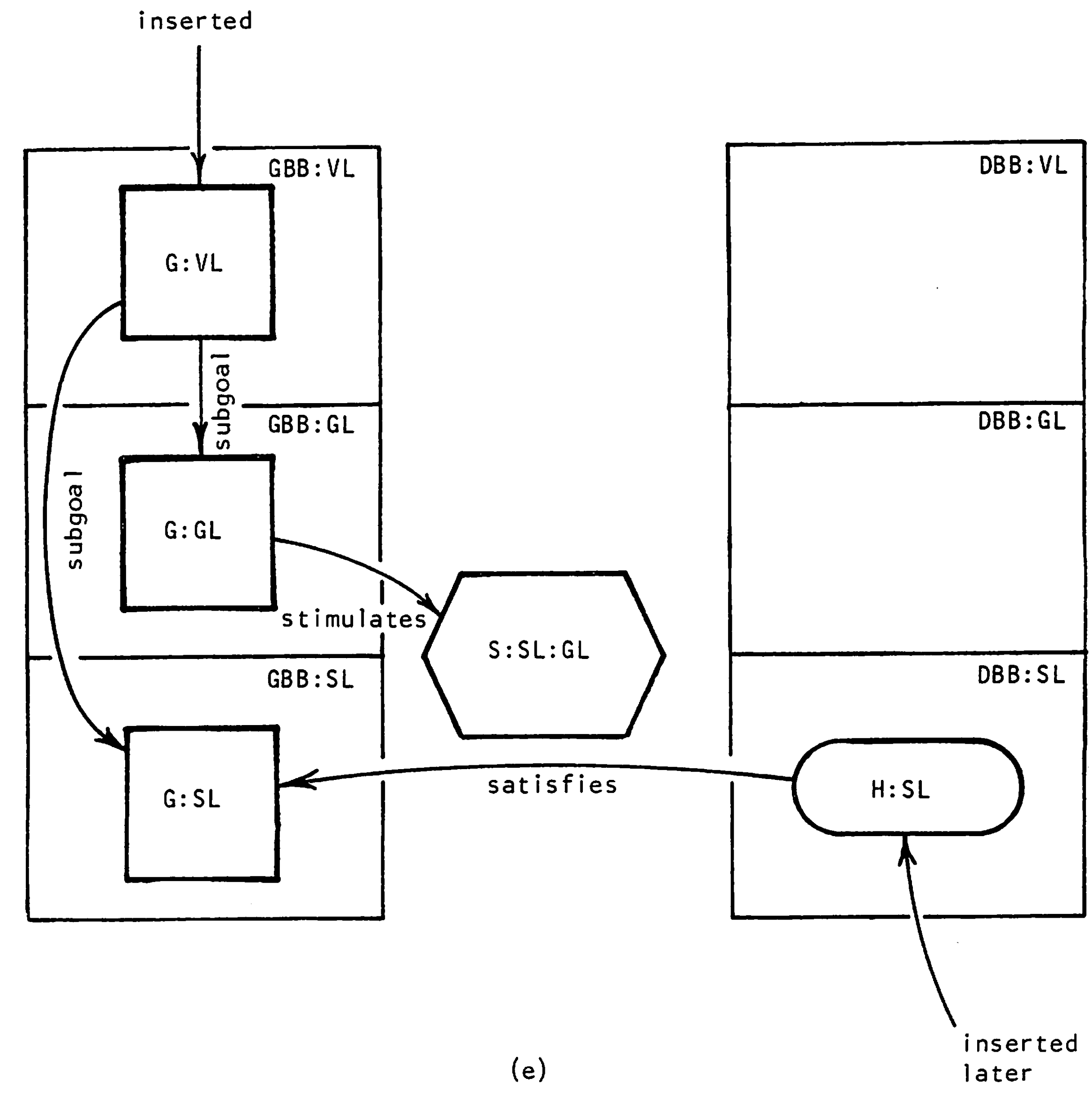


Figure 28: Continued.

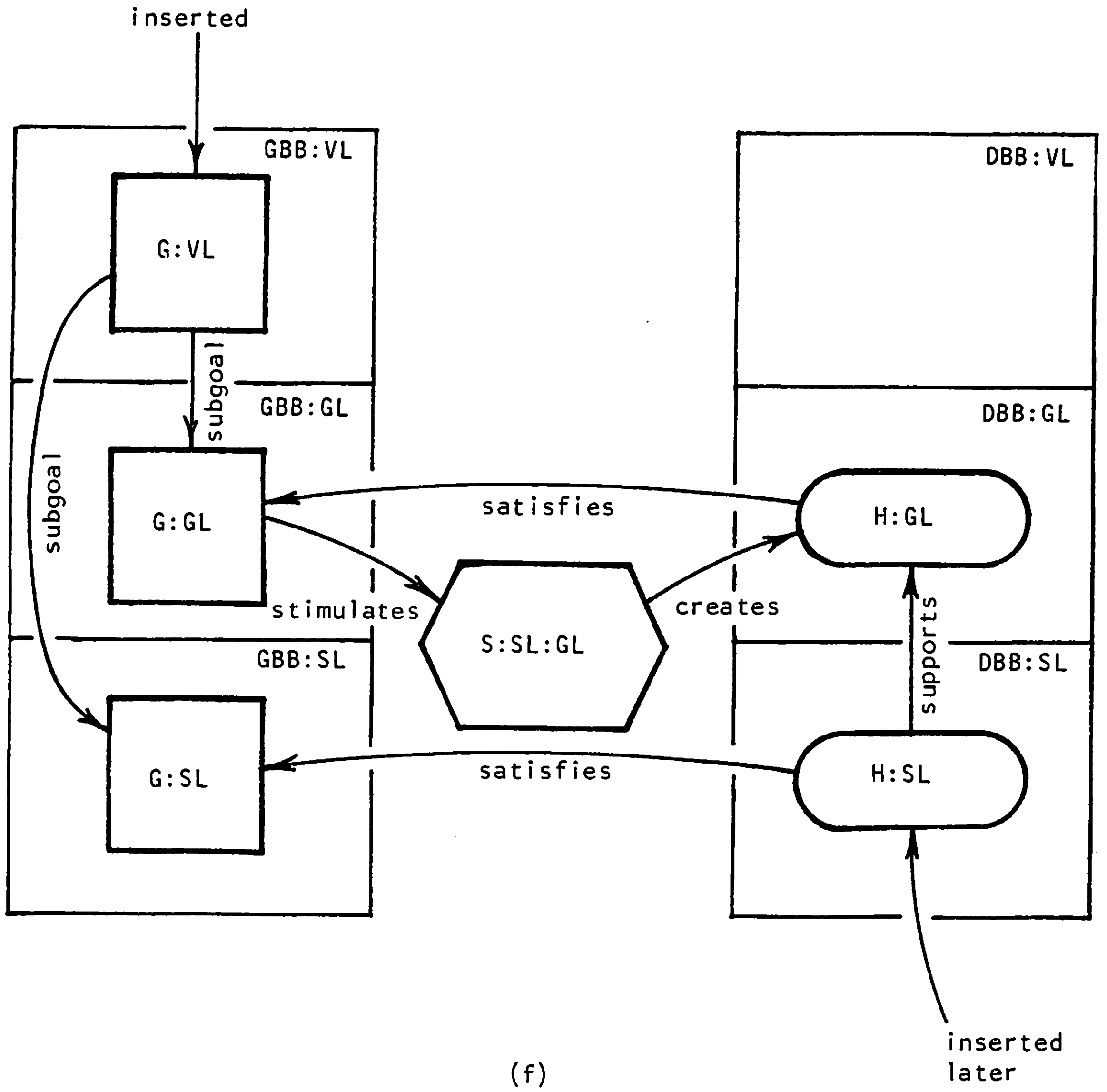


Figure 28: Continued.

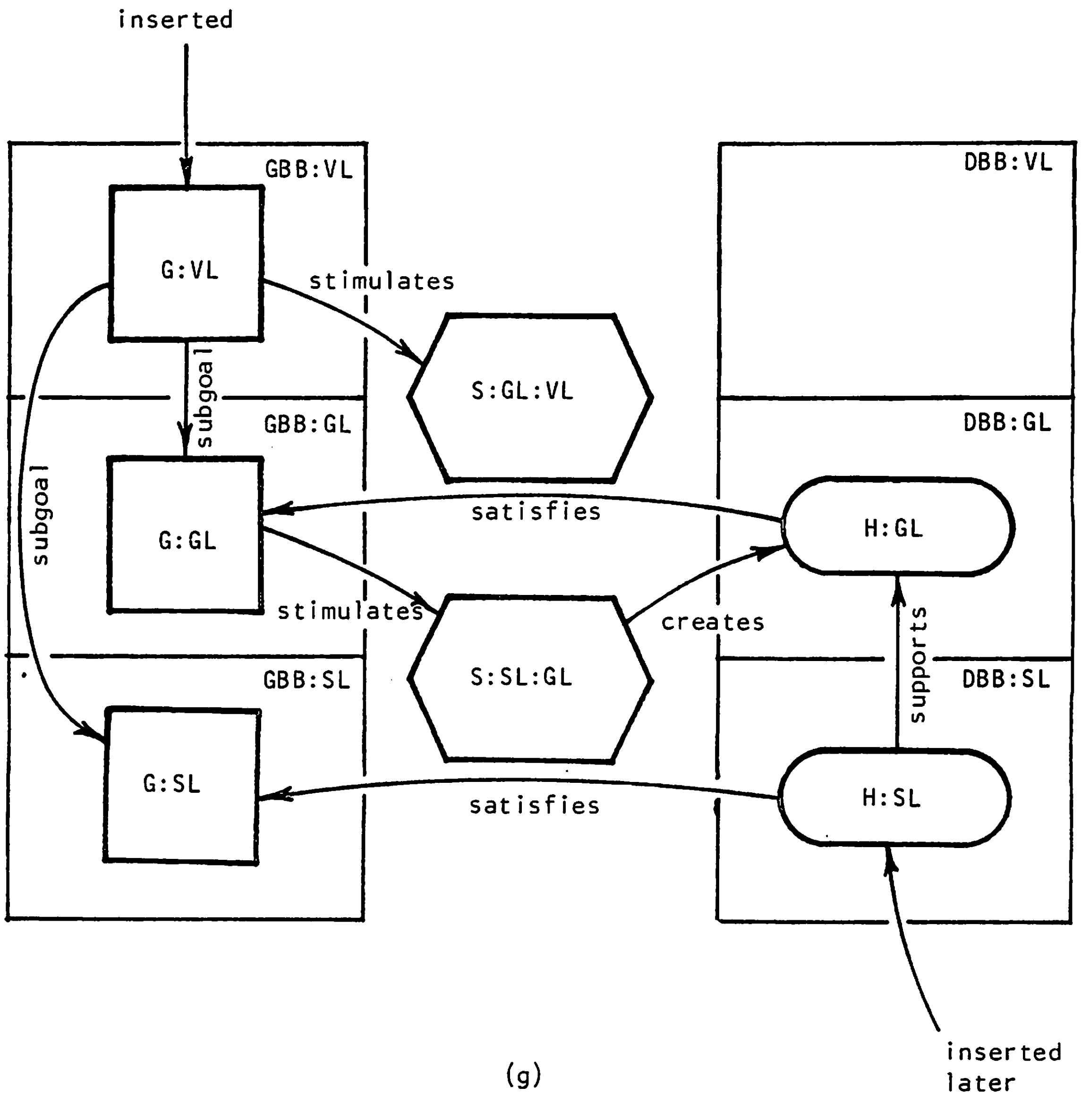


Figure 28: Continued.

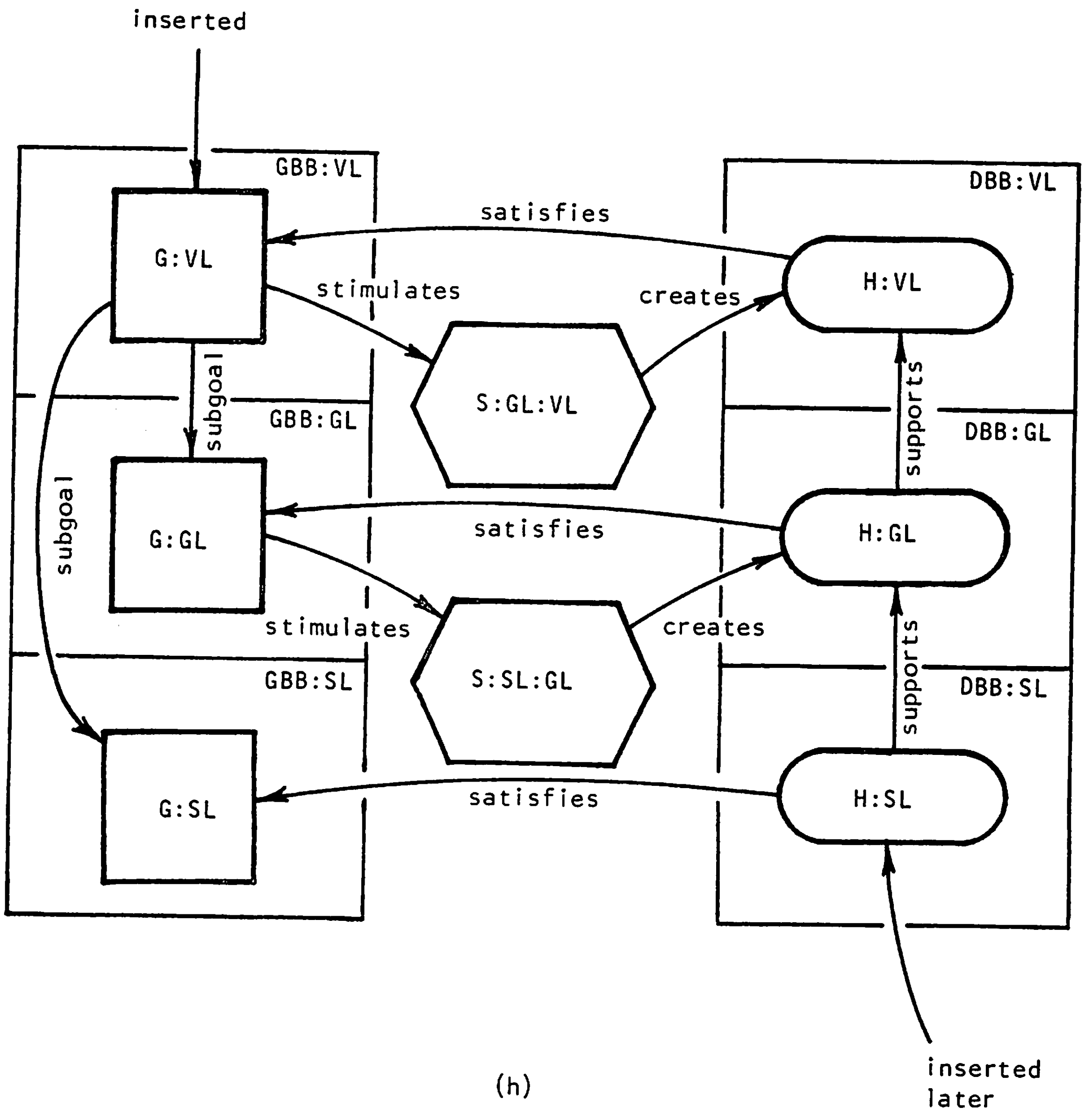


Figure 28: Continued.

hypotheses is generally used for two distinct activities: the generation of the lower-level structure of a hypothesis (to discover details) and the determination of which existing low-level hypotheses should be driven-up to verify or extend a high level hypothesis based on expectations (for focusing). Top-down elaboration of hypotheses is best suited only to the first activity. Subgoaling of predicted goals on the goal blackboard is a more effective way to perform expectation-based focusing (see Section 4.2.7). When hypothesis elaboration is used as a focusing technique, the elaboration process has to be conservative in order to reduce the number of hypotheses generated and to reduce the possibility of generated low-level hypotheses being used as "real data" by knowledge sources in other contexts. Because subgoals are distinct from hypotheses, they can be liberally abstracted (such as supplying a range of values for an attribute) and underspecified (such as supplying a "don't care" attribute). Subgoaling the high-level goal of generating the expectation-based hypothesis (including the possible use of "level-hopping") avoids the combinatorial and context confusion problems associated with the use of top-down hypothesis elaboration for focusing.

4.1.3 Control in the goal-directed Hearsay-II architecture.

The goal-directed approach permits all three of these goal-processing activities to be performed in an opportunistic way by the planner. Highly rated low-level hypotheses can be driven up in a data-directed fashion while high-level goals generated from strong expectations can be subgoaled downward to control low-level synthesis activities. Similarly, processing in low rated areas can be stimulated if a highly rated knowledge source requires the creation of a precondition hypothesis in that area.

Control decisions in the goal-directed architecture can be made or deferred at a number of points in the data blackboard event to knowledge source execution process, based on the availability and reliability of control information. When goals are created by the blackboard monitor, they are assigned an initial priority rating based on the belief of its stimulus hypotheses, the rating of any knowledge source instantiations which require satisfaction of the goal as a precondition, the priority rating of any goals which have the goal as a subgoal, on externally supplied priority (for goals received from other nodes), and as we will see later, on the organizational responsibilities assigned to the node. These ratings are used to focus planning activity to higher rated goals. The blackboard monitor also serves as a "goal filter" by placing only those goals with a priority rating above a specified goal-threshold onto the goal blackboard.

Similarly, the planner can use a second dynamic threshold value to ignore goals placed onto the goal blackboard which are rated below threshold. The distinction between the two is that the planner can eventually plan for goals on the goal blackboard which are currently below the planning-threshold, but goals which are not inserted by the blackboard monitor are lost and must be recreated by the planner from other planning activities.

The goal-directed architecture also provides the scheduler with additional information for selecting which competing knowledge source instantiations to execute. The relationship between alternative knowledge sources which achieve the same goal is explicitly available. Figure 29 shows two highly rated goals, each of which can be satisfied by a moderately expensive knowledge source or a more expensive knowledge source. However, the more expensive knowledge source can satisfy both goals. Due to the lack of this information in the data-directed system, the single-shot scheduler would select the two moderately expensive knowledge sources rather than the shared knowledge source which satisfies both goals for a lower total cost.

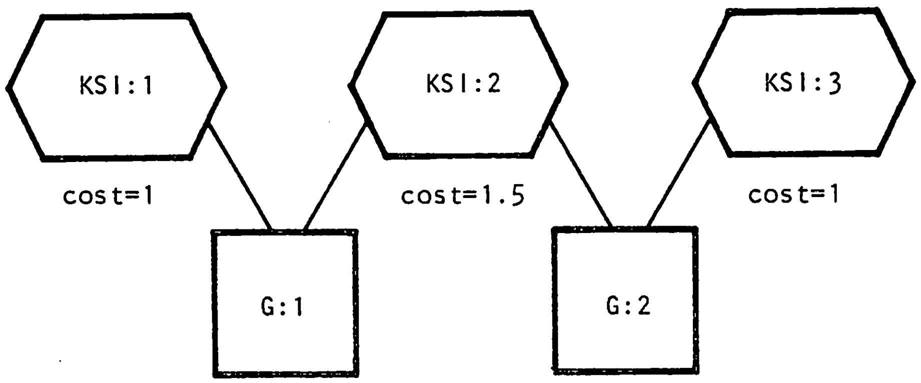


Figure 29: Alternative Knowledge Source Instantiations.

Selecting a more expensive knowledge source that can achieve both goals over two moderately expensive knowledge sources that can each achieve one of the goals. The relationships among knowledge source instantiations and the goals they can achieve allows the scheduler to execute the more expensive knowledge source over the other two for a lower total cost.

4.1.4 Generalizations in the Hearsay-II knowledge source model.

Two generalizations in the model of a knowledge source are also important in the goal-directed approach. In the data-directed system, the knowledge source precondition procedure uses its stimulus hypotheses as an input context for the knowledge source instantiation. In the goal-directed system, the planner can elect to supply a knowledge source with only stimulus hypotheses, only stimulus goals, or both stimulus hypotheses and goals. In the first case, the knowledge source precondition procedure functions as in the data-directed system. In the second case, the knowledge source precondition procedure itself determines the input context in order to best achieve the stimulus goals. In the third case, the stimulus hypotheses are used as the input context and the goals are used as an output filter by the knowledge source; hypotheses which are outside the scope of the goals are not created by the knowledge source.

The second generalization of the knowledge source model is the establishment of bidirectional communication between each knowledge source and the planner. While the precondition process is used by the planner to determine if the major conditions necessary for the knowledge source to achieve the goal are present, a knowledge source may still fail due to the lack of "secondary" preconditions or detailed incompatabilities between its input hypotheses. If the knowledge source reports the nature of the problem back to the planner, the planner can elect to create highly specific precondition goals for the knowledge source and attempt to achieve them or to choose another knowledge source which can satisfy the original goal in a different manner.

4.1.5 Goal attributes.

A number of attributes can be associated with each goal. The attributes listed in this section are potentially useful in many application domains. Undoubtedly the most important goal attribute is the satisfaction specification attribute. It is a declaration of the desired state of hypotheses on the data blackboard that the goal represents and serves as the basic means of communication between the blackboard monitor, the planner, precondition processes, and knowledge sources. In the testbed it is represented by three values: a blackboard level, a set of data blackboard times and their associated regions, and a disjunctive set of desired event classes. A second goal attribute is the minimum satisfaction specification, the minimum conditions under which the goal is considered satisfied. The satisfaction specification states what conditions are desired, and the minimum satisfaction specification indicates when the goal is sufficiently satisfied to allow processing to proceed to knowledge source instantiations that require minimal achievement of the goal. In the testbed this is represented by a minimum belief threshold on the satisfaction condition.

Another important goal attribute is the rating of the goal's importance, which is used to direct the planner and to influence the rating of knowledge source instantiations. In some domains the importance of goals may sharply decline due to time constraints on the resulting hypotheses. In this case it is useful to associate a "time-out" condition with the rating at which time it is recalculated. An estimate of the cost of achieving the goal is another useful goal attribute. Cost estimates can be used by the planner to choose between alternative strategies based upon the resources which must be expended in each. Similarly, an estimate of the probability of satisfying the goal can be used by the planner to choose a strategy that has the greatest chance of success.

Goals also contain a number of attributes linking them to other goals, knowledge source instantiations, and hypotheses. These link attributes include: the goal's subgoals, goals that include it as a subgoal (supergoals), goals that are more abstract, goals that are more specialized, goals that overlap, the hypotheses that stimulated the creation of the goal, the hypotheses that can satisfy the goal (at least in part), knowledge source instantiations that can potentially satisfy the goal if executed, and knowledge source instantiations that require the goal to be achieved as a precondition.

Goals can be used to request a number of different types of knowledge source processing based on the details of their satisfaction specification attribute. Specific hypothesis goals request a change to be made to a particular (named) hypothesis. "Increase the belief of hypothesis H:SL:057" is a specific hypothesis goal. Generic hypothesis goals request the creation or modification of a single (unnamed) hypothesis which satisfies a set of specified attributes. "Create a hypothesis on level S:GL at time 3 with belief greater than 5000" is a generic hypothesis goal. Area goals request the establishment of relationships between hypotheses in a specified area of the data blackboard. "Create at least 5 hypotheses with belief greater than H:GL:184 on level S:GL at time 3" is an example of an area goal.

Goals can also be characterized according to their duration. Single shot goals remain active (the planner attempts their satisfaction) only until they are first satisfied. Single shot goals may be reactivated by the blackboard monitor or by the insertion of external goals, but are distinct from continuous goals which always remain active. "Send all created hypothesis on level DBB:VL which have belief greater than 8000 to node 2" is an example of a continuous goal.

Goals such as "create only those hypotheses with belief greater than 6000 from stimulus hypothesis H:SL:057" naturally implement threshold control of knowledge source activity. Threshold control reduces the number of hypothesis inserted onto the data blackboard by treating knowledge sources as generator functions which only create their highest rated output hypotheses and can be reinvoked if lower rated hypotheses are desired.

4.1.6 Goal processing.

In addition to the use of data blackboard events to create new goals, these events must also be checked to determine if they can satisfy existing goals on the blackboard. This checking can be performed by the blackboard monitor, the planner, or a combination of both, depending on characteristics of the task domain. If the checking requires only simple, syntactic matching of the attributes of hypotheses with those of goals, then it should be implemented as part of the blackboard monitor. (This is the approach taken in the testbed). However, if extensive task domain knowledge and processing resources are required for matching, then the planning module is the appropriate place to perform the checking. The philosophy is to keep the blackboard monitor a simple table-driven procedure with low processing overhead. A combined approach, in which a quick syntactic check is first performed by the blackboard monitor, can also be used. If this check results in only partial matches, then the event and partially matched goals are passed on to the planner for more extensive analysis. In all of these cases, the parallel structure of the data and goal blackboards facilitates the effective implementation of event and goal matching.

Goal merging, which involves recognizing that two goals can be satisfied by the same conditions, can be similarly implemented in the blackboard monitor, the planner, or a combination of both. Again the choice is based on the complexity of the operation required by the task domain.

Additional goal-processing operations also need to be implemented. These operations involve checking newly created goals for their relationships with existing goals, including the relationships of: goal/subgoal, goal/overlapping-goal, goal/precondition-goal, goal/abstract-goal, and goal/specialized-goal. The need for determining these relationships depends on the task domain and planning strategies used in the system and, due to their complexity, most seem best implemented in the planning module.

4.2 Implementation of the Goal-Directed Hearsay-II Architecture in the Testbed

The complete goal-directed Hearsay-II architecture with both bottom-up synthesis and subgoaling capabilities has been implemented in the distributed vehicle monitoring testbed. Precondition-action backward chaining has not been implemented (although there are situations in the testbed where it would be beneficial). A rudimentary planner has also been implemented. A key aspect of the control framework implemented in the testbed is the use of a nonprocedural and dynamically variable specification of the behaviors of each local node's planner, its scheduler, and its communication knowledge sources. These data structures are used to implement particular network configurations and coordination policies and, as will be discussed at the end of this chapter, provide the interface between the activity decisions of the local node and organizational structuring decisions.

The following sections describe the testbed implementation.

4.2.1 Additional knowledge sources.

Additional communication knowledge sources used in the goal-directed testbed architecture are listed in Table 4. A goal send knowledge source transmits goals created on the goal blackboard to other nodes based on the level, time frames, event classes, regions, and rating of the goal. Goal send transmits goals based on organizational criteria -- whether or not the node is to attempt to achieve the goal locally. A goal help knowledge source is used to transmit any goals that the node's planner has determined cannot be satisfied locally

ADDITIONAL KNOWLEDGE SOURCES

Goal Transmission:	Goal Reception:
GOAL-SEND:SL:SL	GOAL-RECEIVE:SL:SL
GOAL-SEND:ST:ST	GOAL-RECEIVE:ST:ST
GOAL-SEND:GL:GL	GOAL-RECEIVE:GL:GL
GOAL-SEND:GT:GT	GOAL-RECEIVE:GT:GT
GOAL-SEND:VL:VL	GOAL-RECEIVE:VL:VL
GOAL-SEND:VT:VT	GOAL-RECEIVE:VT:VT
GOAL-SEND:PL:PL	GOAL-RECEIVE:PL:PL
GOAL-SEND:PT:PT	GOAL-RECEIVE:PT:PT

Assistance Goal Transmission:	Satisfying Hypotheses Transmission:
GOAL-HELP:SL:SL	HYP-REPLY:SL:SL
GOAL-HELP:ST:ST	HYP-REPLY:ST:ST
GOAL-HELP:GL:GL	HYP-REPLY:GL:GL
GOAL-HELP:GT:GT	HYP-REPLY:GT:GT
GOAL-HELP:VL:VL	HYP-REPLY:VL:VL
GOAL-HELP:VT:VT	HYP-REPLY:VT:VT
GOAL-HELP:PL:PL	HYP-REPLY:PL:PL
GOAL-HELP:PT:PT	HYP-REPLY:PT:PT

Table 4: Additional Distributed Vehicle Monitoring Testbed Knowledge Sources.

The additional testbed knowledge sources used for communicating goals and replies to goals. Each knowledge source can be individually selected and weighted at each testbed node.

The name each knowledge source has the form:

type : input-level(s) : output-level.

(possibly after executing a number of local problem solving knowledge sources). This separation allows a clear distinction in the criteria for sending goals to other nodes for organizational reasons and for asking for help with goals that cannot be achieved locally. Both knowledge sources use a simple model of the goals that have been seen by each node and of the availability of the communication channel to decide whether or not to send a particular goal.

A goal receive knowledge source places goals received from other nodes onto the node's goal blackboard. Incoming goals are filtered according to the level, time frames, event classes, regions, and rating of the received goal to ensure that the node is truly interested in receiving goals of that type. (Specification of the interest areas of a node is discussed in the next section.) Goal receive knowledge sources also use a simple model of the credibility of the sending node to possibly lower the rating of the received goal before it is placed on the blackboard.

A goal reply knowledge source transmits hypotheses created on the blackboard that satisfy a goal that specifies a reply to another node when satisfied.

4.2.2 Interest areas.

The activities of the local node planner, scheduler, and communication knowledge sources are influenced by data structures called _interest areas_. There are six sets of interest areas for each node in the testbed:

local processing interest areas	— influence the local problem solving activities in the node by modifying the priority ratings of goals and knowledge source instantiations and the behavior of the node's planner and scheduler;
hypothesis transmission interest areas	— influence the behavior of HYP-SEND knowledge sources in the node;
hypothesis reception interest areas	— influence the behavior of HYP-RECEIVE knowledge sources in the node;
goal transmission interest areas	— influence the behavior of GOAL-SEND knowledge sources in the node;
goal help transmission interest areas	— influence the behavior of GOAL-HELP knowledge sources in the node;
goal reception interest areas	— influence the behavior of GOAL-RECEIVE knowledge sources in the node.

Each interest area is a list of areas of the data or goal blackboard specified by a set of blackboard levels, a set of event classes, and a set of time frame, spatial region lists. Associated with each interest area are one or more parameters that modify the behavior of the node (see Table 5).

Each local processing interest area has a single parameter associated with it: a weight specifying the importance of performing local processing within the interest area. Transmission interest areas (hypothesis transmission, goal transmission, and goal help transmission) are specified for one or more lists of nodes that are to receive information from the node. Similarly, reception interest areas (hypothesis reception and goal reception) are specified for lists of

Table 5: Interest Area Parameters.

The parameters associated with a node's local processing interest areas, transmission interest areas (hypothesis transmission, goal transmission, and goal help transmission), and reception interest areas (hypothesis reception and goal reception) are used to influence the behavior of the node.

Only hypothesis reception interest areas have the focusing-weight parameter.

Local Processing Interest Areas:

```
                      ((interest-area importance)
                       (interest-area importance)
                                  . . .
                       (interest-area importance))
```

Transmission Interest Areas:

```
          ((node-list ((interest-area threshold importance)
                       (interest-area threshold importance)
                                            . . .
                       (interest-area threshold importance))
          ((node-list ((interest-area threshold importance)
                       (interest-area threshold importance)
                       (interest-area threshold importance)
                                            . . .
                       (interest-area threshold importance))
                            . . .
          ((node-list ((interest-area threshold importance)
                       (interest-area threshold importance)
                                            . . .
                       (interest-area threshold importance)))
```

Reception Interest Areas:

```
    ((node-list ((interest-area threshold importance credibility
                 [focusing-weight])
                 (interest-area threshold importance credibility
                 [focusing-weight])
                                              . . .
                 (interest-area threshold importance credibility
                 [focusing-weight]))
    ((node-list ((interest-area threshold importance credibility
                 [focusing-weight])
                 (interest-area threshold importance credibility
                 [focusing-weight])
                                              . . .
                 (interest-area threshold importance credibility
                 [focusing-weight]))
                          . . .
    ((node-list ((interest-area threshold importance credibility
                 [focusing-weight])
                 (interest-area threshold importance credibility
                 [focusing-weight])
                                              . . .
                 (interest-area threshold importance credibility
                 [focusing-weight])))
```

Table 5: Interest Area Parameters.

nodes that are to transmit information to the node. Each transmission interest area has a weight specifying the importance of transmitting hypotheses or goals from that area (to nodes specified in the node-list) and a threshold value specifying the minimum hypothesis belief or goal rating needed to transmit from that area. Each reception interest area has a weight specifying the importance of receiving a hypothesis or goal in that area (from a node specified in the node-list), a minimum hypothesis belief or goal rating needed for the hypothesis or goal to be accepted, and a credibility weight. The credibility weight parameter is used to change the belief of received hypotheses or the rating of received goals. A node can reduce the effect of accepting messages from a node by lowering the belief or rating of messages received from that node. Each hypothesis reception interest area also has a focusing weight parameter that is used to determine how heavily received hypotheses are used in making local problem solving focusing decisions (see Section 4.2.6).

As we will see in Section 4.4, interest areas specifications provide the interface between the activity decisions made by a node and organizational structuring decisions. A node's organizational responsibilities are changed by modifying its interest areas. These changes can be made as a part of network initialization as well as dynamically, during the course of network problem solving.

4.2.3 Major goal attributes.

In this section the major attributes of goals in the testbed implementation are described. (The complete list of goal attributes is given in Appendix B.)

As mentioned earlier, three attributes are used in the testbed to specify the desired state of hypotheses on the data blackboard: the goal's level, active-time-region-list, and event-classes attributes. The level attribute is simply one of the eight blackboard levels. The active-time-region-list attribute[6] is a list of time frame, spatial region pairs where each blackboard region is a rectangular area specified by the list (x_min y_min x_max y_max). The event-classes attribute is a list of event class numbers. The pairs in the active-time-region-list attribute and the event classes in the event-classes attribute represent disjunctive goal states on the data blackboard. The importance of achieving a goal is indicated by its rating attribute

The relationships between a goal and hypotheses, knowledge source instantiations, and other goals are indicated by various link attributes. The goal's stimulus-hyps attribute connects the goal with one or more hypotheses that, when created, stimulated the creation of the goal. The goal's stimulated-ksis attribute connects the goal with those knowledge source instantiations that are attempting to achieve it. Hypotheses that satisfy the goal (at least in part) are linked as satisfying-hyps of the goal. Goals whose level,

6. Goals also have an inactive-time-region-list attribute used to further specify a goal. It is described below.

active-time-region-list, and event-classes attributes overlap the goal's attributes are linked using the overlapping-goals attribute. The subgoals attribute points to goals that are subgoals of the goal. The supergoal attribute indicates the reverse relationship.

4.2.4 The testbed blackboard monitor and blackboard events.

The testbed blackboard monitor recognizes seven types of blackboard events. They are:

hypothesis creation	-- creation of a new hypothesis on the data blackboard;
hypothesis modification	-- modification of the belief of an existing hypothesis on the data blackboard;
hypothesis reception	-- reception of one or more hypotheses in the node's communication buffer (the hypotheses are inserted onto the data blackboard by a HYP-RECEIVE knowledge source);
goal creation	-- creation of a new goal on the goal blackboard;
goal modification	-- modification of the rating or stimulus hypotheses of an existing goal on the goal blackboard;
goal reception	-- reception of one or more goals in the node's communication buffer (the goals are inserted onto the goal blackboard by a GOAL-RECEIVE knowledge source);
goal satisfaction	-- creation or modification of one or more hypotheses of sufficient belief in the data blackboard area specified by the goal.[7]

A seventh event, quiescence, is also used in the testbed. Not a true blackboard event, a quiescence event is signalled when there are no executable knowledge source instantiations above a specified knowledge

7. See Section 4.3.3 for a discussion of the problems associated with determining goal satisfaction.

source instantiation execution threshold rating at the node.

The blackboard monitor behaves differently for each type of blackboard event. We look first at hypothesis creation.

Hypothesis creation events.

If the created hypothesis is a location hypothesis, the blackboard monitor uses the event-to-goals mapping to determine the attributes of one or more goals to be be created at the next higher location blackboard level (if one exists). These goals represent the desire to create an output hypothesis with attributes derived from the newly created hypothesis at that next higher location level. Since the location synthesis knowledge sources allow a shifting of one unit in the X and Y dimensions, the goals need to cover a three unit square region centered around the newly created hypothesis. Similarly, since the frequency class can be shifted up or down by one class on the signal location and group location levels, signal location and group location goals must cover a three frequency event class range centered around the event class specified in the grammar. For example, suppose a node using the grammar presented in Chapter II (reproduced as Figure 30) creates a signal location hypothesis with a time-location-list attribute of ((1 (5 15))) and an event-class attribute of 17. The blackboard monitor would create a group location goal with an active-time-region-list attribute of ((1 (4 14 6 16))) and an event-classes attribute of (5 6 7) and a second group location goal with the same active-time-region-list attribute and an event class attribute of (9 10 11).

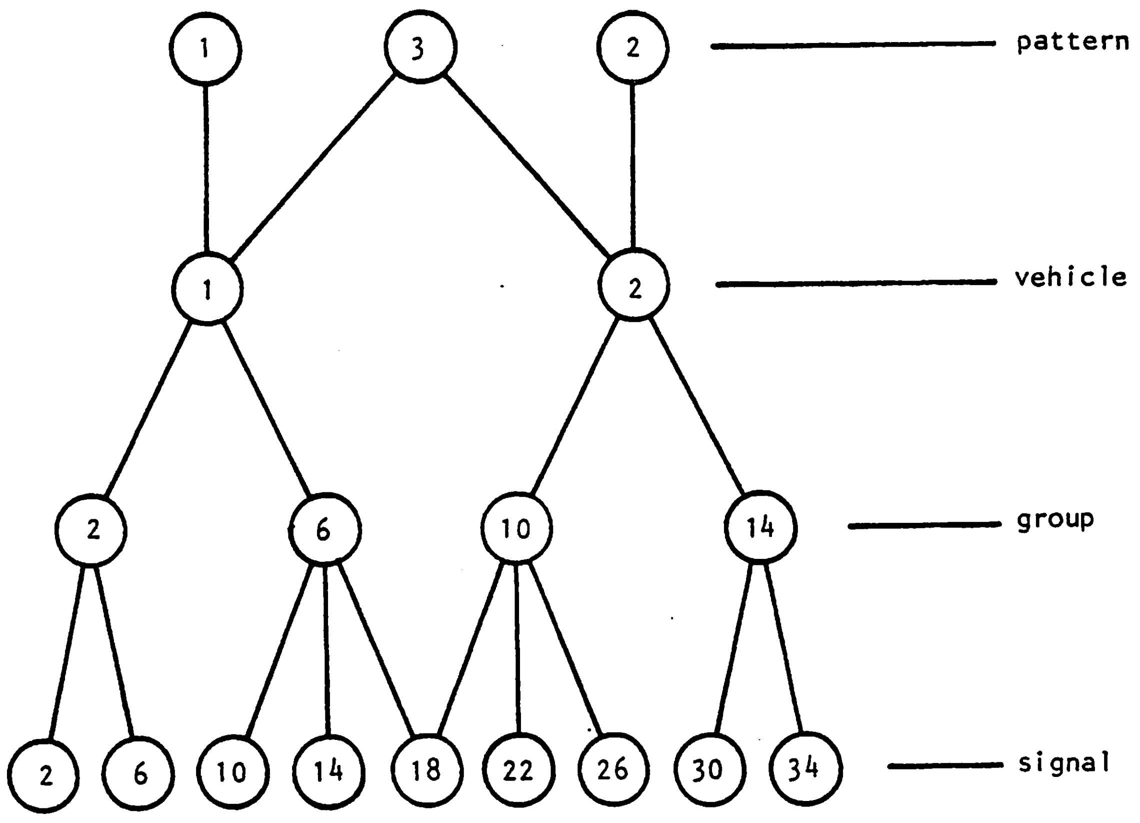


Figure 30: The Simple Testbed Grammar.

The illustrative grammar used in the examples presented in this section.

For signal and group location hypothesis creation events only 1 higher location level goal is created. However, vehicle location creation events are somewhat more complicated because the grammar specifies a spatial displacement of pattern locations with respect to vehicle locations. In this case multiple goals may be needed to represent the possible output hypotheses that can be synthesized at the pattern location level from the vehicle location hypothesis. (The blackboard monitor attempts to coalesce goals as much as possible.)

For each location hypothesis creation event the blackboard monitor also creates two track formation goals at the location hypothesis's corresponding track level: one to form a track from the created hypothesis using a hypothesis in the previous time frame and the other to form a track using a hypothesis in the following time frame. Because the velocity of the vehicle cannot be estimated from a single location, each track formation goal is a region twice the maximum vehicle velocity square, centered at the position of the location hypothesis. For a vehicle location or pattern location hypothesis, the event-classes attribute of each track formation goal contains only the event class of the location hypothesis. For a signal location or group location hypothesis, the event-classes attribute of each track formation goal must contain one shifted frequency event class in each direction from the frequency of the location hypothesis. Because the track goals are at the same level of abstraction in the data hierarchy as the created hypothesis, no vehicle-to-pattern conversion, (and therefore no spatial displacement of goal regions) is involved.

For example, a node using the grammar of Figure 30 and a maximum vehicle velocity of 3 units per time frame would generate one signal track formation goal with an active-time-region-list attribute of ((0 (2 12 8 18))) and an event-classes attribute of (17 18 19) and the other signal track formation goal with an an active-time-region-list attribute of ((2 (2 12 8 18))) and an event-classes attribute of (17 18 19) from the signal location hypothesis with a time-location-list attribute of ((1 (5 15))) and an event-class attribute of 17.

If the output hypothesis is a track hypothesis, the blackboard monitor uses the event-to-goals mapping to determine the attributes of one or more goals to be created at the next higher track blackboard level (if one exists). Each time frame, spatial location pair in the track hypothesis is converted into a time frame, spatial region pair for the active-time-region-list attribute of these goals. This conversion is identical to the conversion of the single time frame, spatial location pair of a location hypothesis. For example, a node using the grammar of Figure 30 creating a signal track hypothesis with a time-location-list attribute of ((1 (5 15)) (2 (8 16))) and an event-class attribute of 17 would create a group track goal with an active-time-region-list attribute of ((1 (4 14 6 16)) (2 (7 15 9 17))) and an event-classes attribute of (5 6 7) and a second group track goal with the same active-time-region-list attribute and an event class attribute of (9 10 11). As with pattern location goals, pattern track goals can require a spatial displacement of the spatial-region from the spatial-location of the newly created vehicle track hypothesis.

For each track hypothesis creation event the blackboard monitor also creates two track extension goals at the same blackboard level: one specifying an extension of the track forward in time and one specifying an extension backward in time. With track extension goals it is possible to compute the velocity of the vehicle at the end of the track (or the inverse of the velocity at the beginning of the track if extending backward). The extension region is formed by extrapolating this velocity one time-frame and creating a region two times the maximum vehicle acceleration square, centered around the extrapolated position.

The event-classes attribute contains the event class number of the newly created track hypothesis (and plus or minus one frequency class for signal track and group track goals due to possible frequency shifting). The extension-direction attribute of the goal is also set to forward or backward, as appropriate.

For example, the forward extension track goal for the above signal track hypothesis example would have an active-time-region-list attribute of ((3 (9 15 13 17))) and an event-classes attribute of (17 18 19) if the maximum vehicle acceleration is 2 units per time frame.

To differentiate this goal from another forward track extension goal that might happen to have the same active-time-region-list attribute but a completely different track to be extended (such as two vehicles crossing at right angles), an inactive-time-region-list attribute is also formed. The <u>inactive-time-region-list</u> attribute specifies information about the track that stimulated the creation of the goal. For the above example, the inactive-time-region-list attribute would be ((1 (5 15 5 15)) (2 (8 16 8 16))).

While it may seem reasonable to create track extension goals that explicitly name the track hypothesis to be extended (a specific hypothesis goal), the inactive-time-region-list attribute allows the blackboard monitor some latitude in creating a single (generic hypothesis) goal to extend a number of "similar" track hypotheses. For example, by including only the time frame, spatial region pairs nearest the extension end of the track, hypotheses that differ far from the extension end but which are identical near the extension can be included in a single track extension goal. Similarly, enlarging the region size in the inactive-time-region-list attribute allows track hypotheses that are close together to be included in a single goal. Finally, goals with an inactive-time-region-list can be communicated to another node without sending its stimulus track hypotheses.

The blackboard monitor does not immediately create goals on the blackboard from hypothesis creation events. Instead, it first looks to see if the attributes of the new goal are similar to those of an existing goal. If they are, the attributes of the similar goal are changed and a goal modification event is signalled. If they are not, a new goal is created.

In addition to creating or modifying goals from hypothesis creation events, the blackboard monitor also must check to see if the newly created hypothesis satisfies any existing goals on the goal blackboard. If it does the hypothesis is linked as a satisfying hypothesis to those goals and a goal satisfaction blackboard event is signalled. If the created hypothesis is at a level at which the node is transmitting hypotheses to another node, the precondition procedure for HYP-SEND is executed. If it indicates that the hypothesis should indeed be transmitted, a HYP-SEND knowledge source is instantiated.

Hypothesis modification events.

If the belief of a hypothesis is modified, the blackboard monitor must recalculate the ratings of the goals that were originally stimulated by the hypothesis and, in turn, notify the scheduler to rerate any pending knowledge source instantiations associated with those goals. An increased hypothesis belief may also be sufficient to satisfy one or more goals, so the blackboard monitor must check for this possibility. In addition, a raised hypothesis belief may cause the hypothesis to be above the transmission belief threshold for a node or to be raised enough to warrant retransmission of the hypothesis, so the blackboard monitor must execute the precondition procedure for HYP-SEND. If it indicates that the hypothesis should indeed be transmitted, a HYP-SEND knowledge source is instantiated.

Finally, the new belief value of the hypothesis may cause the beliefs of hypotheses created by knowledge sources that used the modified hypothesis in their calculations. In the testbed implementation, propagation of modified belief values is not a scheduled activity. Instead, the blackboard monitor is responsible for propagating changes in hypothesis belief values that are above a specified threshold value. Changes below the .threshold value are assumed to have such a small effect on the belief of the eventual solution that the propagation cost is not justified. The relative contribution of each input hypothesis is recorded in the testbed, allowing the blackboard monitor to change each output hypothesis accordingly -- without reexecuting the knowledge source. Propagation of modified hypothesis beliefs recursively signals hypothesis modification events.

Hypothesis reception events.

If one or more hypotheses are received at a node while a knowledge source is executing, a hypothesis reception event is signalled when that knowledge source completes. The blackboard monitor executes a

HYP-RECEIVE precondition procedure to determine if the node is truly interested in any of the hypotheses. If so, a HYP-RECEIVE knowledge source is instantiated to potentially modify the belief of each hypothesis and to place them on the data blackboard (at which time a hypothesis creation event or a hypothesis modification event is signalled by the blackboard monitor).

Goal creation events.

When a goal is created on the goal blackboard, the blackboard monitor calculates a priority rating for the goal (described in Section 4.2.6) and determines if any existing goals overlap the new goal. Two goals overlap if they are on the same blackboard level, have at least one event class in common, and have at least one overlapping region (in an identical time frame) in their active-time-region-list attribute. The overlapping-goals attribute is used to link these goals with the newly created goal. The blackboard monitor then checks to see if the goal is already satisfied by existing hypotheses. If it is not already satisfied the blackboard monitor notifies the planner that it should attempt to satisfy the goal. The actions of the planner are detailed in Section 4.2.5.

If the created goal is at a level at which the node is immediately transmitting goals to another node, the precondition procedure for GOAL-SEND is executed. If it indicates that the hypothesis should indeed be transmitted, a GOAL-SEND knowledge source is instantiated.

Goal modification events.

If the rating of a goal is modified, the blackboard monitor notifies the scheduler to rerate all pending knowledge source instantiations that are scheduled to achieve the goal. It must also rerate the goal's subgoals (and, in turn, their subgoals) to reflect the new rating (see Section 4.2.6). In addition, a raised goal rating may cause the goal to be above the transmission rating threshold for a node or to be raised enough to warrant retransmission of the goal, so the blackboard monitor must execute the precondition procedure for GOAL-SEND. If it indicates that the goal should indeed be transmitted, a GOAL-SEND knowledge source is instantiated.

If one or more new stimulus hypotheses are added to the goal, the blackboard monitor must notify the planner that the goal now has the potential to be achieved using the new hypotheses.

Goal reception events.

If one or more goals are received at a node while a knowledge source is executing, a goal reception event is signalled when that knowledge source completes. The blackboard monitor executes a GOAL-RECEIVE precondition procedure to determine if the node is truly interested in any of the goals. If so, a GOAL-RECEIVE knowledge source is instantiated to potentially modify the rating of each goal and to place them on the goal blackboard (at which time a goal creation event or a goal modification event is signalled by the blackboard monitor).

Goal satisfaction events.

When the blackboard monitor determines that a goal is satisfied, the planner is notified that it is no longer necessary to achieve the goal (see below). In addition, if the goal contains a request to transmit satisfying hypotheses to another node, a HYP-REPLY knowledge source is instantiated.

4.2.5 The planner.

The planner at each node attempts to achieve the goals created at the node by instantiating knowledge sources and by creating subgoals which, if satisfied, increase the likelihood of achieving higher-level goals. The planner also has a rudimentary capability for deciding when goals cannot be achieved locally. This section presents a high-level view of the planner's activities. Section 4.2.7 discusses subgoaling in the testbed. The implementation details involved in handling the asynchronous creation and modification of goals, hypotheses, and knowledge source instantiations efficiently are omitted.

Newly created goals.

When the blackboard monitor notifies the planner that a goal has been created on the goal blackboard, the planner first checks to see if the goal's rating is above the current planning threshold. If the rating is above the threshold, the planner looks at each knowledge source instantiation on the scheduling queue that was stimulated by goals that overlap the newly created goal to see if that knowledge source instantiation can potentially satisfy the newly created goal. (The overlapping goals are determined by the blackboard monitor.) The newly created goal is added as a stimulus goal to any such knowledge source instantiations, and the knowledge source instantiations are rerated.

The planner can also elect to attempt to satisfy the newly created goal directly. In the testbed implementation, each node's planner has a preference ordering for knowledge sources. The planner selects the first knowledge source in this ordering that can create hypotheses on the same blackboard level as the goal and executes that knowledge source's precondition procedure. If the precondition procedure estimates that the knowledge source is likely to generate one or more hypotheses that can potentially satisfy the goal, a knowledge source is instantiated, rated by the scheduler, and placed on the scheduling queue. If the estimation is that the knowledge source will fail to generate a hypothesis for the goal, the planner repeats this procedure with the next knowledge source in the preference ordering. If no local problem solving knowledge source is found for the goal and no lower-level data is expected to provide input data for these knowledge sources, a GOAL-HELP knowledge source can be instantiated.

Modified goals.

When the blackboard monitor notifies the planner that a goal's rating has been increased, the planner checks to see if it rating was previously below the planning threshold value. If it was previously below the threshold, the planner treats the goal as if it were a newly created goal. If the goal was previously above the threshold and there are no new stimulus hypotheses attached to the goal, the planner needs to do nothing else (the scheduler has already rerated any pending knowledge source instantiations that were stimulated by the goal).

If there are new stimulus hypotheses, however, a knowledge source with a higher execution preference that was previously unable to create a hypothesis for the goal may now be able to do so. The planner again executes the precondition procedures for knowledge sources with higher preferences than the knowledge source currently pending for the goal (if one exists), looking for one that may create hypothesis for the goal. If a higher preference knowledge source is found, it is instantiated and rated. The currently pending knowledge source (if one exists) remains on the queue to be used in the event the goal is not satisfied by the more preferred knowledge source instantiation. It is removed from the queue by the planner if the goal is satisfied before it is executed (see below).

Satisfied goals.

Whenever a hypothesis is created, it is linked with any goals that it satisfies (at least in part) by the blackboard monitor. When the currently executing knowledge source instantiation is finished, the blackboard monitor informs the planner of all goals that have had new satisfying hypotheses linked with them or the belief of an existing satisfying hypothesis increased by the knowledge source instantiation. For each of these goals, the planner must determine if the goal is sufficiently satisfied (see Section 4.3.3) that additional work on the goal is unnecessary. If the goal is sufficiently satisfied, the planner must remove from the scheduling queue all pending knowledge source instantiations that are working solely toward the achievement of the goal.

4.2.6 Rating goals.

Table 6 describes the rating calculation used for locally stimulated goals in the testbed. The goal rating calculation is the maximum of two components: one based on the hypotheses that directly stimulated creation of the goal and one based on the ratings of any goals having the goal as a subgoal. The stimulus hypotheses component includes a weighting specified in the local processing interest areas of the node.

Received goals have no local stimulus hypotheses. Instead, the stimulus hypotheses component of received goals is computed directly by the GOAL-RECEIVE knowledge source using the rating of the received goal, the credibility weight specified in the goal reception interest areas, and the local processing interest areas of the node.

There is a subtle problem associated with using the beliefs of received hypotheses in calculating the ratings of local goals resulting from them. If a node receives hypotheses with high belief values from another node, it will create highly rated goals indicating that the node should strongly attempt to use the received hypotheses even if there is little supporting local evidence. While this is fine for an exterally-directed control regime, it is inappropriate for a strongly self-directed control policy.

The rating of a locally created goal is calculated as:

$$\text{Rating(goal)} = \text{MAX}[\text{SHC(goal)},\text{SGC(goal)}]$$

The stimulus hypotheses component (SHC) rating is:

$$\text{SHC}(g) = \text{IAW}(g) * \max_{h \text{ in } \text{SH}(g)} [\text{FW}(h) * \text{belief}(h)]$$

where:

IAW(g)	= the maximum importance weight in any local processing interest area that overlaps the goal g
SH(g)	= the hypotheses that stimulated the creation of the goal g
FW(h)	= the maximum focusing-weight in any received hypothesis interest area that overlaps the hypothesis h (or 1 if the hypothesis h was not received from another node)
belief(h)	= the belief of hypothesis h.

The supergoal component (SGC) rating is:

$$\text{SGC}(g) = \max_{sg \text{ in } \text{SG}(g)} \text{rating}(sg)$$

where:

SG(g)	= the supergoals of goal g (goals that have g as a subgoal)
rating(sg)	= the rating of supergoal sg.

Table 6: Goal Rating Calculation.

The goal rating calculation is the maximum of two components: one based on the hypotheses that directly stimulated creation of the goal and one based on the ratings of any goals having the goal as a subgoal. The stimulus hypotheses component includes a weighting specified in the local processing interest areas of the node. The stimulus hypotheses component of received goals is computed directly by the GOAL-RECEIVE knowledge source using the rating of the received goal, the credibility weight specified in the goal reception interest areas, and the local processing interest areas of the node.

The self-directed control regime requires the exchange of highly rated hypotheses without having the receiving node use their beliefs in its local problem solving activity decisions. The credibility weighting in the hypothesis transmission interest areas cannot be used because it lowers the belief of received hypotheses. Instead, an additional focusing-weight parameter (contained in hypothesis reception interest areas) is used in calculating the rating of goals that are stimulated directly from received hypotheses. This parameter is used to diminish (or raise) the hypothesis belief value used in the goal rating calculation without changing the belief value of the received hypothesis. Thus strongly believed data received from another node can be accepted as such, to be incorporated into the local solution if it is relevant, without directly changing the course of local problem solving. (Of course, if received hypotheses are incorporated into the node's developing solution they will indirectly affect future problem solving focusing decisions.)

4.2.7 Subgoaling.

In addition to instantiating knowledge sources to achieve a goal, the planner can also create subgoals that reflect the importance of lower-level data in achieving the original goal and that, if satisfied, increase the likelihood of achieving the original goal. Subgoaling is an effective means of focusing low-level synthesis activities based on high-level expectations.

Consider the following situation. Problem solving at the node is restricted to location synthesis until the vehicle location level at which point track formation and extension is to be performed. The creation of signal location hypothesis H:SL:01 on the data blackboard causes the blackboard monitor to create group location goal G:GL:01 on the goal blackboard (Figure 31). This goal indicates that the system should attempt to form a group location hypothesis using H:SL:01. The planner next instantiates knowledge source instantiation S:SL:GL:01 to try to achieve this goal. The rating of a knowledge source instantiation is a function of the belief of its stimulus hypotheses and the priority rating of its stimulus goals (if any). The priority of a goal is a function of the belief of its stimulus hypotheses, its level on the blackboard, and its relationships with other goals. Assume that H:SL:01 is weakly believed and consequently S:SL:GL:01 is given a low execution rating. Processing continues with other signal location hypotheses and eventually creates a vehicle track hypothesis H:VT:01 with a moderately high belief. The creation of this hypothesis causes a number of goals to be created, including the goal shown in the figure, G:VT:02. This goal indicates that the node should attempt to extend H:VT:01.

The planner uses its goal-to-subgoal domain knowledge for decomposing this high-level goal into a signal location level subgoal, G:SL:03. This subgoal indicates the area in which signal location hypotheses are needed in order to eventually extend the vehicle track hypothesis. Subgoal G:SL:03 is given the same priority rating as its parent goal G:VT:02. The planner finds that H:SL:01 has already been

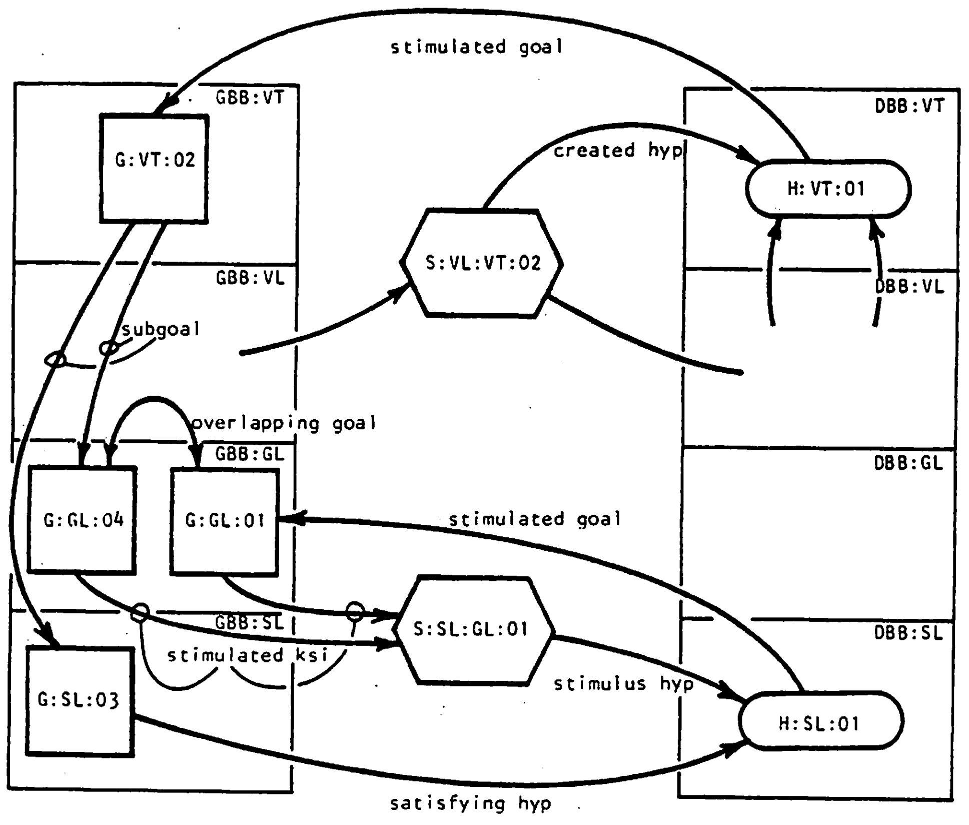


Figure 31: Focus-of-Attention through Subgoaling.

An example showing how subgoaling is used to increase the priority rating of a low-level knowledge source instantiation based on the creation of a goal representing a high-level expectation (G:VT:02). The predicted vehicle track extension goal is subgoaled downward to raise the rating of a pending knowledge source instantiation (S:SL:GL:01) whose output could potentially extend the vehicle track hypothesis (H:VT:01).

created in this area and can satisfy G:SL:03. The planner then creates subgoal G:GL:04 and finds that goal G:GL:01 overlaps with it. The planner adds G:GL:04 as a second stimulus goal of the low-rated knowledge source instantiation S:SL:GL:01. The addition of the higher priority goal causes the rating of the knowledge source instantiation to be increased based on its potential contribution to the track extension goal G:VT:02.

A similar situation occurs when a node receives a high-level goal from another node but has yet to synthesize that data necessary to satisfy the goal. Subgoaling this goal (and potentially instantiating knowledge sources to achieve the subgoals) identifies which lower level data should be synthesized to achieve the higher-level goal.

In the testbed, the domain knowledge needed to perform subgoaling is taken from the grammar and the behavior of knowledge sources (tolerance of location and frequency class shifting at the signal and group levels). Inverting the event class relationships permits the planner to determine the event classes and active-time-region-lists for subgoals to be created at any level. Subgoals normally have the maximum rating of any of their supergoals, however a weighting can be applied to this rating to raise or lower the subgoal's focusing influence (see below).

Because subgoaling requires some effort, its use needs to be controlled. In the testbed, subgoaling is controlled in two ways: by restricting subgoaling to particular levels and by a minimum rating threshold for a goal to be subgoaled.

Each node is supplied with a subgoaling specification data structure that lists the levels at which goals are to be subgoaled and the levels at which these goals are to be created (Table 7). The shrink-factor and rating-weight parameters can be used to create a number of concentric subgoals of increasing sizes and decreasing ratings. Such nesting of goals is useful in representing situations where the likelihood of creating suitable hypotheses decreases with distance from the center of the goals.

Each node also uses three goal rating threshold values to decide whether a goal on a subgoaling level should be subgoaled. The internal-subgoal threshold specifies the minimum goal rating threshold of goals stimulated by locally-created hypotheses. Such locally-stimulated goals must be rated at or above this threshold for subgoaling to occur. Similarly, the received-hypotheses-subgoal threshold specifies the minimum goal rating threshold of goals stimulated directly by hypotheses received from other nodes. Finally, the received-goals-subgoal threshold specifies the minimum goal rating threshold of goals received from other nodes. If a particular goal falls into more than one of these categories, the minimum of the appropriate thresholds is used.

Since subgoaling can have a significant effect on the local activities of a node, the relative settings of these three parameters strongly influence the balance between local and external direction. Experiments with various settings of these parameters are presented in Chapter V.

The subgoaling specification at each node is a list:

```
((goal-level-list
      (subgoal-level-list ((shrink-factor rating-weight)
                           (shrink-factor rating-weight)
                                         ...
                           (shrink-factor rating-weight)))
      (subgoal-level-list ((shrink-factor rating-weight)
                           (shrink-factor rating-weight)
                                         ...
                           (shrink-factor rating-weight)))
                             ...
      (subgoal-level-list ((shrink-factor rating-weight)
                           (shrink-factor rating-weight)
                                         ...
                           (shrink-factor rating-weight))))
                          ...
 (goal-level-list
      (subgoal-level-list ((shrink-factor rating-weight)
                           (shrink-factor rating-weight)
                                         ...
                           (shrink-factor rating-weight)))
      (subgoal-level-list ((shrink-factor rating-weight)
                           (shrink-factor rating-weight)
                                         ...
                           (shrink-factor rating-weight)))
                             ...
      (subgoal-level-list ((shrink-factor rating-weight)
                           (shrink-factor rating-weight)
                                         ...
                           (shrink-factor rating-weight)))))
```

goal-level-list =	a list of levels where created goals are subgoaled (if they are rated above the appropriate belief threshold);
level-list =	a list of levels at which subgoals will be created;
shrink-factor =	the amount that the X and Y dimensions are reduced for each region in the subgoal;
rating-weight =	the weight multiplied by the original goal rating when calculating the new subgoal rating.

Table 7: Subgoaling Specification Data Structure.

The subgoaling specification data structure at each node specifies the levels where created goals are subgoaled and the levels, sizes, and ratings of the subgoals. Its values are strongly related to the local processing and communication interest areas.

4.2.8 Knowledge source precondition procedures.

The overall performance of each node depends on the ability of each knowledge source precondition procedure to correctly anticipate the eventual behavior of executing the knowledge source instantiation. Normally, this problem of selecting the knowledge source that has the appropriate balance between expected execution cost and likelihood of success would be determined by executing the precondition procedures of all knowledge sources that could potentially achieve the stimulus goals. Uncertainty as to the behavior of knowledge source instantiations (as specified in the response frame) makes it difficult for the planner and scheduler to decide what knowledge source instantiations to execute.

In order to investigate the effects of this uncertainty the testbed simulation preexecutes the entire knowledge source as the precondition procedure. The knowledge source does not actually create any hypotheses or goals, but instead places an exact specification of their attributes in the output-set attribute of the knowledge source instantiation. The output-set provides an exact description of what the knowledge source instantiation will do if executed. (The output-set is updated if the input context of the knowledge source instantiation is modified while it is awaiting execution.) The actual hypotheses or goals are created when the knowledge source instantiation executes.

The information contained in the output-set allows the knowledge source instantiation rating to be made with perfect knowledge of the knowledge source instantiation's behavior. Precondition procedures with less than perfect estimation abilities are simulated by perturbing these perfect ratings. The details are described in the next section.

4.2.9 Rating knowledge source instantiations.

Table 8 details the knowledge source instantiation rating calculation used in the testbed. It is basically a weighted sum of a data-directed and a goal-directed component. The data-directed component captures the expected belief of an output hypothesis (as specified in the knowledge source instantiation's output-set attribute). The goal-directed component measures the ratings of goals that would be satisfied (at least in part) by each output hypothesis. The goal-weighting parameter adjusts the importance given to satisfying highly-rated goals versus producing strongly believed hypotheses. The weighted sum of these two components is computed for each output hypothesis in the knowledge source instantiation's output-set attribute and the maximum value (multiplied by the knowledge source efficiency estimate) is used as the base rating for the knowledge source instantiation.

Since the testbed precondition procedures precompute the actual output hypotheses of the knowledge source instantiation, the scheduler's base rating calculation uses the exact beliefs of the output hypotheses and the goals that they satisfy. Gaussian noise can be added to this base rating to simulate the effects of knowledge source precondition procedures that are imperfect in their estimation of output hypotheses's beliefs and of goal satisfaction.

The knowledge sources' precondition procedures use information localized to a particular region of the data blackboard in estimating the belief values of output hypotheses. On the other hand, the scheduler is in a position to determine how a knowledge source

Table 8: Knowledge Source Instantiation Rating Calculation.

The knowledge source instantiation rating calculation is basically a weighted sum of a data-directed and a goal-directed component. The data-directed component captures the expected belief of an output hypothesis (as specified in the knowledge source instantiation's output-set attribute). The goal-directed component measures the ratings of goals that would be satisfied (at least in part) by an output hypothesis. The goal-weighting parameter can be adjusted to change the importance given to producing strongly believed hypotheses versus satisfying highly-rated goals. Gaussian noise is added to the rating calculation to simulate knowledge source precondition procedures with imperfect output hypothesis estimation capabilities.

The rating of a knowledge source instantiation is calculated as:

$$\text{Rating(KSI)} = \text{N(s,BR(KSI))}$$

where:

N(s,m)	=	$m + [GN(s) * (1 - m^2)]$
GN(s)	=	a random variate in [-1,1] drawn from a gaussian distribution with mean 0 and standard deviation s.

The knowledge source instantiation base rating is:

$$\text{BR(KSI)} = \text{AE(KSI)} * \{ \max_{o \text{ in } O(KSI)} [gw*DD(KSI,o) + (1-gw)*GD(o)]\}$$

where:

AE(KSI)	=	the assumed average efficiency (execution cost versus performance) of the knowledge source executing the KSI
O(KSI)	=	the set of output hypotheses of the KSI
gw	=	the goal weighting (a constant in [0,1] that controls the balance between data- and goal-directed control).

The data-directed component is: $DD(KSI,o) = (1-ow)*BV(o) + ow*CV(o)$

where:

ow	=	the oracle weighting (a constant in [0,1] that controls the degree that the consistency of output hypothesis is used in the rating calculation)
BV(o)	=	the (resolved) belief value of the output hypothesis o
CV(o)	=	1 if output hypothesis o is a consistent hypothesis (based on the consistency blackboard); 0 otherwise.

The goal-directed component is: $GD(o) = \text{MAX}\ [MGR(o), WAGR(o)]$

where the maximum rated goal satisfied by output hypothesis o is:

$$MGR(SG) = \max_{g \text{ in } SG(o)} GR(g)$$

and the weighted average rating of the satisfied goals is:

$$WAGR(SG) = \text{MIN}\ [1, \underset{g \text{ in } SG(o)}{\text{AVERAGE}}\ w*GR(g)]$$

and:

SG(o)	=	the set of goals satisfied (at least in part) by output hypothesis o
GR(g)	=	the rating of goal g
w	=	1.0 if $\lvert SG(o)\rvert$ is 1
		1.1 if $\lvert SG(o)\rvert$ is 2
		1.2 if $\lvert SG(o)\rvert$ is 3
		1.3 if $\lvert SG(o)\rvert$ is 4
		1.4 if $\lvert SG(o)\rvert$ is 5 or larger.

Table 8: Knowledge Source Instantiation Rating Calculation.

instantiation's expected output hypotheses fit into the overall developing solution at the node. This difference in viewpoint leads to an interesting engineering issue. Should the scheduler rely solely on the myopic estimations of the precondition functions in rating a knowledge source instantiation or should it be given domain-dependent knowledge of its own to determine consistencies between knowledge source instantiations? To experiment with this issue, an oracle weighting in the data-directed component can be used to introduce the consistency of each output hypothesis (as specified on the consistency blackboard) into the rating calculation. As with the knowledge source instantiations themselves, this consistency information is used to simulate the effects of developing additional knowledge which can better detect the consistencies among hypotheses.

4.3 Additional Issues

This section discusses several additional issues associated with the goal-directed Hearsay-II architecture.

4.3.1 Plans in the goal-directed Hearsay-II architecture.

The explicit goal structure allows the planner to construct plans consisting of multistep sequences of knowledge source executions. A prime example is the S:SL:GL, S:GL:VL, EF:VL/VT:VT knowledge source sequence needed to extend a track into the next time frame. Unfortunately, the current implementation does not adequately represent such sequences. What is needed is an extra goal blackboard plane to contain plans (sequences of goals on goal blackboard). This plan plane would contain plans whose goals and knowledge source instantiations would be associated with particular plans and would be distinguishable from one another. The scheduler would use these competitive and cooperative relationships between knowledge sources and goals to discontinue work on goals in a strategy which has failed.

4.3.2 Balancing the cost of goal processing.

Complex goal processing is not without cost, and as the overhead of goal processing increases, it is important to balance planning activities with knowledge source execution. The scheduler should perform the allocation of processing resources -- both to the planner and the knowledge sources. It is the scheduler which has access to the ratings on knowledge source instantiations and the priorities of goals. The ratings and the relationships between the goals and knowledge source instantiations provide the scheduler with the information necessary to determine the best course for improving the state of the system. Techniques for reasoning about the balance between planning the consequences of actions versus performing them to discover the result are needed. The work on integrating decision theory with heuristic search by Feldman and Sproull [FELD77] is a first step in this direction.

Complex goal-processing raises the issue of whether the planner itself should be implemented as a data-directed system with its own planning knowledge sources and whether that system should be augmented with a (meta) goal blackboard and goal-processing mechanisms. Goals requesting changes on the goal blackboard can be used to explicitly represent the problem-solving strategies of the system. Work by Davis, Barbara and Frederick Hayes-Roth, Stefik, and Erman, London, and Fickas are representative of similar uses of meta-level problem solving [DAVI80, HAYE79, STEF80, ERMA81]. Such meta-level goals would represent strategies for the planner. Subgoaling a high-level, expectation-based goal to low-level goals and then driving-up the appropriate low-level hypotheses upward is an example of a useful strategy which could be represented by a meta-level goal.

If there are a number of meta-level goals, then a strategy for choosing between them is needed. This raises the problem of controlling the meta-controller, and so on. One approach is to add control layers until the highest level controller becomes a simple procedure. A second approach is to introduce a controller which can reason about its own control decisions as well as those it is making for the lower levels (while avoiding problems associated with self-reference). The testbed has one level of meta-level goals, the organizational structuring level described in Section 4.4.

4.3.3 The goal satisfaction problem.

An important aspect of the goal-directed Hearsay-II architecture is determining when a goal is satisfied by hypotheses on the data blackboard. As discussed earlier in this chapter, it is the blackboard monitor's responsibility to make this determination. Given a goal and a set of hypotheses, it is easy to identify those hypotheses that overlap the goal's level, active-time-region-list, and event-classes attributes (and fall within the goal's inactive-time-region-list attribute). The blackboard monitor connects these hypotheses to the goal using the satisfying-hypotheses goal attribute.

However, two questions remain to be answered. Given a goal and its set of satisfying hypotheses:

1. Is the goal sufficiently satisfied that executing additional knowledge sources to work toward the goal would be superfluous?

2. Is the goal sufficiently satisfied that work on higher-level goals should be allowed to proceed?

As we will see, the answers to these two questions are beyond the modest capabilities of the blackboard monitor and instead must be addressed by the local node planner.

The first question is very important if the planner has a number of knowledge sources at its disposal for working on the goal. We would like to avoid the creation of identical hypotheses (with identical beliefs) using different knowledge sources because work on the goal was not curtailed. The difficulty in determining when a goal has been "sufficiently satisfied" is due to an incomplete specification of the goal. When a goal is created from a stimulus hypothesis, it is supplied with level, active-time-region-list, inactive-time-region-list, and event-classes attributes. However, the goal does not specify how many hypotheses could (or should) be created within its boundaries or their expected belief values.

A simple technique for estimating these missing values is to use the predicted output hypotheses (and their expected belief values) provided by the precondition procedures of all knowledge sources able to work on the goal. If a precondition procedure predicts that its knowledge source will create a new hypothesis within the boundaries of the goal or will substantially increase the belief of an existing satisfying hypothesis, then work on the goal should not be stopped until that knowledge source has been executed. Only when the precondition procedure of all unexecuted knowledge sources predict that their knowledge source will not contribute new information to the goal can the goal be considered as satisfied. A satisfied goal must be reactivated if a new stimulus hypothesis is added, and the precondition procedures for all knowledge sources that can work on the goal must be reevaluated.

If the precondition procedure estimates can be determined to never fall short of the knowledge source's actual performance, only truly superfluous knowledge source executions will be eliminated, and the hypotheses and belief values satisfying the goal will be equivalent to executing all the knowledge sources. The closer the estimates are to the actual knowledge source performance (without falling short of the actual performance), the better the pruning of superfluous knowledge source activity.

Although the precondition procedure estimates do aid the planner in deciding when to curtail work (at least temporarily) on local problem solving knowledge sources associated with a goal, they are insufficient for deciding when to stop work on the goal altogether. The precondition procedure estimates are based on the current input context available to the knowledge sources. It may be that a knowledge source's input context is incomplete because data on lower problem solving levels has not been driven up to the level of the input context. Stated differently, there exist unsatisfied lower-level goals that can contribute to the input context of the knowledge source.[8] Since satisfying these lower-level goals can add lead to new information in the input contexts of the higher-level knowledge sources, a goal should not be considered "completely satisfied" until its associated lower-level goals are completely satisfied.

The potential for obtaining additional satisfying hypotheses from other nodes also complicates the goal satisfaction decision. While it may be reasonable to expect the precondition procedure of local problem solving knowledge sources to provide reasonably accurate estimates, accurately estimating what might be received by asking another node to reply with any hypotheses it has that satisfy the goal is unrealistic. In addition, a node can receive unsolicited information from another node that adds new stimulus hypotheses to a goal. In the latter situation, the node has no choice but to reactive the restimulated goal (and eventually any higher-level goals that have new information supplied by working on the restimulated goal). In the former situation, the node may be able to use beliefs about the other node's problem solving activities to roughly estimate the information that might be replied in response to sending a help goal to that node. While this can be viewed as a problem associated with writing precondition procedures

8. Hayes-Roth and Lesser describe two types of relationships between knowledge source instantiations and goals, termed direct and indirect goal satisfaction [HAYE77]. _Direct goal satisfaction_ means that a knowledge source instantiation is a candidate for achieving a goal because of its potential for creating hypotheses matching the desired attributes of the goal. _Indirect goal satisfaction_ means that the knowledge source instantiation does not directly satisfy the goal, but increases the probability that the goal will be satisfied by another knowledge source instantiation by producing hypotheses useful in the achievement of the goal.

for GOAL-SEND knowledge sources, the planner needs to decide whether or not it is worth asking for help in satisfying a goal or whether the goal can be sufficiently satisfied by local information.

Deciding if the goal is sufficiently satisfied for work to begin on higher-level goals is made easier by the precondition procedure estimates. If the execution of knowledge sources working is allowed to begin as soon as any hypotheses exist in their input contexts can lead to repeated execution of those knowledge sources as additional low-level hypotheses are created. On the other hand, delaying work on knowledge sources associated with a higher-level goal until any related lower-level goals are completely satisfied reduces the repeated execution of higher-level knowledge sources, but it also delays the development of high-level predictive information. For example, if a signal location hypothesis can be quickly synthesized into one or more vehicle location hypotheses (assuming the node is working only on the location levels), the vehicle location hypotheses can be used to predict what signal location hypotheses are missing and need to be identified.

While determining when higher-level work should be performed is, in effect, an overall problem solving strategy rather than a simple, syntactic, goal satisfaction decision, the precondition procedure estimates can be used by the planner to estimate much additional improvement might be made in the data associated with low-level goals.

The goal satisfaction issue is a local instance of the overall network "stopping" problem. A major issue with a functionally accurate, cooperative distributed problem solving network is determining when (and if) the nodes have converged to an acceptable solution. It is possible for each node to decide it has satisfied all its goals without realizing that other nodes have additional information that could greatly improve its portion of the answer map. This issue has not been directly addressed in this research and remains a crucial area of open research in the area of functionally accurate, cooperative distributed problem solving networks.

4.4 A Framework for an Organizational Designer

The preceding sections of this chapter described the local control framework used at each testbed node, including the goal-directed Hearsay-II architecture, the local node planner, and the interest areas and subgoaling specifications used to influence the activities of the node. Still to be discussed is the how organizational structuring decisions interface with the local control framework. This is the subject of this section.

Because all activity decisions made by a node are influenced by its interest areas and subgoaling specifications, a node's organizational responsibilities can be established and changed by simply modifying these data structures. These data structures can be viewed as rudiments of a third blackboard -- an organizational blackboard containing the organizational roles and responsibilities for the node. The specification data structures themselves do not provide an explicit, high-level representation of these organizational roles and responsibilities, but instead serve as a low-level "job description" of those activities a node is should be performing and those activities a node should be avoiding. While organizational structuring could be performed by directly changing these structures (and is the approach used in the experiments reported in Chapter V), an indirect approach allows the node to adopt or reject its organizational roles.

Instead of modifying the specifications directly, a second, separate set of node activity specification data structures is kept at each node. The original interest areas and subgoaling specifications remain as the behavioral command center of the node. Their settings directly control the node's activities. The second specifications set forms the lowest level of the full-fledged organizational blackboard. They are the result of elaborating higher-level organizational roles and responsibilities into an "organizational job description". The complete structure of this organizational blackboard, and the processing needed to perform the elaboration, remain an open research issue. What is important here is that the specifications directly controlling the behavior of a node and the behavior suggested by the organizational structure are separated. The node undertakes its organizational activities only by transferring organizational specifications into its interest areas and subgoaling specifications.

The activities of a node should also be influenced by its potential for performing them. A node is continually receiving sensory data and hypotheses from other nodes. This information provides numerous opportunities for local node activities. However, the node's interest areas and subgoaling specifications (possibly set from the organizational blackboard) may be strongly opposed to performing these activities. The node's potential for work is represented on a fourth blackboard, the local node focusing blackboard. This blackboard contains low-level specifications that indicate where the node perceives there is substantial work it is able to perform. As with the organizational specifications, these focusing specifications can be transferred to the node's interest areas and subgoaling specifications, at which point the node will actively pursue these activities.

Given the above structure, a mechanism implementing node skepticism can easily be added (see Figure 32). When the roles and responsibilities represented in the organizational blackboard are in conflict with the criteria on the local node focusing blackboard, an arbiter for determining the actual interest areas and subgoaling specifications is needed. This arbiter is, in fact, implementing node skepticism. Favoring the specifications on the organizational blackboard make the node's behavior less skeptical (more of a "company node"), while favoring the local node focusing specifications make the node more responsive to its ability to immediately perform work.

The existence of the organizational and local node focusing blackboards also help indicate when the portion of the network organizational structure relating to the node needs changing. A strong mismatch between the two blackboards is a sign of trouble, and the information contained in the focusing blackboard can be a valuable aid in determining new roles and responsibilities.

The next chapter presents experiments performed with the testbed in which nodes' interest areas and subgoaling specifications were varied (directly from the environment file). These experiments demonstrate that node behavior can indeed be controlled using these data structures.

Figure 32: The Organizational and Local Node Focusing Blackboards and Node Skepticism.

The behavior of a node is controlled by its interest areas and subgoaling specifications. These data structures are modified by behavioral specifications contained on the organizational blackboard (representing organizational roles and responsibilities) and on the local node focusing blackboard (representing the ability of the node to perform work on particular parts of the overall problem). Since these two behavioral specifications are not necessarily in agreement, an arbiter is required for determining the relative influence of each. This arbiter is at the heart of node skepticism.

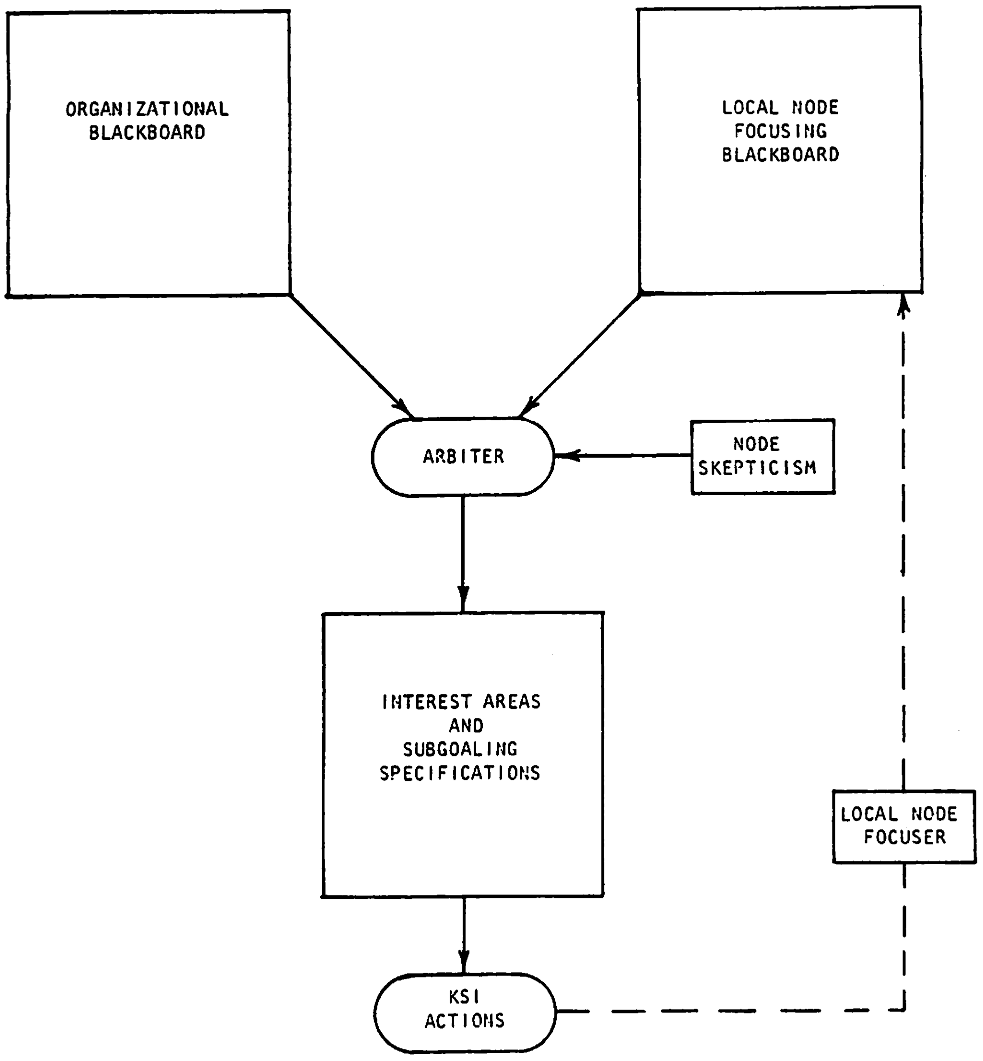

ORGANIZATIONAL BLACKBOARD
LOCAL NODE FOCUSING BLACKBOARD
ARBITER
NODE SKEPTICISM
INTEREST AREAS AND SUBGOALING SPECIFICATIONS
LOCAL NODE FOCUSER
KSI ACTIONS

Figure 32: The Organizational and Local Node Focusing Blackboards and Node Skepticism.

I really think I'm entitled to an answer to that question.

— HAL in Stanley Kubrick's 2001: A Space Odyssey

CHAPTER V

DISTRIBUTED VEHICLE MONITORING TESTBED EXPERIMENTS

This chapter presents experimental evidence demonstrating the flexibility of the testbed, the capabilities of the goal-directed Hearsay-II architecture and its local node planner, and the impact of organizational structuring decisions on the local control component of a testbed node.

In the next section, the two distributed vehicle monitoring environmental scenarios used in these experiments are described. This is followed by a description of the performance of a centralized (single node) architecture in these environments. Later sections present the performance results of four and five node network architectures operating under different organizational structuring specifications.

5.1 The Environmental Scenarios

The experiments described in this chapter involve two different distributed vehicle monitoring environments: a straight, single vehicle pattern with a parallel ghost track (hereafter called the "straight vehicle" environment) and a bent, single vehicle pattern with a ghost track extension (called the "bent vehicle" environment). Both environments are designed to test the network's ability to distinguish the actual track from a particularly difficult ghost track. To this end, these environments involve a simple grammar and specific sensory data ambiguity.

5.1.1 The grammar.

The grammar for these environments is the example grammar from Chapter II and reproduced here (Figure 33). Pattern classes 1 and 2 represent single vehicles of class 1 and 2, respectively. Pattern class 3 represents a two vehicle pattern containing one vehicle of class 1, displaced (3,3) units from the center of the pattern, and one of class 2, displaced (-3,3) units. (These displacements are not illustrated in the figure.) All supports for the two vehicle classes are disjoint except for signal class 18. Sensing a signal of class 18 provides weak support for both vehicle classes, and consequently, for all three pattern classes.

The tracking component of this grammar (also not illustrated) specifies a maximum velocity for a vehicle as 4 units per time frame with a maximum acceleration of 2 units per time frame.

5.1.2 The sensors.

The sensor configuration in these environments is shown in Figure 34. The area to be monitored is twenty-two units square, with vertices at (0,0), (22,0), (22,22), and (0,22). Four sensors with identical characteristics are located at (6,16), (16,16), (6,6), and (16,6). Each sensor uniformly covers a twelve unit square area with the sensor at its center. This leads to a two unit overlap among the sensors' coverages. Overlapping sensor coverage is not a problem

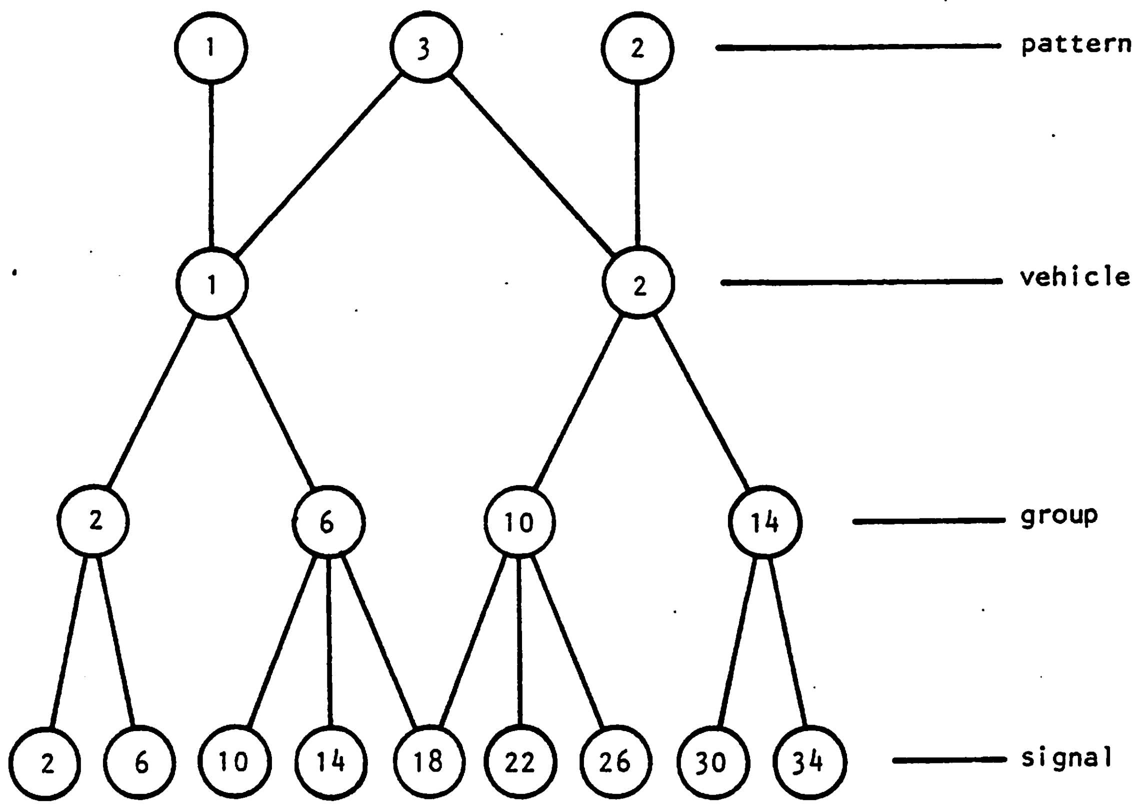


Figure 33: Grammar Used in the Testbed Experiments.

This illustrative grammar contains three pattern classes. Pattern classes 1 and 2 represent single vehicles of class 1 and 2, respectively. Pattern class 3 represents a two vehicle pattern containing one vehicle of class 1 and one of class 2. The appearance of signal class 18 in the supports of both vehicle classes adds minor confusion.

---

solving requirement and is actually a source of redundant processing in the multinode experiments.

In order to precisely control the ambiguity present in these environments, the random shifting of signal location hypotheses in location and frequency class (as described in Section 2.3.4) is turned off. All supporting signal location hypotheses of an actual or ghost

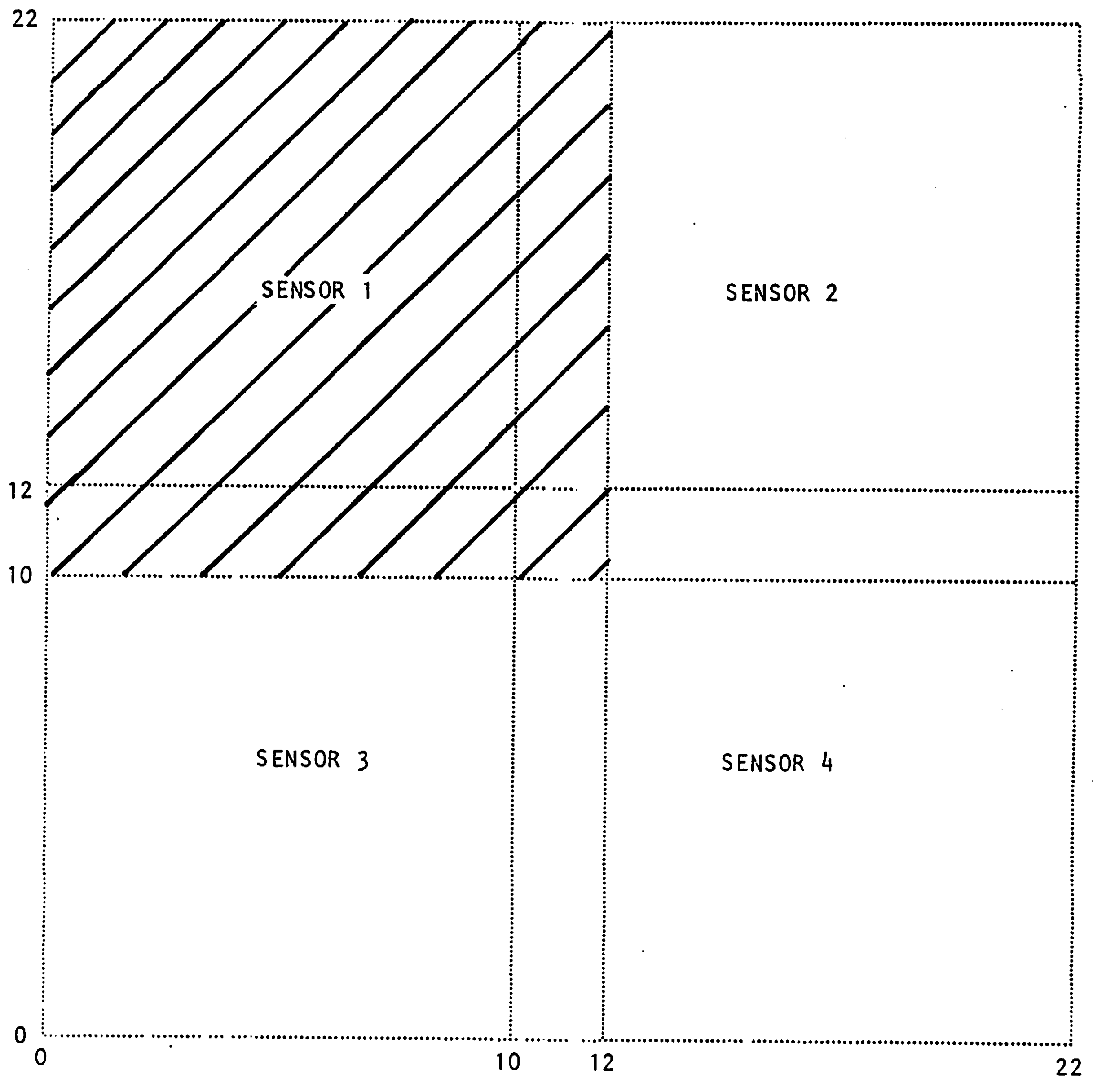


Figure 34: Sensor Configuration.

The four sensors covering the (0,0) to (22,22) monitoring area (location (0,0) is at the lower left). The area covered by Sensor 1 at location (6,16) is shaded.

vehicle are generated by the sensors at their specified position, frequency, and belief.

5.1.3 Knowledge sources.

The generation of the answer map is restricted to the

$$\begin{tikzcd}
\text{Signal Location} \arrow[r] & \text{Group Location} \arrow[r] & \text{Vehicle Location} \arrow[r] & \text{Vehicle Track} \arrow[r] & \text{Pattern Track}
\end{tikzcd}$$

synthesis path in these experiments. This path is interesting because tracking is performed only once a vehicle has been identified. Tracking at such a high abstraction level is appropriate when identification of vehicle types is less difficult than tracking their movements.

The basic testbed problem solving knowledge sources enabled in these experiments are listed in Table 9. In these experiments, each basic problem solving knowledge source requires one processing time unit to execute, no matter what work it performs. This is specified in the environment file by setting the runtime of each knowledge source to have a fixed overhead of one time unit and a multiplier of zero time units for each stimulus hypothesis (see the Knowledge-Source-Set Definitions description in Appendix A). This uniform execution time provides a convenient means for referring to processing time in the network (as distinct from environmental time, measured in time frames). A <u>network cycle</u>, or simply <u>cycle</u>, is defined to be one processing time unit expended at each node in the network. Since the basic problem solving knowledge source requires one processing time unit, three network cycles in a two node network indicates that a maximum of six knowledge source instantiations can be executed. If a node has no work to perform during a cycle, its potential knowledge source execution is lost.

<u>Knowledge Sources</u>

Location Synthesis:	Forward Extension:	Forward Merging:	Forward Location-to-Track Joining:
S:SL:GL S:GL:VL	EF:VT/VL:VT	MF:VT:VT MF:PT:PT	JF:VL/VT:VT
Track Formation:	**Backward Extension:**	**Backward Merging:**	**Location-to-Track Joining:**
FT:VL:VT	EB:VT/VL:VT	MB:VT:VT MB:VT:VT	JB:VL/VT:VT
Track Synthesis:	**Hypothesis Transmission:**		**Hypothesis Reception:**
S:VT:PT	HYP-SEND:VT:VT		HYP-RECEIVE:VT:VT

Table 9: Knowledge Sources Used in the Testbed Experiments.

The fourteen basic problem solving knowledge sources used in the testbed experiments described in this chapter.

Except for the FRONTEND and SENSORS knowledge sources, the name each knowledge source has the form:

type : input-level(s) : output-level.

The resolver stage of each knowledge source is set to leave the beliefs of the output hypotheses generated by the candidate generator stage unchanged. This means that these experiments use only the simple knowledge incorporated in the candidate generators and do not use the interpretation contained on the consistency blackboard in generating an answer map.[1]

1. The candidate generator and resolver stages of knowledge sources are discussed in Section 2.3.7.

In these experiments, track merging is performed at both the vehicle track and the pattern track levels. This is interesting for several reasons.

First, merge knowledge sources are redundant in a single node network. Since all supporting location data is available at the node, the track extension and location joining knowledge sources can be used to extend any tracks created by the track formation knowledge source, FT:VL:VT. The presence of merge knowledge sources provides minor optimization since two overlapping or abutting vehicle track segments (formed opportunistically from stronger supporting data -- see below) can be spliced together without recomputing their structure using extends and joins.

The major reason the merge knowledge sources are included is for the multinode network experiments. In these experiments hypotheses are communicated only at the vehicle track level. Without merges, a node receiving a vehicle track segment from another node would be unable to incorporate it because of the lack of the underlying supporting location hypotheses needed by extends and joins.

A second point of interest is why track merge knowledge sources are also included at the pattern track level. A key issue in controlling the activities of a testbed node is how the space of possible interpretations should be searched. A completely breadth-first search in which all signal location data is exhaustively synthesized to pattern track hypotheses is computationally unrealistic. Instead, a more depth-first opportunistic strategy is used; signal location hypotheses with higher beliefs are synthesized to higher abstraction levels before work begins on lower believed signal location hypotheses. This is implemented in the testbed by increasing the importance of knowledge sources working at higher abstraction levels by changing the weights associated with the local processing interest areas (see Table 8 in Chapter IV).

In the experiments reported in this chapter, a strong bias for work at higher levels is used, resulting in an almost depth-first search of the space of interpretations. This greatly reduces the number of knowledge sources that are executed if a suitable solution path is selected. Knowledge sources working on the vehicle track level are given only 0.5 the importance of knowledge sources working at the pattern track level. The weighting used for the vehicle location level is 0.1 that of the pattern track level, and the weighting used for the signal location and group location levels 0.05 that of the pattern track level. Such a strong depth-first tendency would result in the following behavior if the pattern track merge knowledge sources were omitted in these experiments.

Assume two mergable vehicle track hypotheses are created. The planner can instantiate a track synthesis (S:VT:PT) knowledge source for each vehicle track hypothesis and a track merge (MF:VT:VT) knowledge source to merge the two vehicle track hypotheses. Since the scheduler is biased to push results to higher levels, the two synthesis knowledge sources are executed first. However, without a track merge knowledge source at the pattern track level the two mergable pattern track hypotheses cannot be combined into a single track hypothesis. Eventually the vehicle track merge knowledge source may be run and the merged vehicle track synthesized to a pattern track hypothesis, but this behavior is far from desirable. With the pattern track merge knowledge sources the merge is performed at the pattern track level, directly following the synthesis of the two pattern track hypotheses.

Such an almost depth-first behavior was selected for these experiments to accentuate the effects of goal processing, both positive and negative. When subgoaling from the developing high-level solution, the first hypotheses created at the higher levels can have a significant impact on the future processing at the node. If these hypotheses are part of the actual interpretation, the node will focus on the correct solution. If they are incorrect, the node will follow this false path. The two environmental scenarios created for these experiments are designed with this in mind. They are described in the following sections.

5.1.4 The straight vehicle environment.

The straight vehicle environment is designed to test the network's ability to use prediction to extend strongly sensed portions of an actual vehicle track through weakly sensed portions in the presence of a moderately sensed "ghost" track. Ghost tracks are a particularly problematic phenomenon in the distributed vehicle monitoring domain, caused by multiple propagation paths of the actual signals and by geometrical ambiguity in combining signals from multiple vehicles. The ghost track in this environment mirrors the actual vehicle track for eight consecutive time frames. This is unusual. Typically ghosts behave as normal vehicles for a brief period only to abruptly disappear or to turn at sharp angles and accelerate to infinite velocity [GREE82]. The ghost in this environment represents a "worst-case" situation, appearing as a normal vehicle with moderately strong sensory support.

In fact, the labels "actual" and "ghost" associated with the tracks represent the interpretations, nay intentions, of the author in specifying the environments. While the signal location data does support the actual track slightly better than the ghost track, which label should be associated with which track or whether they should both be labelled as ghost or actual vehicles is a difficult question because the actual number of patterns (or vehicles) in the environment is not known by the network. Therefore, the selection of the pattern track hypothesis with the highest belief is not sufficient. Additional processing to determine which detected patterns are considered "real" must be performed.

This additional "answer map generation" level of problem solving is not included in the testbed. Since the testbed is being used for investigating network coordination techniques, our main interest is in the behavior of the network during the course of problem solving. In the experiments reported in this chapter, the network is run until all "correct" pattern track hypotheses (as specified by the consistency blackboard) are created above a specified threshold.

The signal location data generated by the sensors is shown in Figure 35. The signal location hypotheses associated with the actual vehicle track (vehicle class 1) are detected as follows:

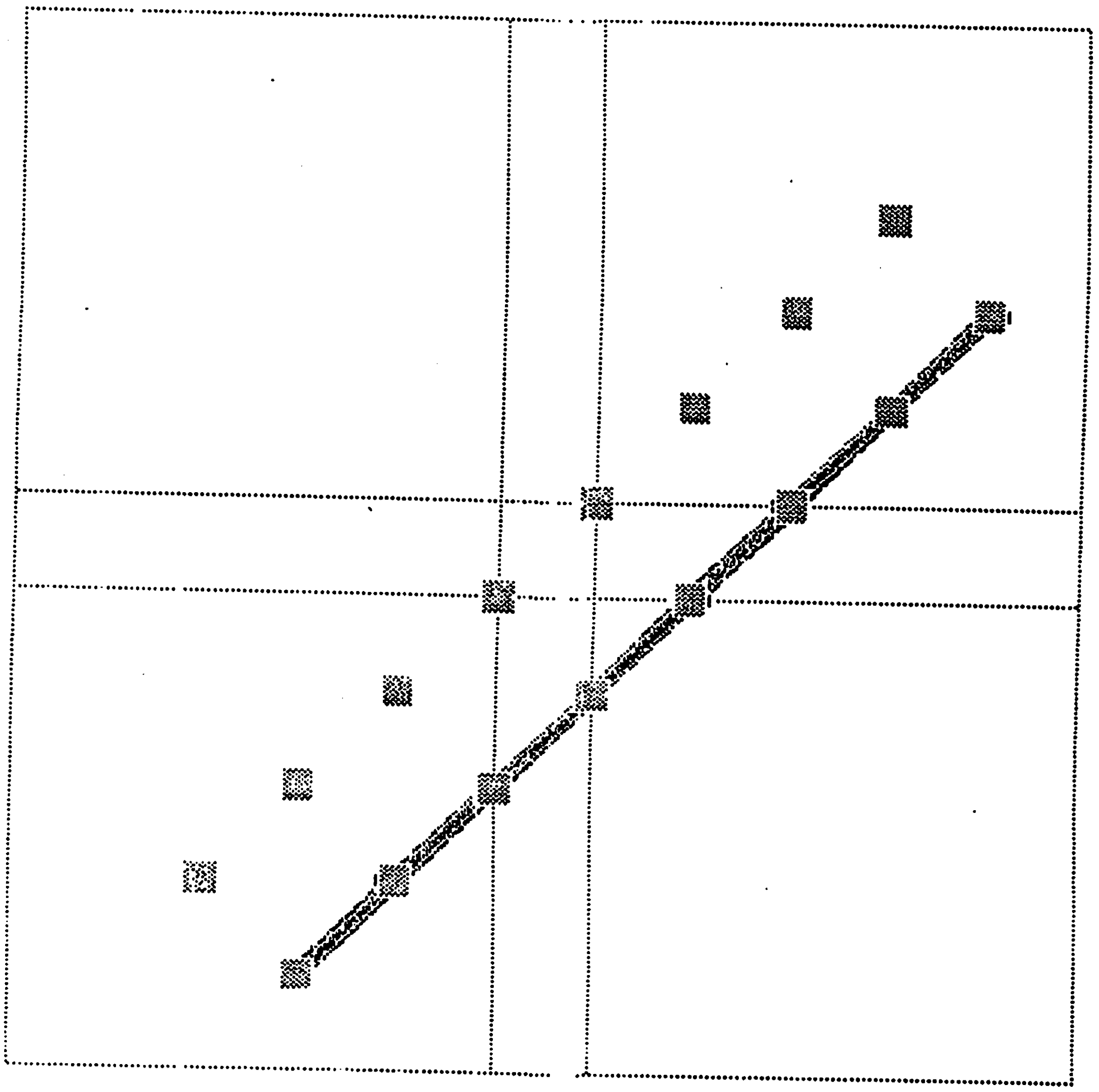


Figure 35: Straight Vehicle Environment: Sensory data.

Both ends of the actual vehicle track (vehicle class 1) are sensed strongly with signal location hypothesis beliefs of 5000 (on a scale from zero to 10000), but the middle portion is sensed weakly with signal location beliefs of 2000. The parallel "ghost" track (also vehicle class 1) is sensed moderately with signal location beliefs of 4100, and the "ghost" track is close enough to the actual track to be connected with it. The sensory signal location hypotheses are indicated by squares, with the actual vehicle track indicated by lines connecting the signal hypotheses. In this environment, prediction from the strong ends of the actual track can help determine that the track continues through the weaker central portion.

Time	Position	Sensed Belief
1	(6, 2)	5000
2	(8, 4)	5000
3	(10, 6)	5000
4	(12, 8)	2000
5	(14,10)	2000
6	(16,12)	5000
7	(18,14)	5000
8	(20,16)	5000

The parallel ghost track (also vehicle class 1) is close enough to the actual track to be connected with it and its signal location hypotheses are sensed as:

Time	Position	Sensed Belief
1	(4, 4)	4100
2	(6, 6)	4100
3	(8, 8)	4100
4	(10,10)	4100
5	(12,12)	4100
6	(14,14)	4100
7	(16,16)	4100
8	(18,18)	4100

In this environment, predicting from the strong ends of the actual track that the track continues through the weaker central region can significantly reduce the effort needed to determine the track.

5.1.5 The bent vehicle environment.

In the bent vehicle environment, the actual vehicle track bends to such a degree that predicting that the track continues in a straight line can falsely lead the system to use the ghost track (Figure 36). Again both ends of the actual vehicle track are sensed strongly and the middle portion (where the change in direction occurs) is sensed weakly. The signal location hypotheses associated with the actual vehicle (of

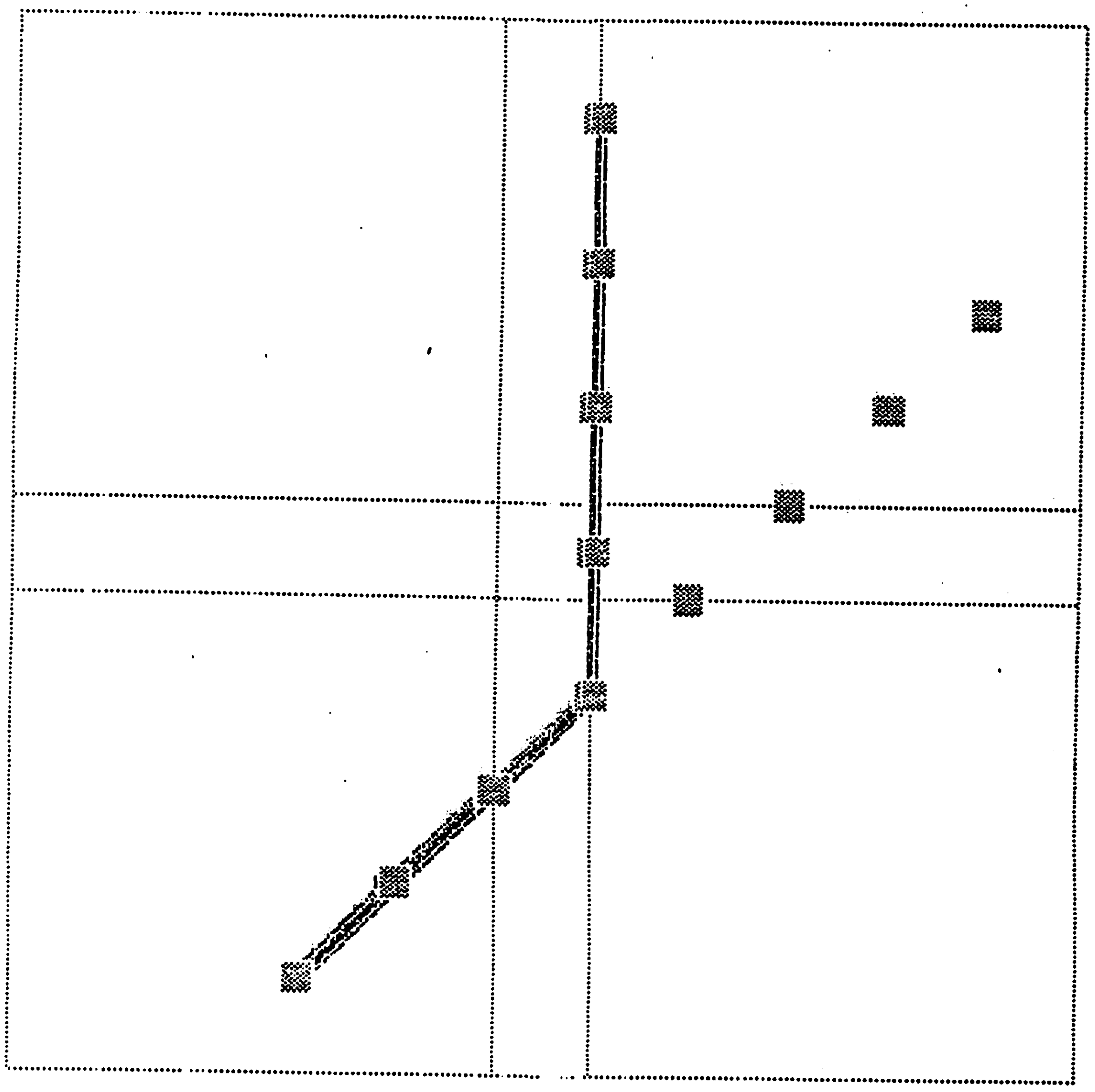


Figure 36: Bent Vehicle Environment: Sensory Data.

Both ends of the actual vehicle track (vehicle class 1) are sensed strongly with signal location hypothesis beliefs of 5000 (on a scale from zero to 10000), but the middle portion is sensed weakly with signal location beliefs of 2000. The ghost track (also vehicle class 1) extending in the original direction of the actual track is sensed moderately with signal location beliefs of 4100. In this environment, prediction from the actual track can falsely direct activity to the ghost track.

class 1) are sensed as:

Time	Position	Sensed Belief
1	(6, 2)	5000
2	(8, 4)	5000
3	(10, 6)	5000
4	(12, 8)	2000
5	(12,11)	2000
6	(12,14)	5000
7	(12,17)	5000
8	(12,20)	5000

In this environment, the moderate ghost track continues in the original direction of the actual vehicle track. Its signal location hypotheses are sensed as:

Time	Position	Sensed Belief
5	(14,10)	4100
6	(16,12)	4100
7	(18,14)	4100
8	(20,16)	4100

The purpose of this environment is to investigate the dangers of relying too strongly on prediction.

5.2 Experiments with the Goal-Directed Hearsay-II Architecture

This section describes experiments performed on a single node (centralized) testbed network configuration. The purpose of these experiments is to experiment with the goal-directed Hearsay-II architecture separately from the issues of a distributed network. In these experiments all four sensors report to the single node which generates the answer map. A table summarizing the results of these experiments is presented in Section 5.2.7.

5.2.1 Local coordination strategies.

In these experiments the local node planner is instructed (via the subgoaling specifications[2] contained in the environment file) to generate subgoals from vehicle track and pattern track goals created from internal data-directed events. Vehicle track goals have subgoals generated at the vehicle location, group location, and signal location levels and pattern track goals have subgoals generated at the vehicle track, vehicle location, group location, and signal location levels. These subgoals are given the same rating and cover the same time frames and blackboard regions as the high level goal.

Two different local coordination strategies were explored by changing (again in the environment file) the subgoaling threshold used by the planner. In the first strategy the threshold is set at 10000, indicating that only those vehicle track and pattern track goals rated at 10000 should be subgoaled. This setting effectively eliminates the generation of any subgoals and knowledge sources are scheduled solely on the basis of data-directed events.

In the second strategy the threshold is set at zero, indicating that any vehicle track or pattern track goal with a positive value should be subgoaled. This setting effectively causes the generation of subgoals from all vehicle track and pattern track goals. The higher ratings of these subgoals, stemming from the general increase in the beliefs of hypotheses at higher abstraction levels, causes the ratings of knowledge source instantiations contributing to vehicle track or

2. See Section 4.2.7.

pattern track goals to be increased.

In most of the experiments reported in this chapter, a goal weighting of 8000 is used in the knowledge source instantiation rating calculation. (Experiments in which the goal weighting is varied are discussed in Section 5.2.6.) A goal weighting of zero specifies that the predicted beliefs of the hypotheses generated by the knowledge source instantiation are used for scheduling. A weighting of 10000 specifies that the rating of the goals the knowledge source instantiation is attempting to achieve is used for scheduling. The 8000 weighting specifies that 20 percent of the former and 80 percent of the latter be used in rating knowledge source instantiations (see Section 4.2.9).

5.2.2 Straight vehicle environment.

Without focusing through the creation of subgoals, the system executes 157 knowledge source instantiations (excluding the FRONTEND and SENSORS) to completely generate the correct track (Figures 37 through 42). With focusing based on subgoaling from the vehicle track and pattern track levels the system executes 52 knowledge source instantiations (Figures 43 through 45).

This significant difference comes from the system performing the complete generation of the actual pattern track hypothesis (at which point it is stopped) before the expenditure of considerable effort in attempting to develop track hypotheses that integrate high belief data from the actual track with medium belief false data and on the medium belief false data. Without subgoaling, these activities seem reasonable

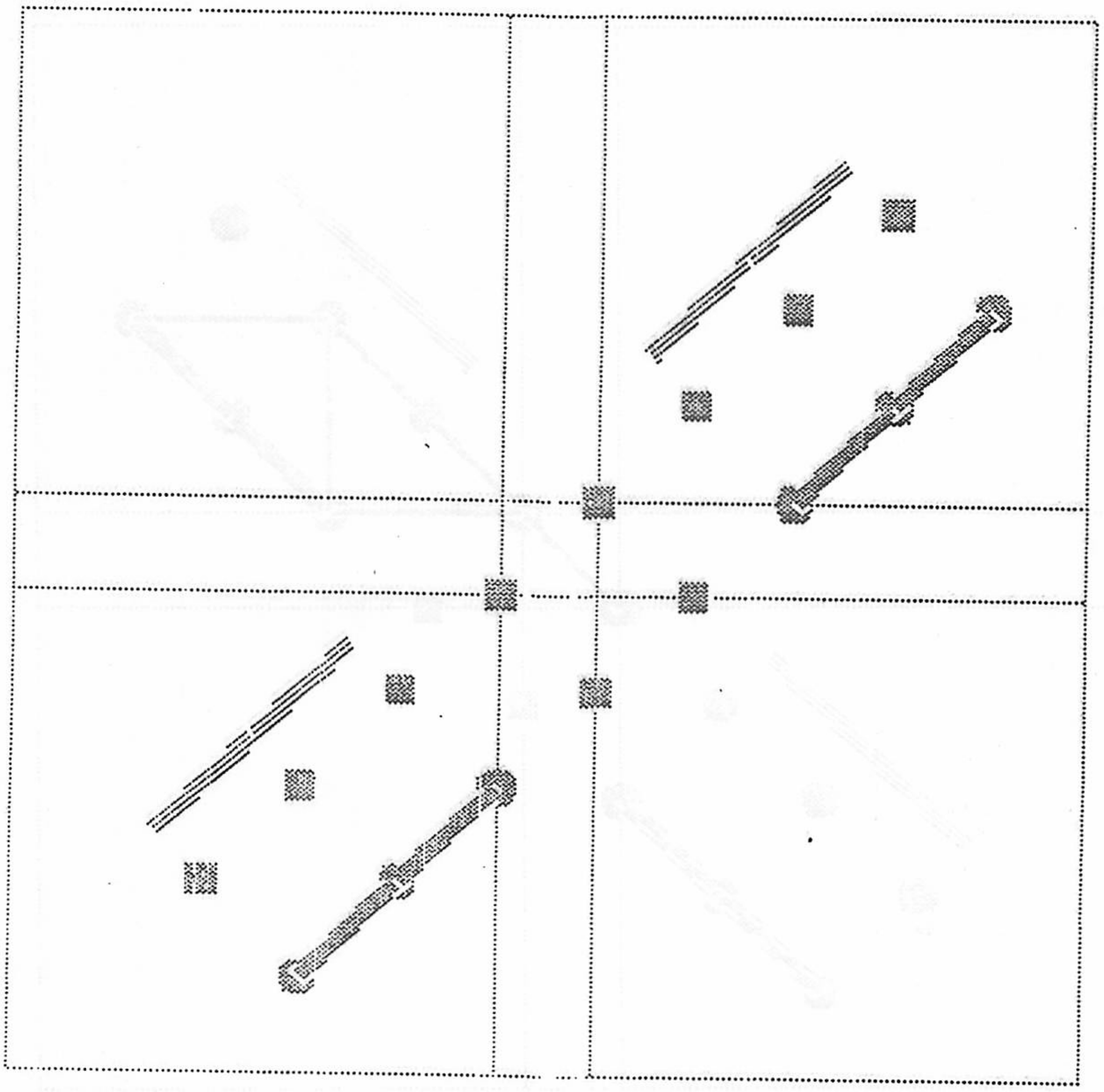


Figure 37: Straight Vehicle Environment without Subgoaling: Cycle 30.

In the first 30 knowledge source executions, the system first synthesizes vehicle location hypotheses from the strongly sensed portions of the actual vehicle track and then forms pattern track hypotheses out of these hypotheses. (The lighter tracks to the upper left are weakly hypothesized pattern class 3 hypotheses generated from the single support of vehicle type 1.)

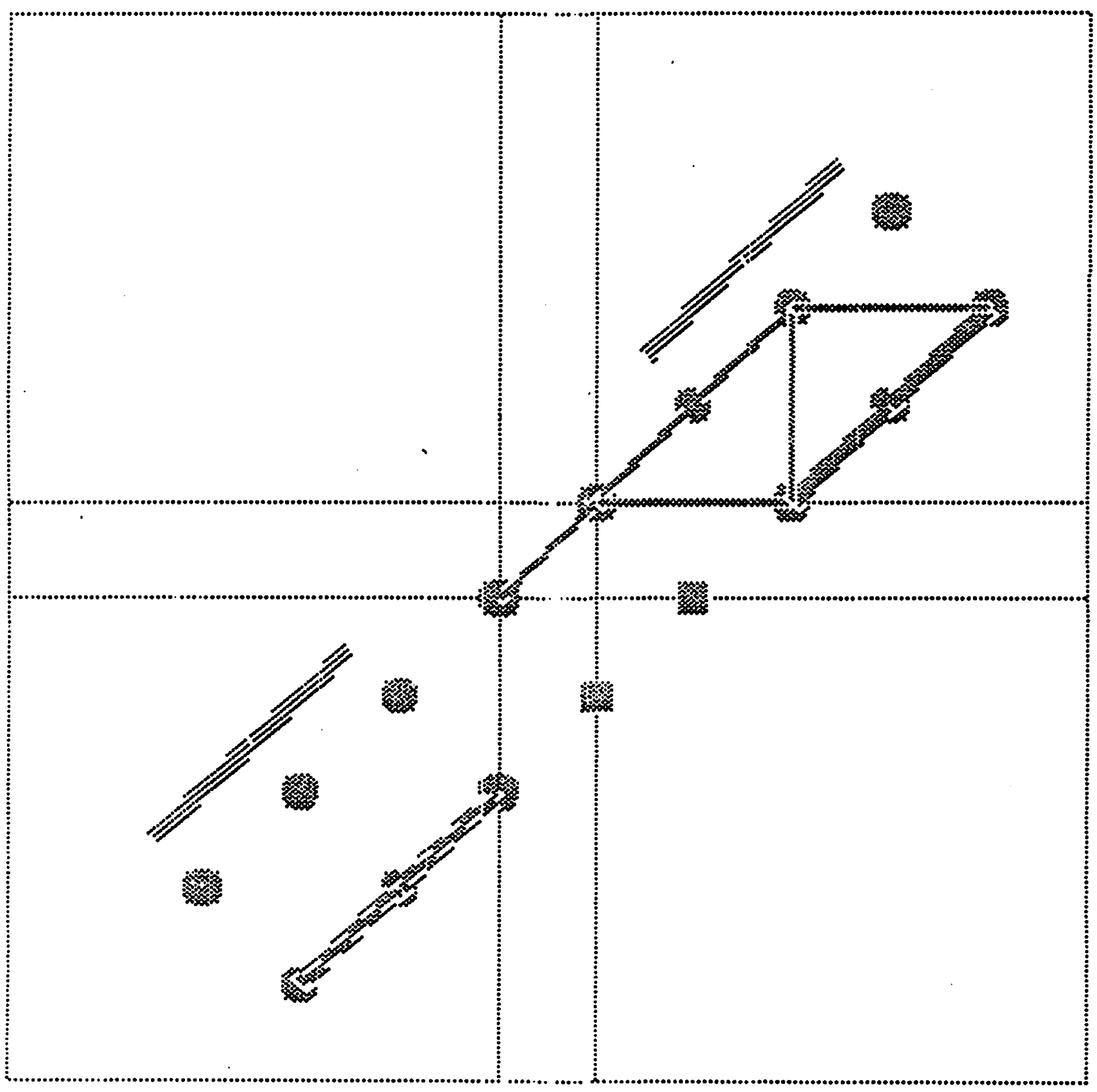


Figure 38: Straight Vehicle Environment without Subgoaling: Cycle 61.

The system next synthesizes all vehicle location hypotheses of the ghost track and uses these in forming vehicle track hypotheses with some of the strongly sensed portions of the actual vehicle track. (The narrow tracks represent vehicle track hypotheses and the wider tracks pattern track hypotheses.)

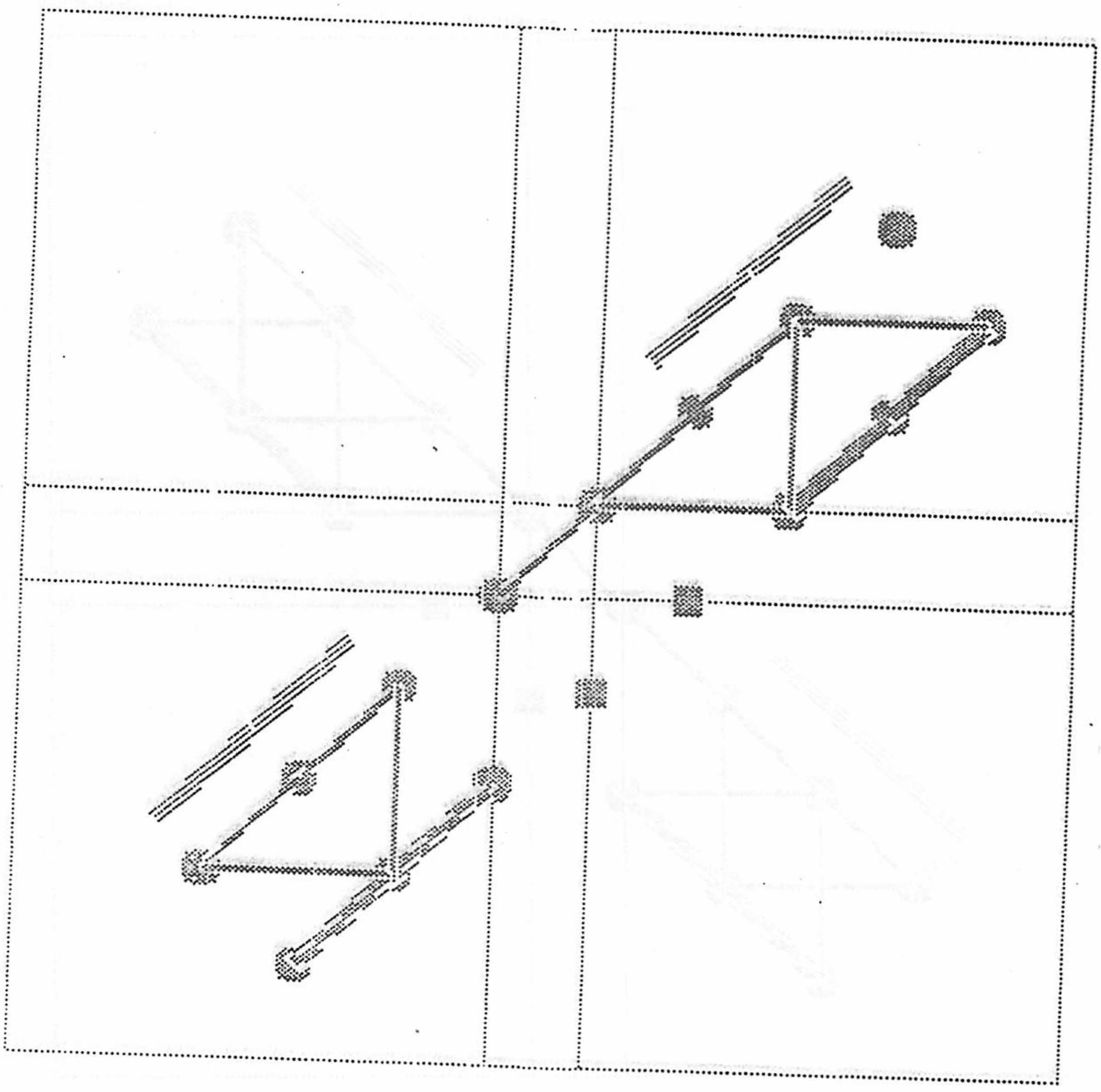


Figure 39: Straight Vehicle Environment without Subgoaling: Cycle 62.

The next knowledge source instantiation continues forming vehicle track hypotheses with other strongly sensed portions of the actual vehicle track.

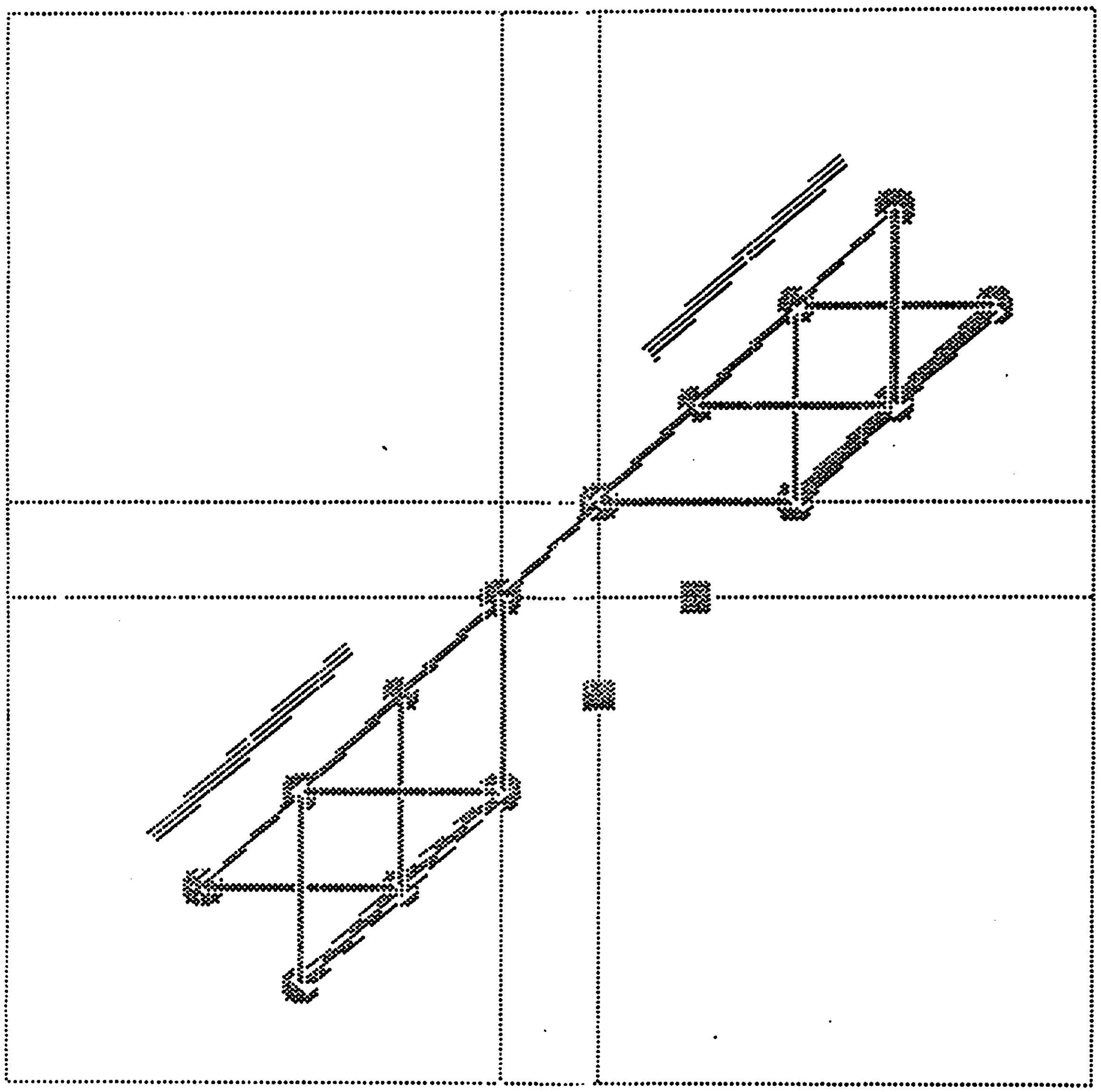


Figure 40: Straight Vehicle Environment without Subgoaling: Cycle 63.

The next knowledge source instantiation completes formation of vehicle track hypotheses of the ghost track. Although the figure makes it appear as if there is a single vehicle track hypothesis representing the entire ghost track, there are actually a number of shorter hypotheses which have not yet been merged together.

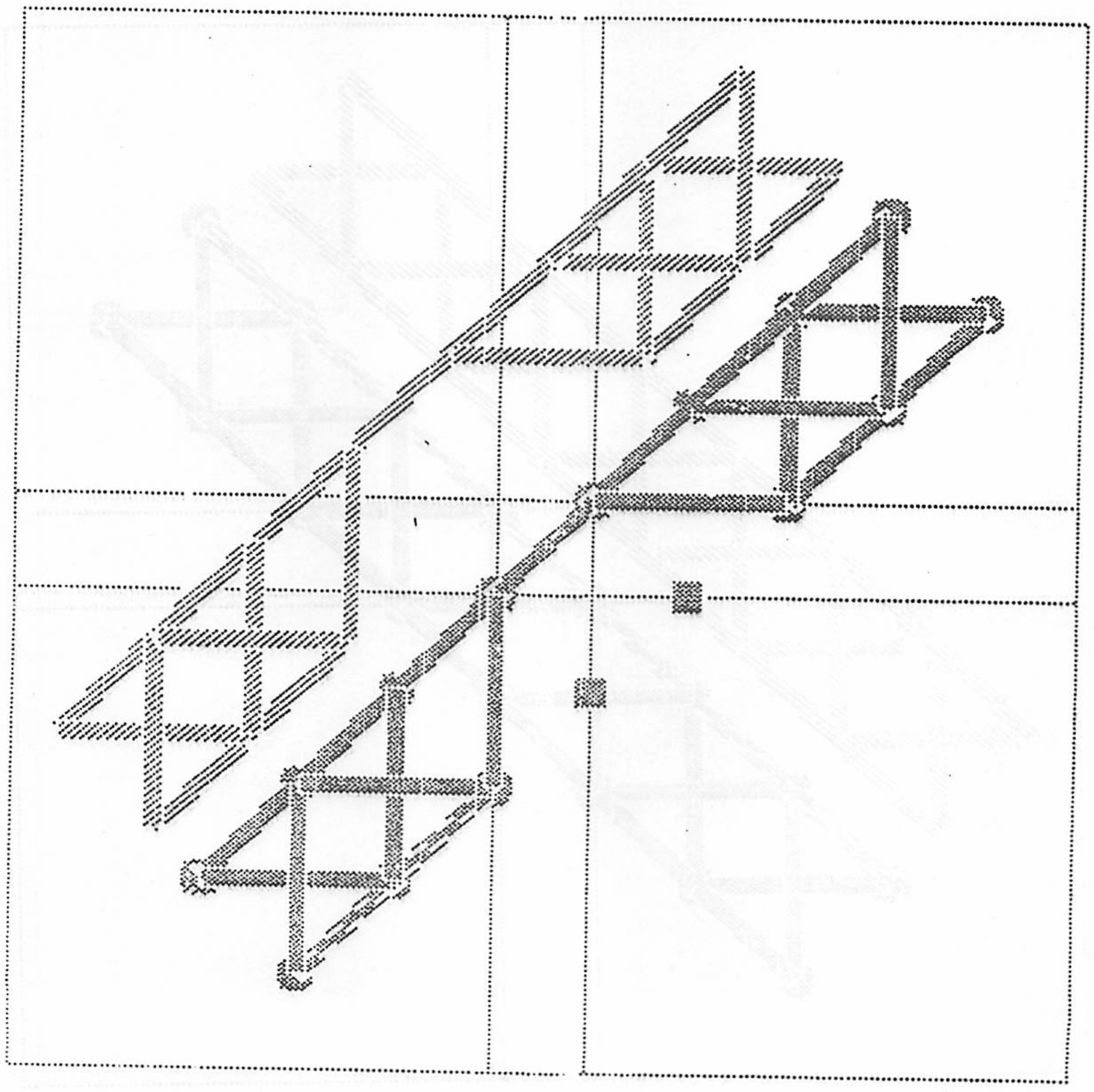


Figure 41: Straight Vehicle Environment without Subgoaling: Cycle 73.

Because the central portion of the actual track is believed so weakly, the next 10 knowledge source instantiations are spent forming pattern track hypotheses from the ghost vehicle track hypotheses.

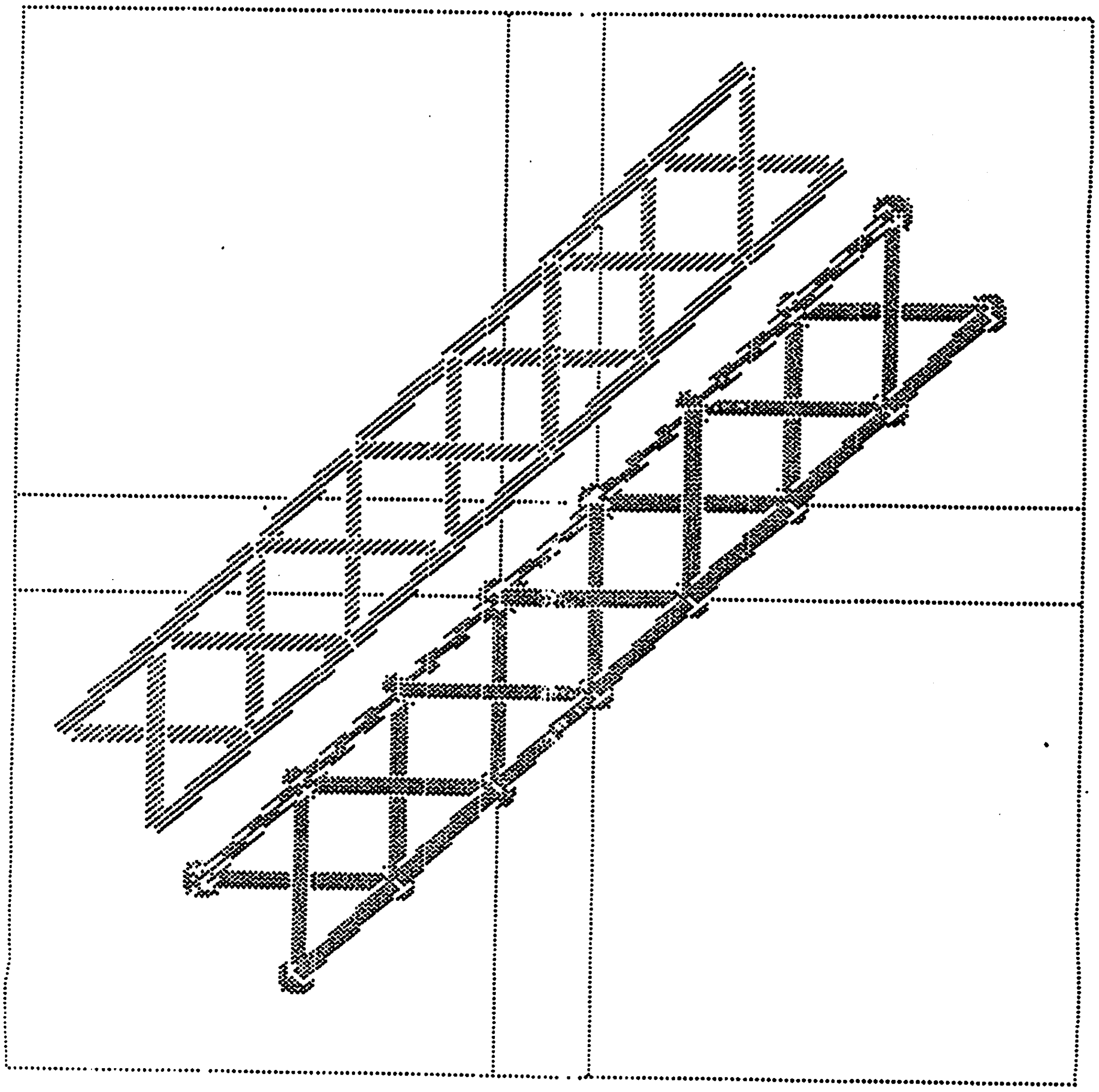


Figure 42: Straight Vehicle Environment without Subgoaling: Solution at Cycle 157.

The weaker central portion of the actual track is finally worked on and the complete vehicle track and pattern track hypotheses for the actual track are generated. The complete actual pattern track hypothesis is generated in cycle 157.

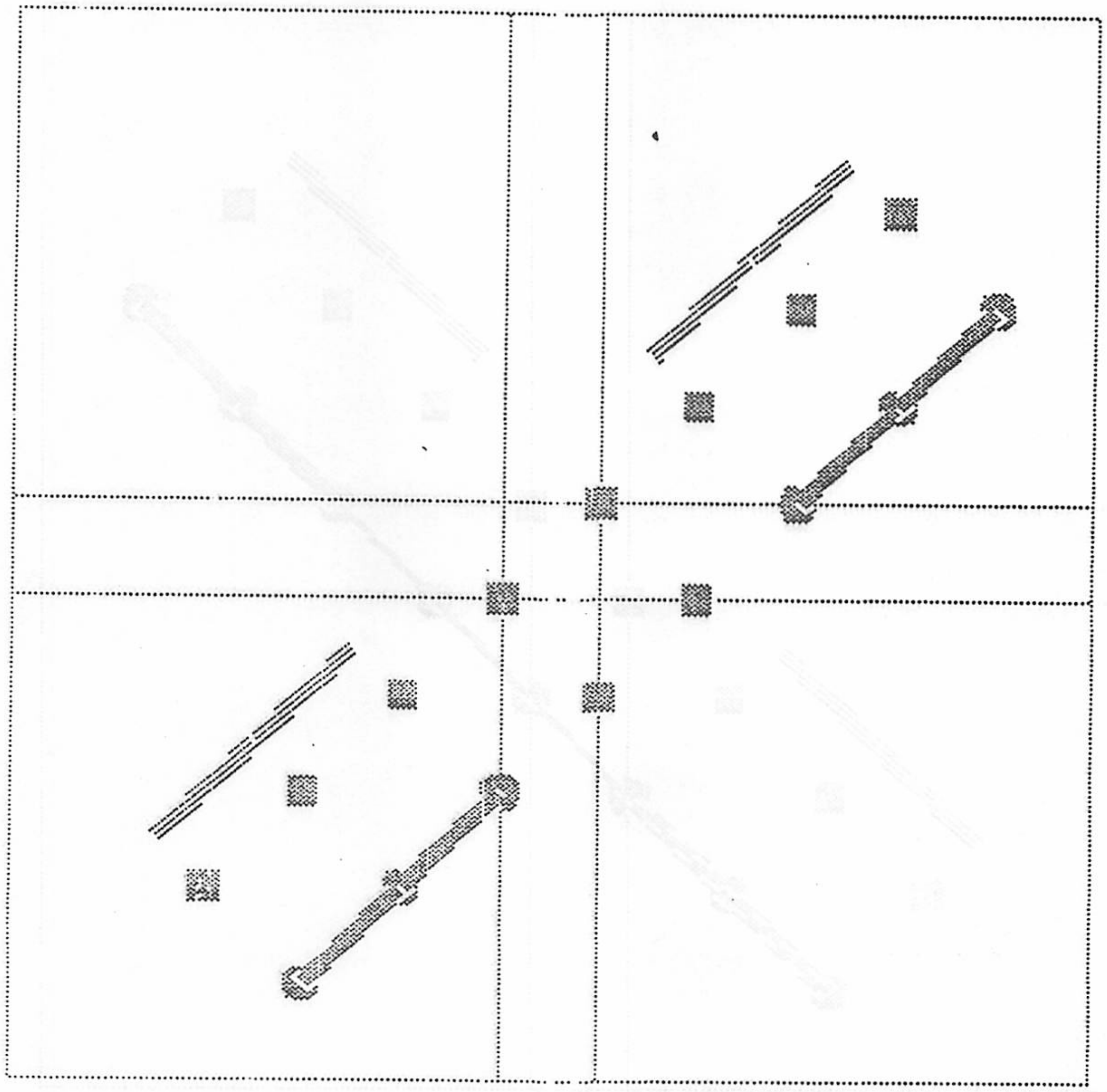


Figure 43: Straight Vehicle Environment with Subgoaling: Cycle 23.

In the first 23 knowledge source executions, the system synthesizes vehicle location hypotheses from the strongly sensed portions of the actual vehicle track and then forms pattern track hypotheses out of these hypotheses.

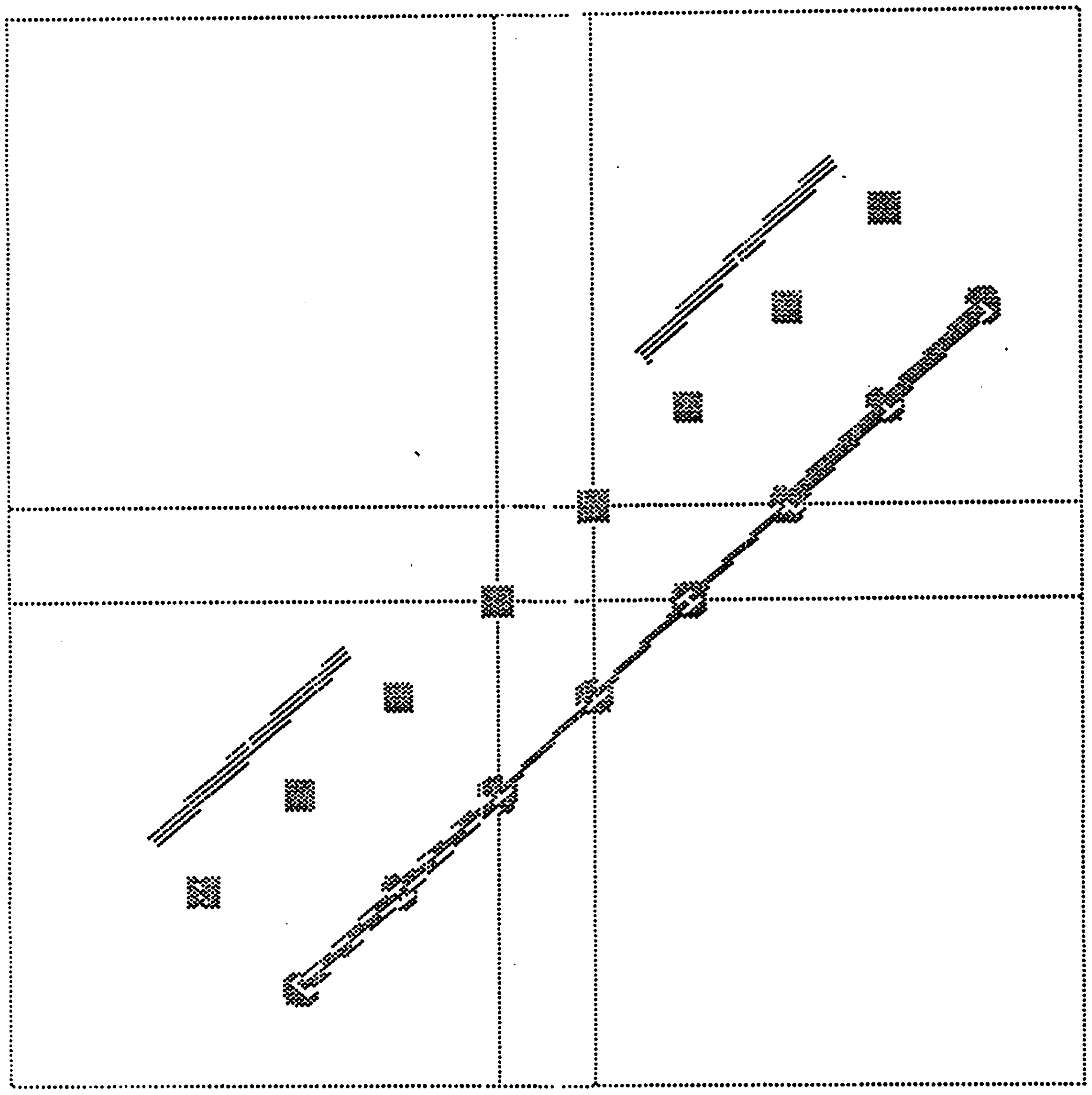


Figure 44: Straight Vehicle Environment with Subgoaling: Cycle 38.

Subgoals generated to extend these pattern track hypotheses (both forward and backward in time) raise the ratings on knowledge source instantiations working on the weaker portions of the actual track and the system next synthesizes vehicle location hypotheses extending through the weaker central portion of the actual track.

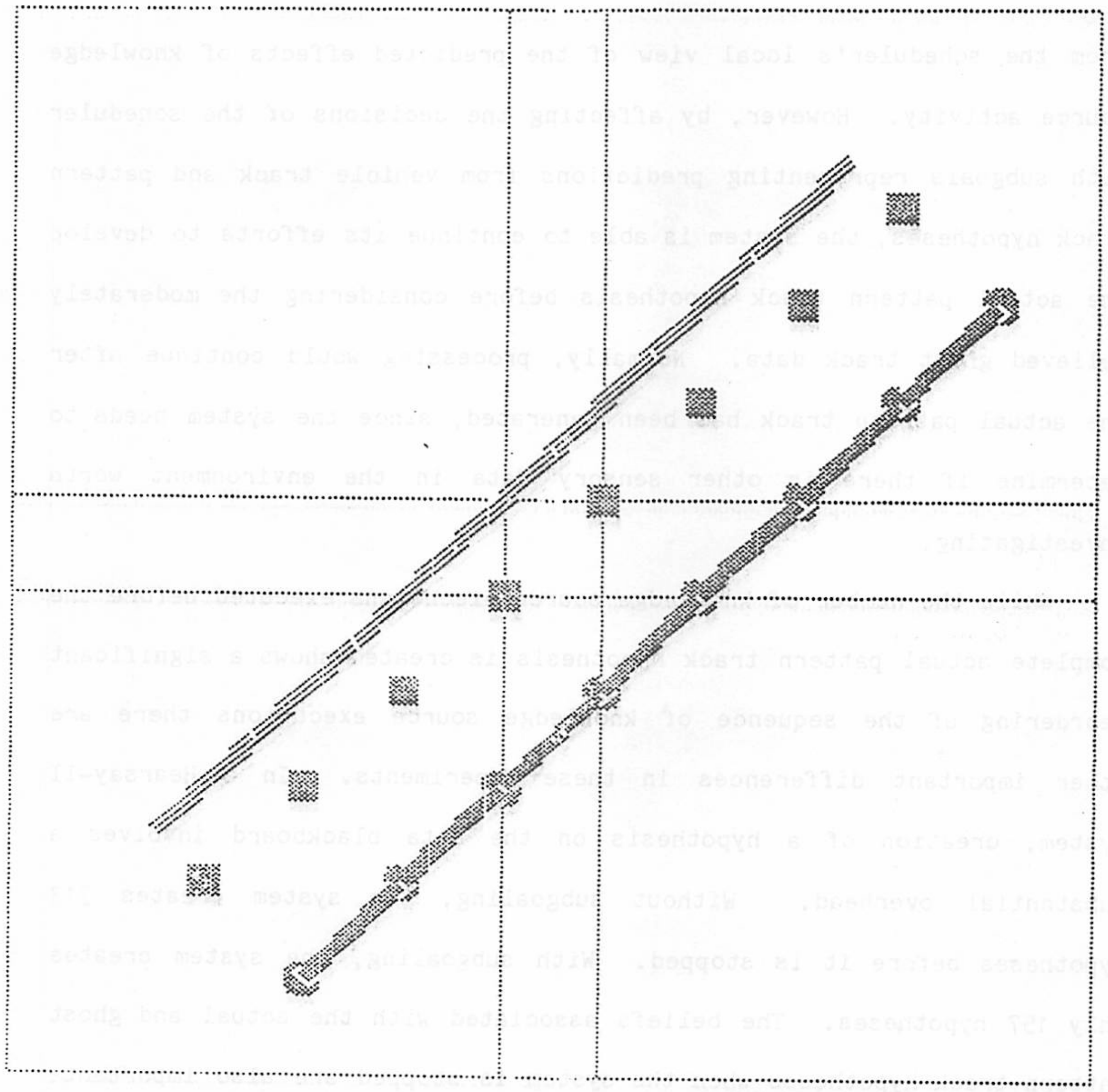


Figure 45: Straight Vehicle Environment with Subgoaling: Solution at Cycle 52.

The remaining knowledge source instantiations merge the vehicle track and pattern track hypotheses together, eventually generating the complete actual pattern track hypothesis. The use of subgoaling has changed the ordering of knowledge source instantiations to such a degree that the actual pattern track hypothesis is generated before any work at all is performed on the ghost track. (Normally, processing would be allowed to continue beyond this cycle, since the system needs to determine if there is other sensory data in the environment worth investigating.)

from the scheduler's local view of the predicted effects of knowledge source activity. However, by affecting the decisions of the scheduler with subgoals representing predictions from vehicle track and pattern track hypotheses, the system is able to continue its efforts to develop the actual pattern track hypothesis before considering the moderately believed ghost track data. Normally, processing would continue after the actual pattern track has been generated, since the system needs to determine if there is other sensory data in the environment worth investigating.

While the number of knowledge source executions executed before the complete actual pattern track hypothesis is created shows a significant reordering of the sequence of knowledge source executions there are other important differences in these experiments. In a Hearsay-II system, creation of a hypothesis on the data blackboard involves a substantial overhead. Without subgoaling, the system creates 313 hypotheses before it is stopped. With subgoaling, the system creates only 157 hypotheses. The beliefs associated with the actual and ghost pattern track hypotheses when the system is stopped are also important. Without subgoaling, the system places a belief of 4499 on the actual pattern track hypothesis and a belief of 4098 on the ghost pattern track hypothesis. With subgoaling, the system places a belief of 4666 on the actual pattern track hypothesis and is stopped before it creates a ghost pattern track hypothesis.

To help place these numbers in perspective, the experiments were rerun to exhaustion by setting the solution threshold to 10000 (an unattainable belief). Without subgoaling the complete generation of all hypotheses from the sensory data requires 323 cycles and the system creates 552 hypotheses, places a belief of 4952 on the actual pattern track hypothesis, and places a belief of 4098 on the ghost pattern track hypothesis. With subgoaling the complete generation of all hypotheses from the sensory data requires 322 cycles and the system creates 572 hypotheses, places a belief of 4952 on the actual pattern track hypothesis, and places a belief of 4100 on the ghost pattern track hypothesis. The negligible differences between these two experiments are due to asymmetries in the simple knowledge used in the knowledge sources' candidate generators: a knowledge source may generate a slightly different belief for a created hypothesis if the supports are added by multiple invocations of the knowledge source than if all the supports are available in a single invocation.

5.2.3 Bent vehicle environment.

The bent vehicle environment is designed to thwart subgoal focusing through the prediction of uniform vehicle movement. Instead of continuing in the same direction, the actual track makes a sharp bend in the middle of the weakly sensed area. To make matters worse, a ghost track extends in the original direction of the actual track.

Without focusing through the creation of subgoals, the system executes 116 knowledge source instantiations to completely generate the correct track, creates 258 hypotheses, places a belief of 4912 on the actual pattern track hypothesis, and places a belief of 4752 on the ghost pattern track hypothesis (Figures 46 through 52). With focusing based on subgoaling from the vehicle track level the system executes 65

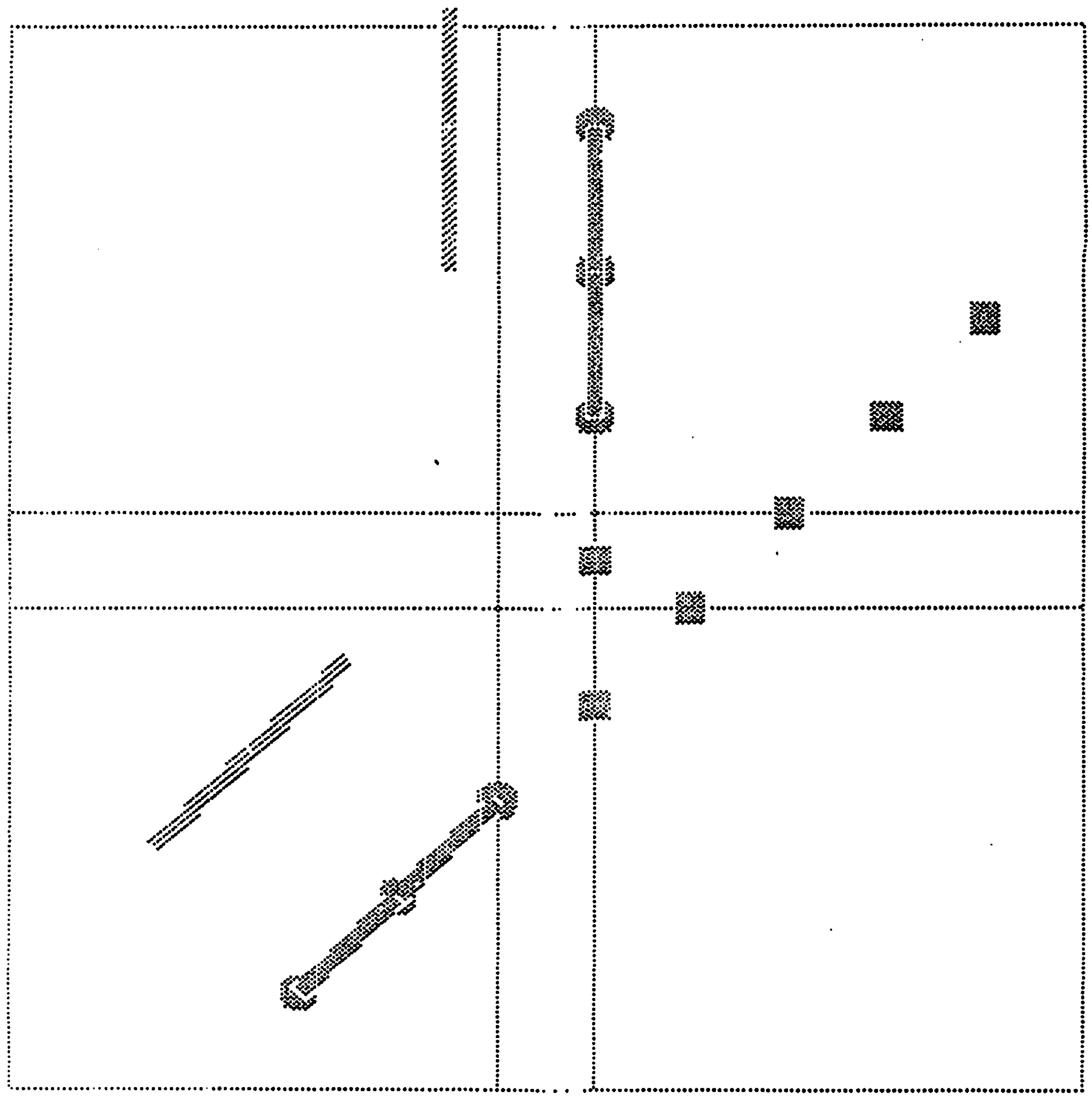


Figure 46: Bent Vehicle Environment without Subgoaling: Cycle 28.

In the first 28 knowledge source executions, the system synthesizes pattern track hypotheses from the strongly sensed portions of the actual track.

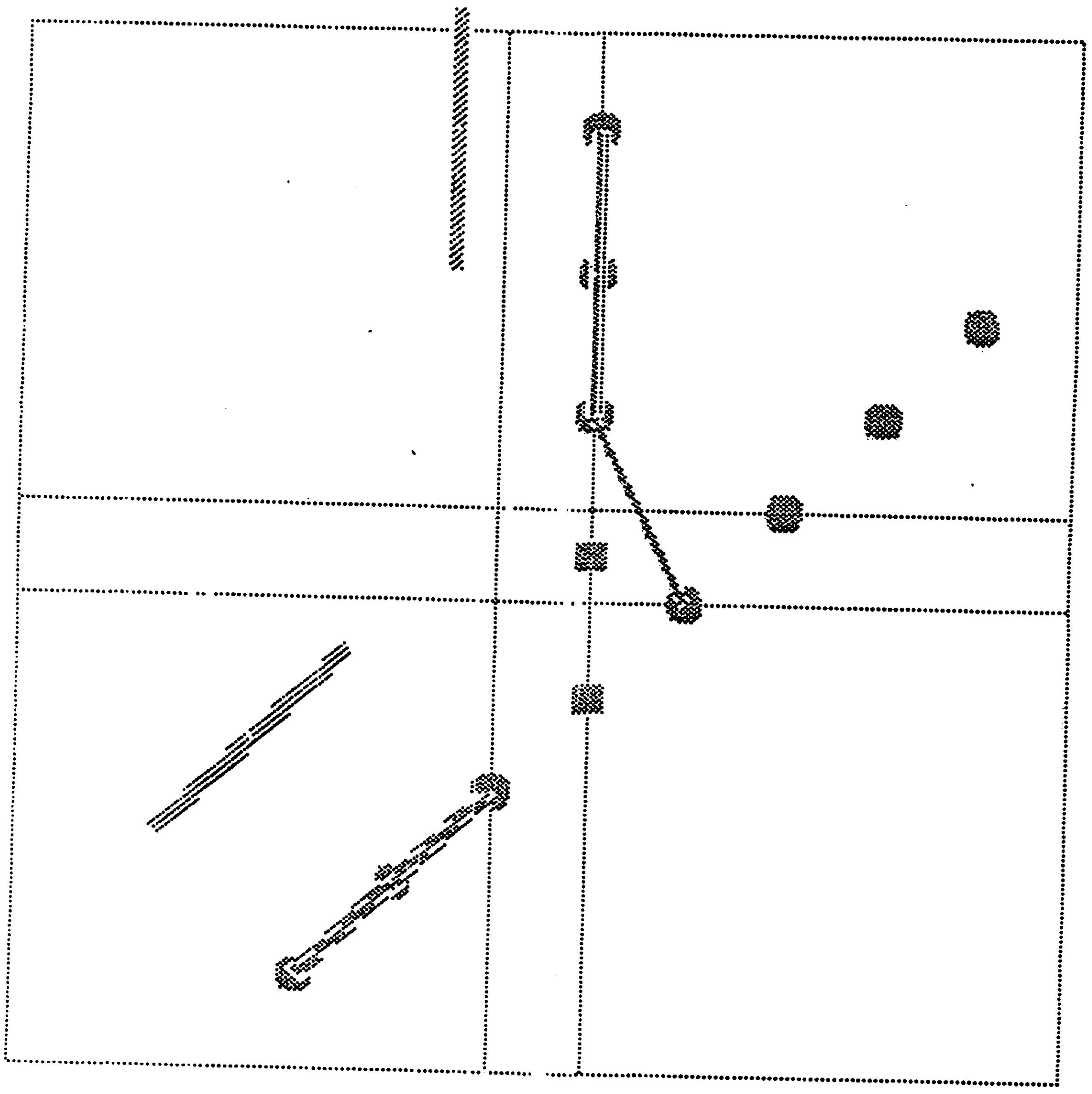


Figure 47: Bent Vehicle Environment without Subgoaling: Cycle 45.

The system next synthesizes all vehicle location hypotheses of the ghost extension and forms a vehicle track hypothesis from the first ghost location to the strong end of the actual track.

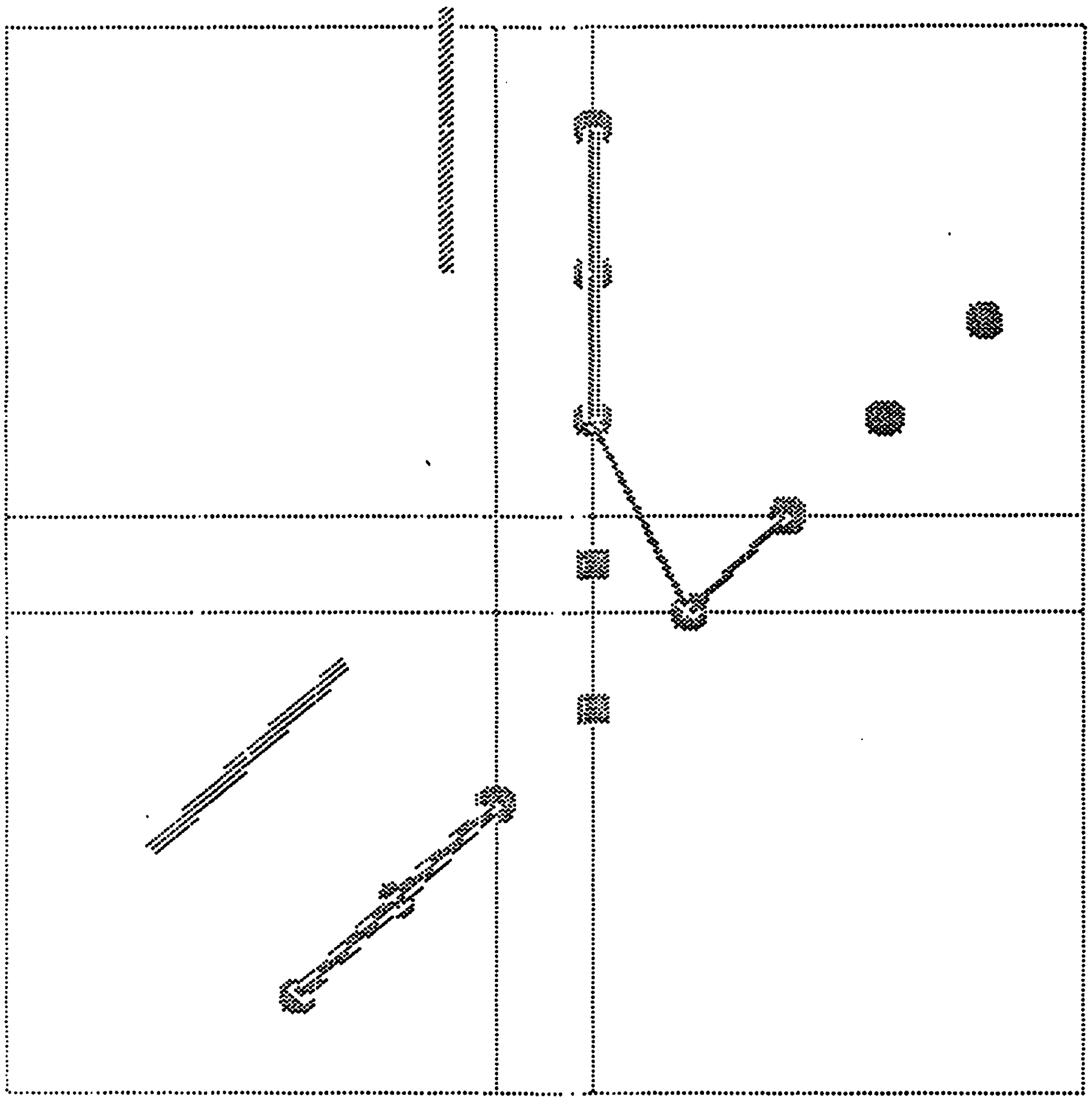


Figure 48: Bent Vehicle Environment without Subgoaling: Cycle 46.

The system next forms an additional vehicle track hypothesis using the ghost vehicle location hypotheses.

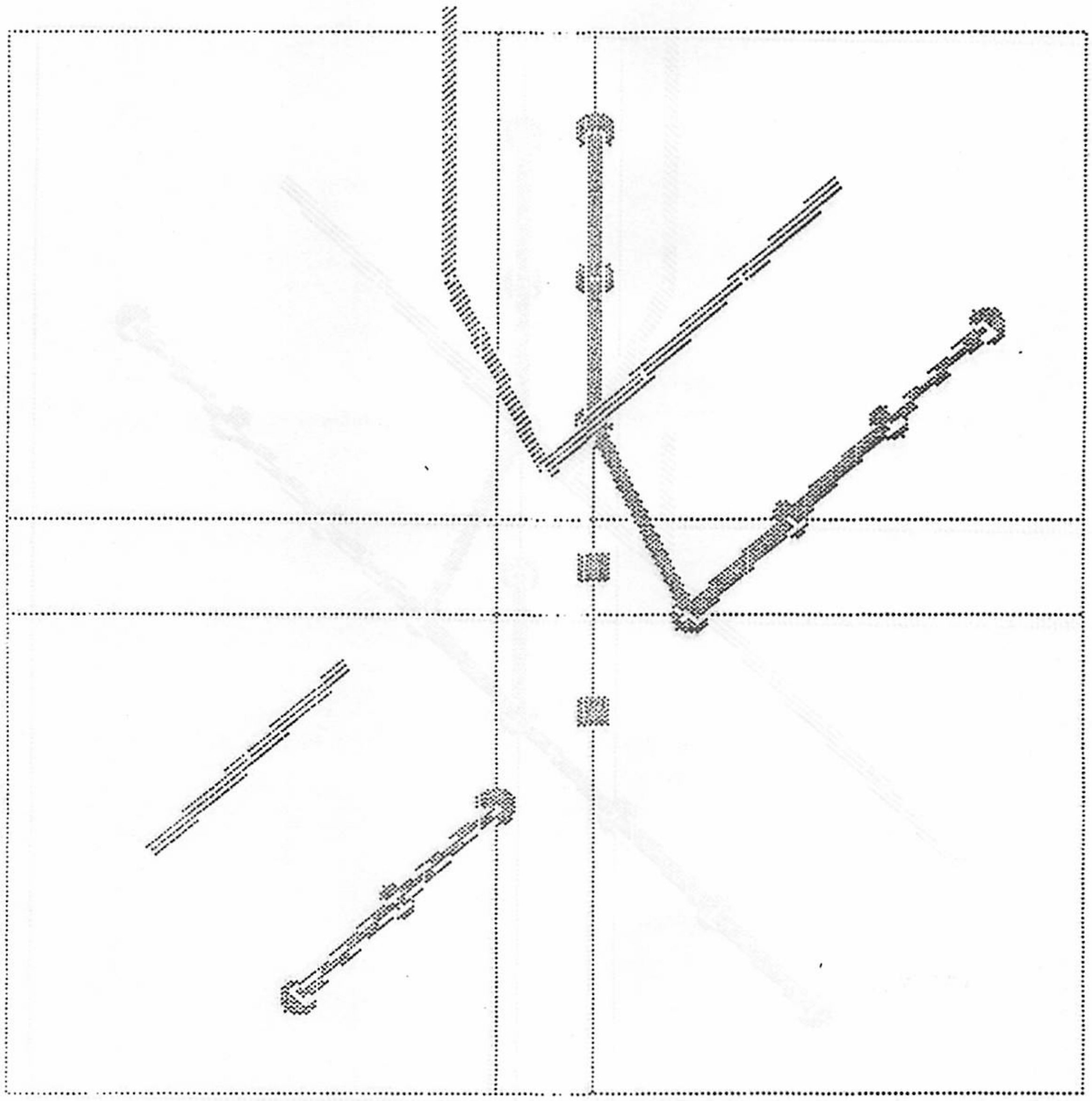


Figure 49: Bent Vehicle Environment without Subgoaling: Cycle 52.

The ghost extension is completed and pattern track hypotheses are synthesized from all vehicle track hypotheses.

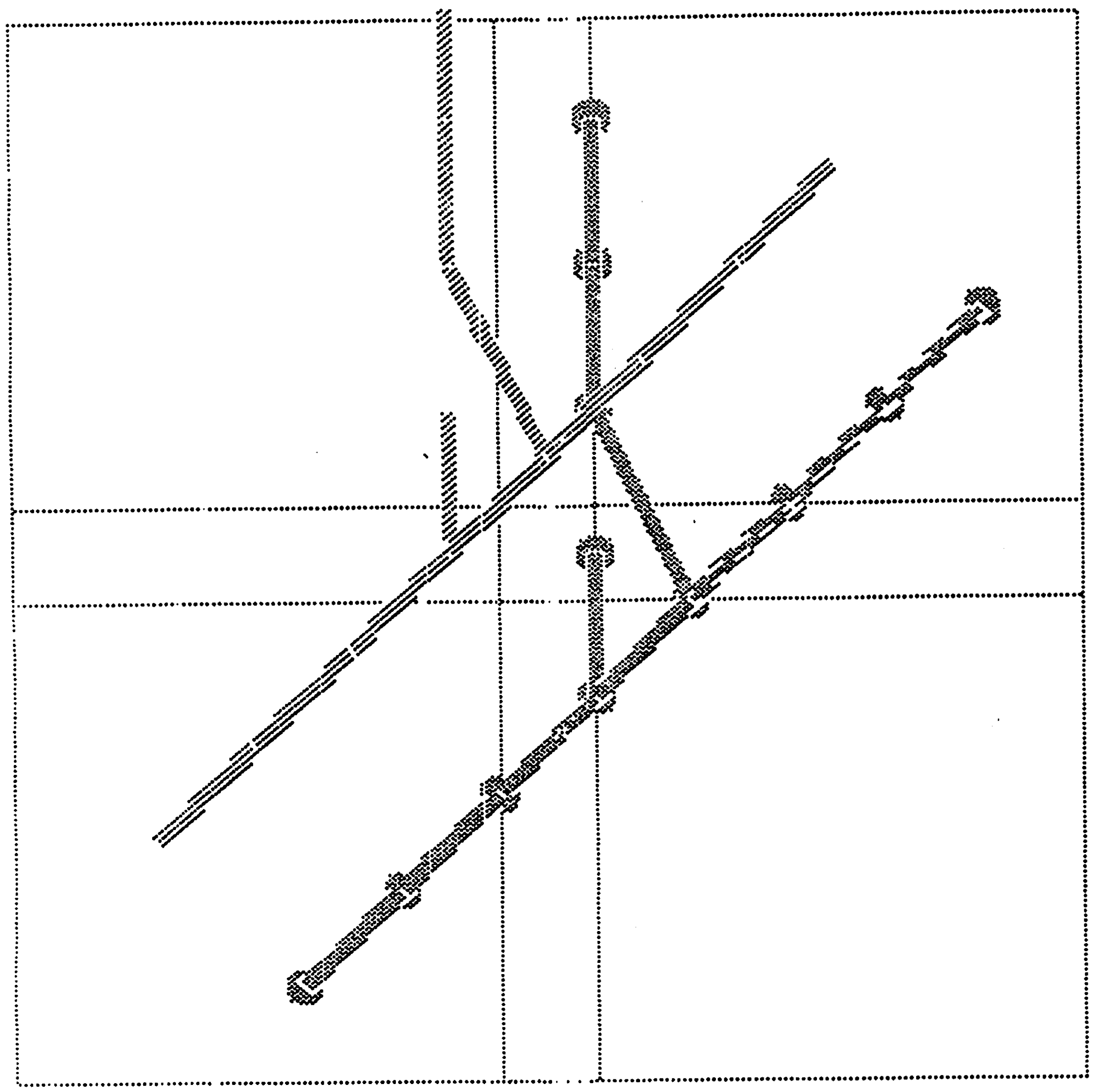


Figure 50: Bent Vehicle Environment without Subgoaling: Cycle 71.

The system finally begins to work in the weak central region, forming pattern track hypotheses from both the ghost extension and the actual track.

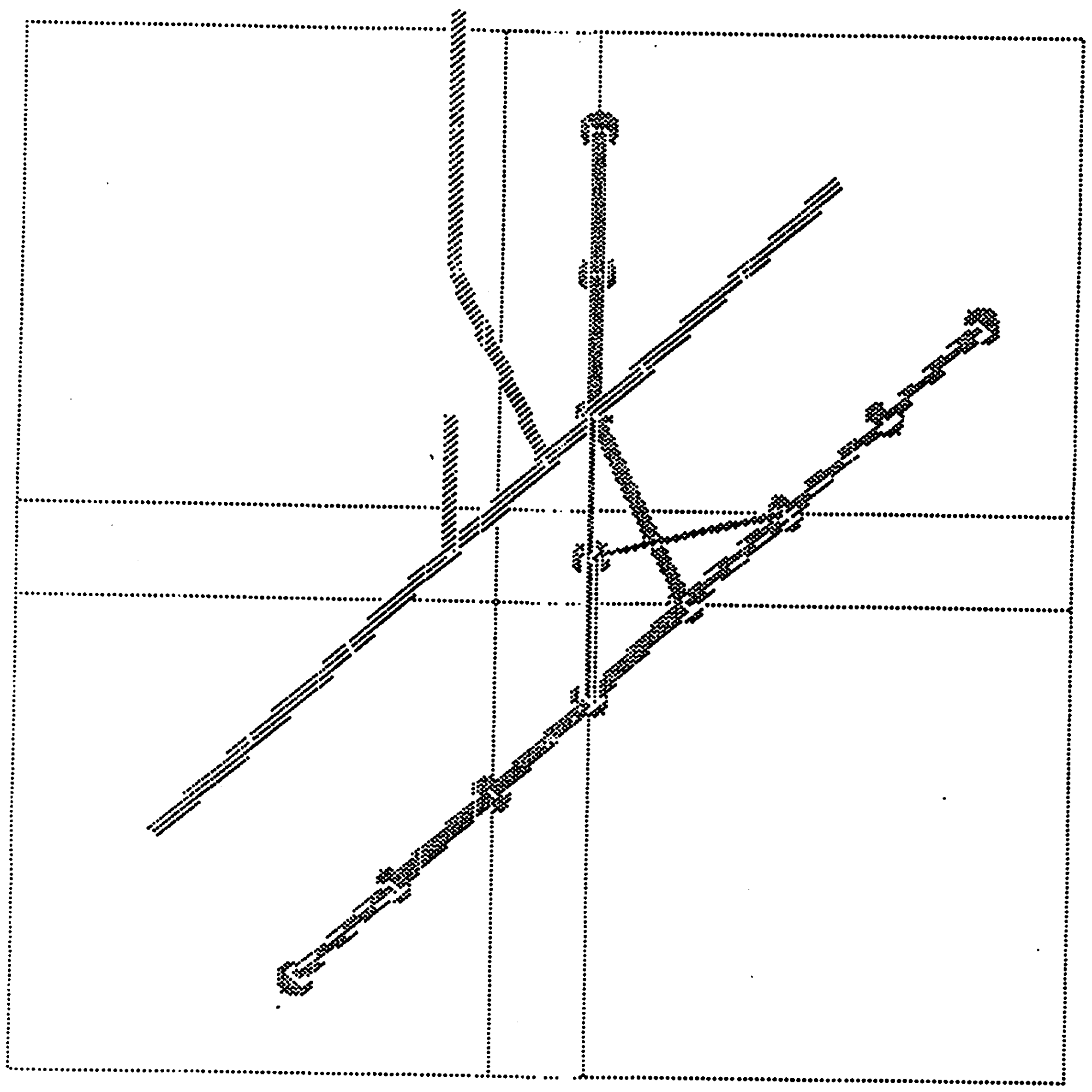


Figure 51: Bent Vehicle Environment without Subgoaling: Cycle 74.

Additional vehicle track hypotheses are made, including the last section of the actual track.

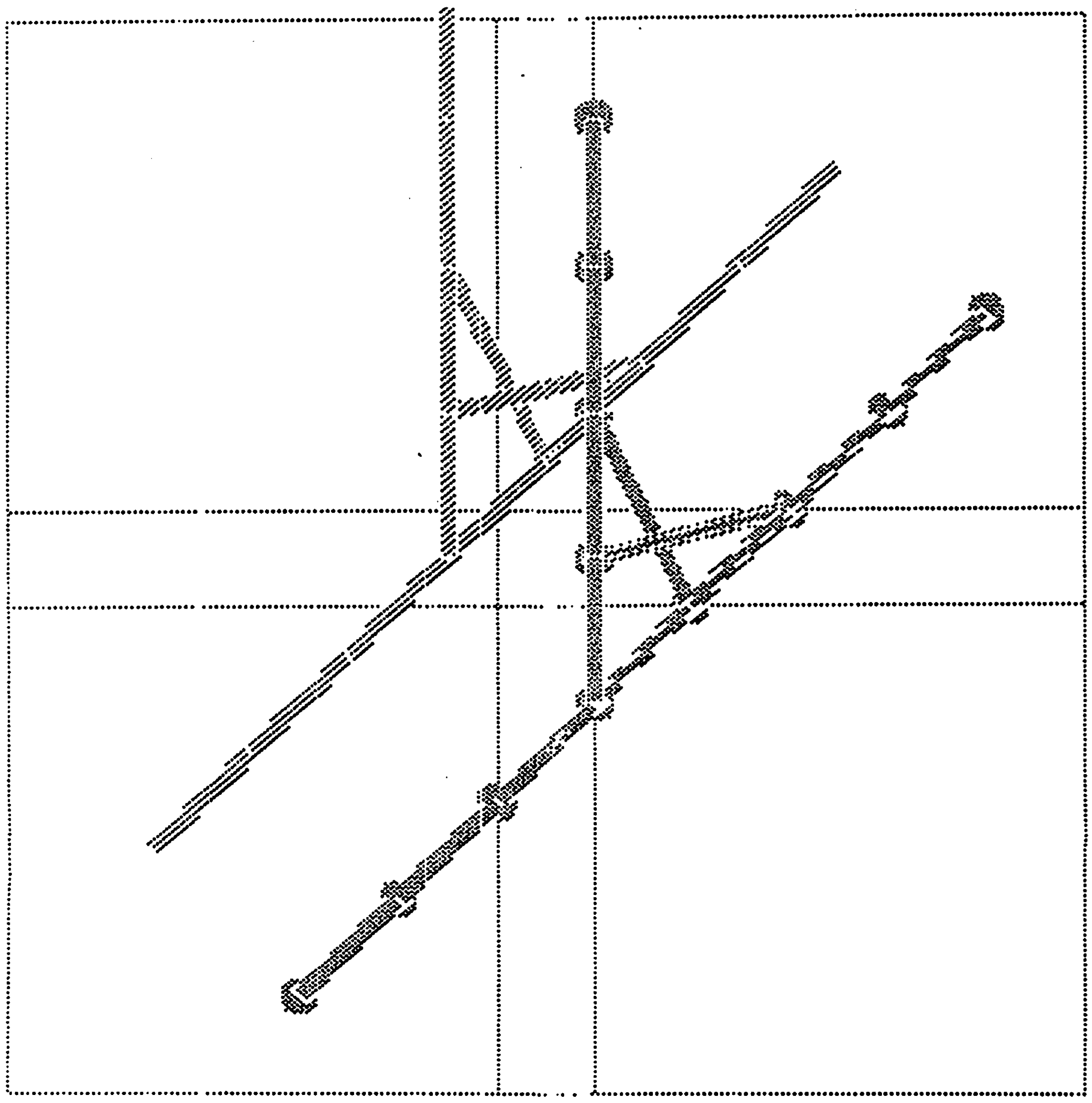


Figure 52: Bent Vehicle Environment without Subgoaling: Solution at Cycle 116.

Pattern track hypotheses are synthesized from the remaining vehicle track hypotheses and these hypotheses are merged into the complete actual pattern track hypothesis and the ghost pattern track extension.

knowledge source instantiations, creates 173 hypotheses, places a belief of 4416 on the actual pattern track hypothesis, and is stopped before creating the ghost pattern track hypothesis (Figures 53 through 56).

Despite the bend in the actual track, the use of subgoaling effectively changed the ordering of knowledge source instantiations to allow completion of the actual track before any work was begun on the ghost extension. This level of performance was better than expected. It was expected that subgoaling from predicted vehicle track and pattern track extension goals would lead the system to work on the ghost track extension, causing the performance of the system with and without subgoaling to be more similar. (But see the real-time experiments, reported in the next section).

Again these experiments were rerun to exhaustion by setting the solution threshold at 10000. Without subgoaling the system executes 184 knowledge source instantiations, creates 384 hypotheses, places a belief of 4912 on the actual pattern track hypothesis, and places a belief of 4752 on the ghost pattern track hypothesis. With subgoaling the system executes 164 knowledge source instantiations, creates 390 hypotheses, places a belief of 4416 on the actual pattern track hypothesis, and places a belief of 4304 on the ghost pattern track hypothesis. The difference in these results needs a bit of explanation.

Even though both of the exhaustive search experiments ran until there was no longer any work left to be performed, they searched the space of possible solutions very differently. Even when subgoaling is turned off, the planner in the goal-directed Hearsay-II architecture is using data-directed goals as local contexts for determining which

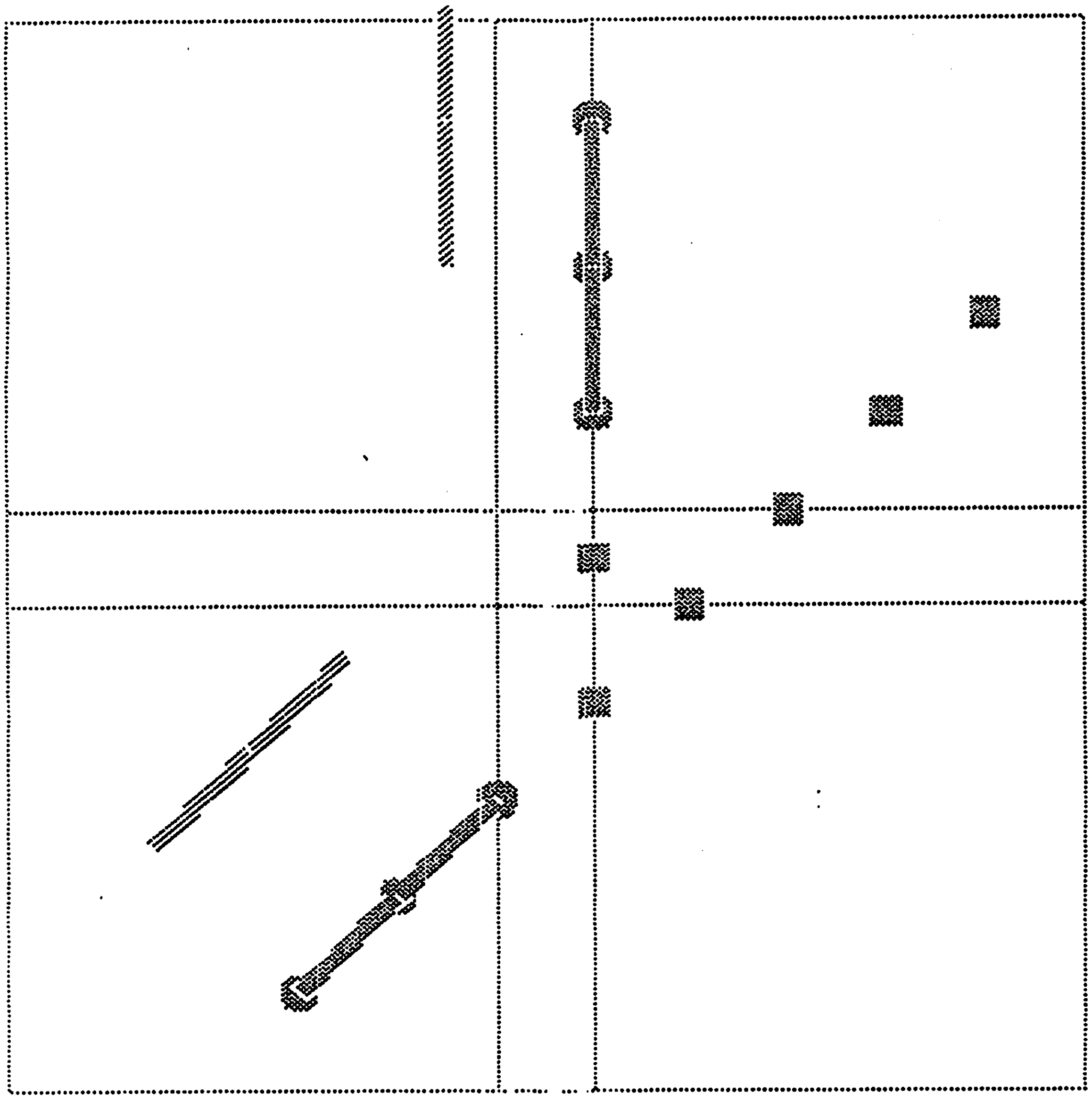


Figure 53: Bent Vehicle Environment with Subgoaling: Cycle 28.

In the first 28 knowledge source executions, the system synthesizes pattern track hypotheses from the strongly sensed portions of the actual track.

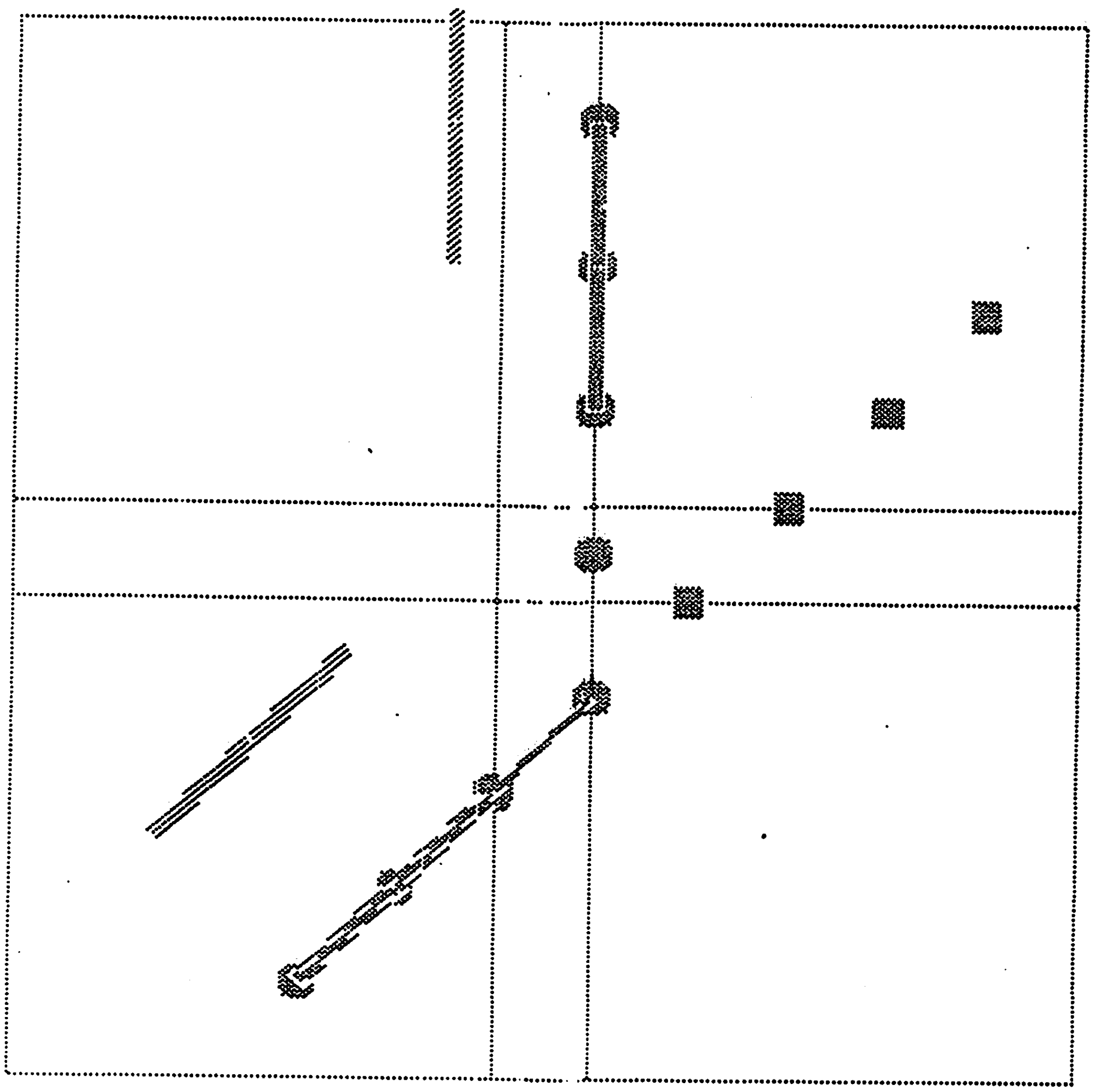


Figure 54: Bent Vehicle Environment with Subgoaling: Cycle 35.

Subgoals from pattern track extension goals raise the ratings of synthesis knowledge source instantiations working on the weak portion of the actual track. Vehicle location hypotheses are synthesized for both weak locations and a vehicle track extension is hypothesized.

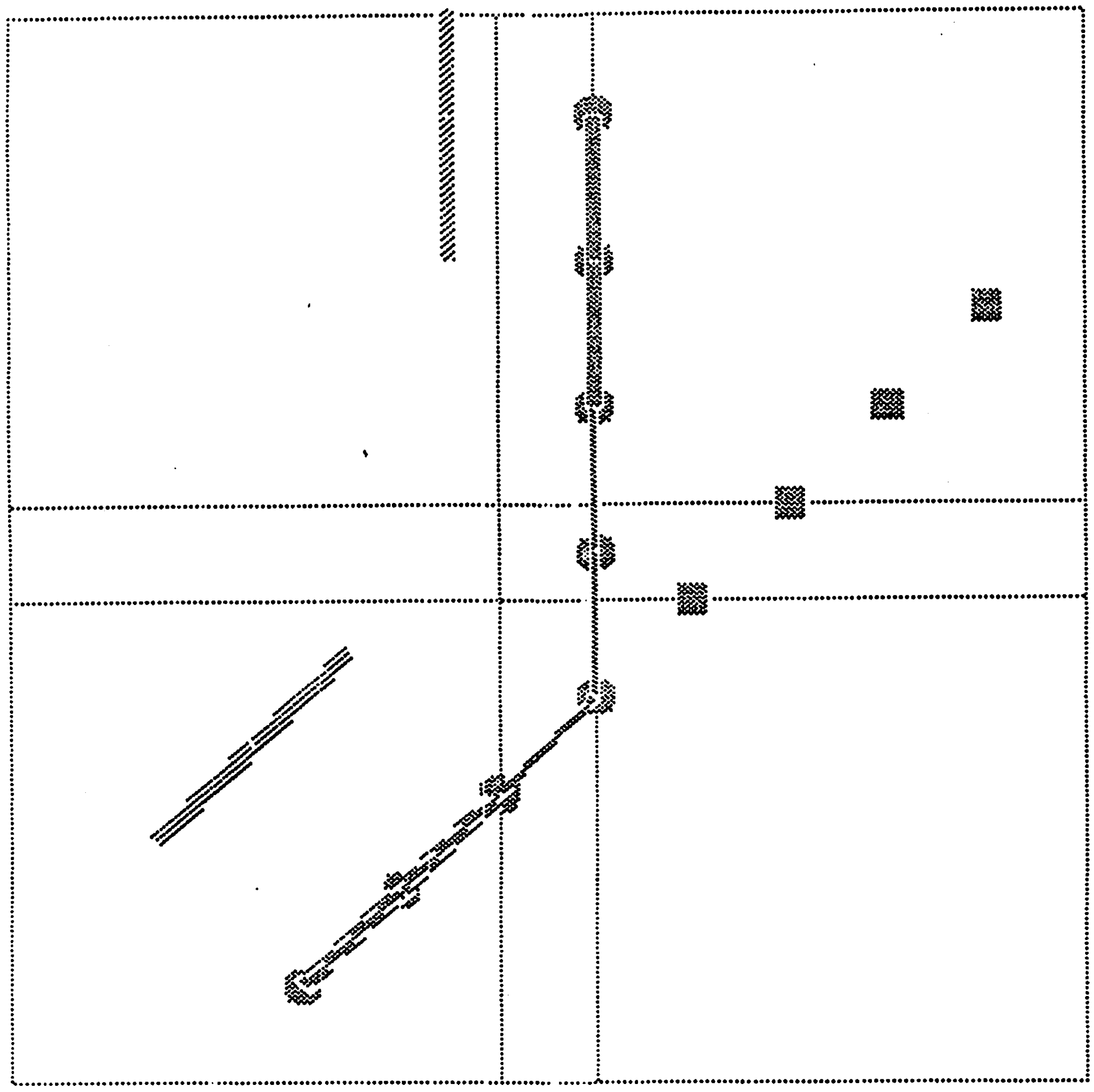


Figure 55: Bent Vehicle Environment with Subgoaling: Cycle 36.

A spanning set of vehicle track hypotheses for the actual track is completed.

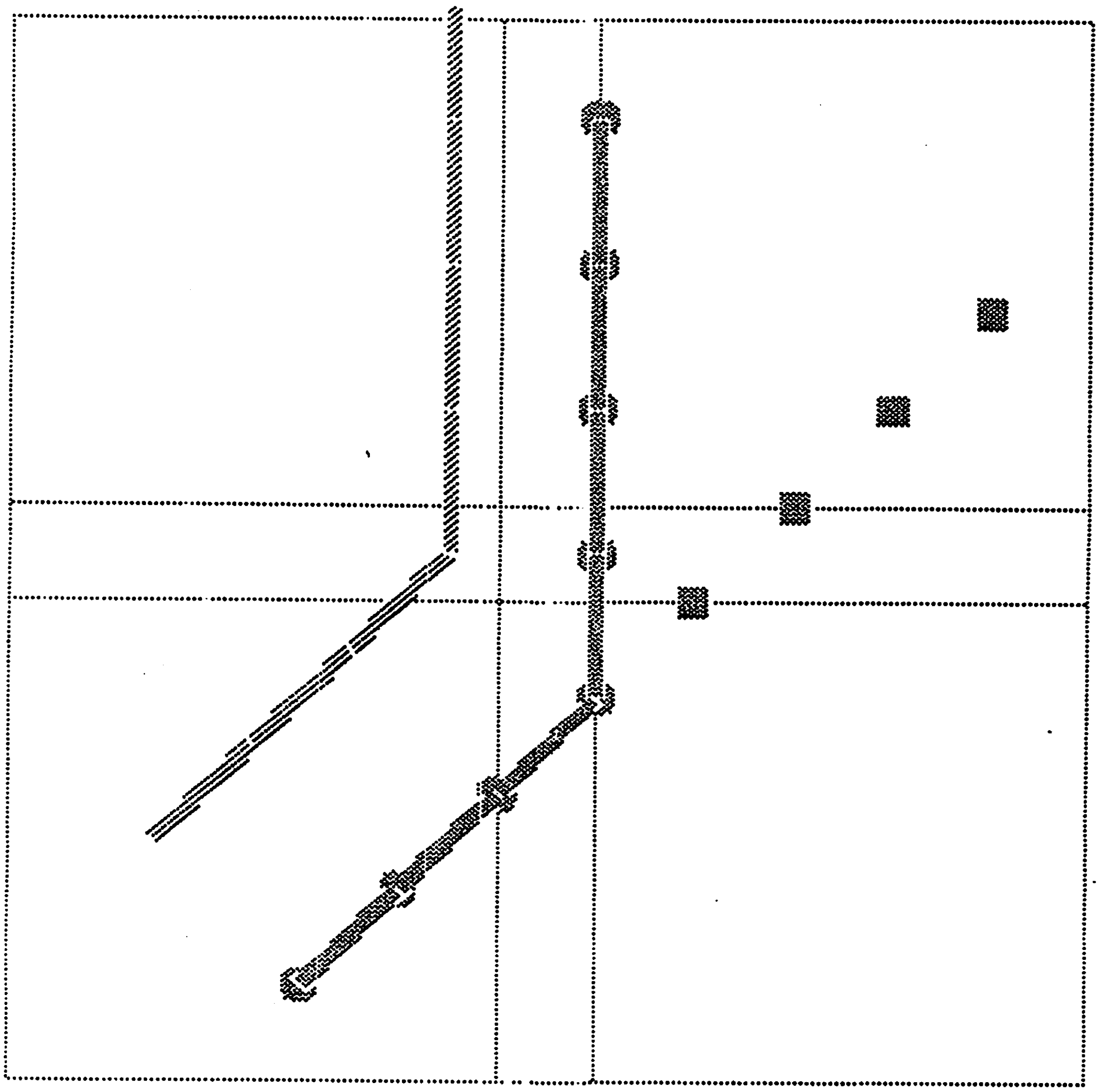


Figure 56: Bent Vehicle Environment with Subgoaling: Solution at Cycle 65.

The vehicle track hypotheses are synthesized and merged into a complete pattern track hypothesis for the actual track. Despite the bend in the actual track, the use of subgoaling effectively changed the ordering of knowledge source instantiations to allow completion of the actual track before any work was begun on the ghost extension. This level of performance was better than expected.

knowledge sources should be executed. When the ratings of goals are changed through subgoaling, some goals that were worked on early in one experiment are delayed in the other and vice versa. The delayed goals continue to acquire additional stimulus hypotheses before the knowledge source instantiation is executed. These additional hypotheses provide a different input context for the knowledge source. Goals which are highly rated may have their knowledge source instantiation executed before additional stimulus hypotheses arrive. These latter stimulus hypotheses cause the goal to be reactivated and possibly worked on by a different knowledge source instantiation in the newer context. These timing differences between the two experiments, coupled with asymmetries in the candidate generators, lead to the different results. It should be noted that although the absolute values of the beliefs of the actual and ghost pattern track hypotheses are different in the two experiments, the relative difference between the actual and ghost hypothesis beliefs remain unchanged. Therefore, the actual belief values placed on hypotheses are less significant than the relative relationship among the beliefs when comparing the performance of testbed experiments.

5.2.4 Real-time experiments.

In the previous experiments, all sensory data were available at the start of the run. Such "batch" mode experiments allow the system to work wherever the data are the strongest. A set of experiments in which the data for each time frame are inserted every fifth cycle was performed to investigate how the system performs in a "real-time" setting. (Again this is a simple change to the environment file.) The four non-exhaustive experiments described above were rerun with all parameters unchanged except for the real-time insertion of signal location hypotheses.

Straight vehicle environment.

Real-time insertion of sensory signal location hypotheses improved the performance of the system, both with and without the use of subgoaling. Without subgoaling the system executes 152 knowledge source instantiations (a speed-up of 5 cycles over the batch mode experiment), creates 301 hypotheses, places a belief of 4443 on the actual pattern track hypothesis, and places a belief of 4100 on the ghost pattern track hypothesis. With subgoaling the improvement is even greater. The system executes 39 knowledge source instantiations (a speed-up of 13 cycles over the batch mode experiment), creates 135 hypotheses, places a belief of 4712 on the actual pattern track hypothesis, and is stopped before creating a ghost pattern track hypothesis.

Bent vehicle environment.

Without subgoaling the real-time bent vehicle environment executes 114 knowledge source instantiations (a speed-up of two cycles over the batch mode experiment), creates 247 hypotheses, places a belief of 4704 on the actual pattern track hypothesis, and places a belief of 4160 on the ghost pattern track hypothesis (Figures 57 through 63). With subgoaling the real-time bent vehicle environment executes 76 knowledge source instantiations (an increase in 11 cycles over the batch mode experiment), creates 175 hypotheses, places a belief of 4912 on the actual pattern track hypothesis, and places a belief of 4056 on the

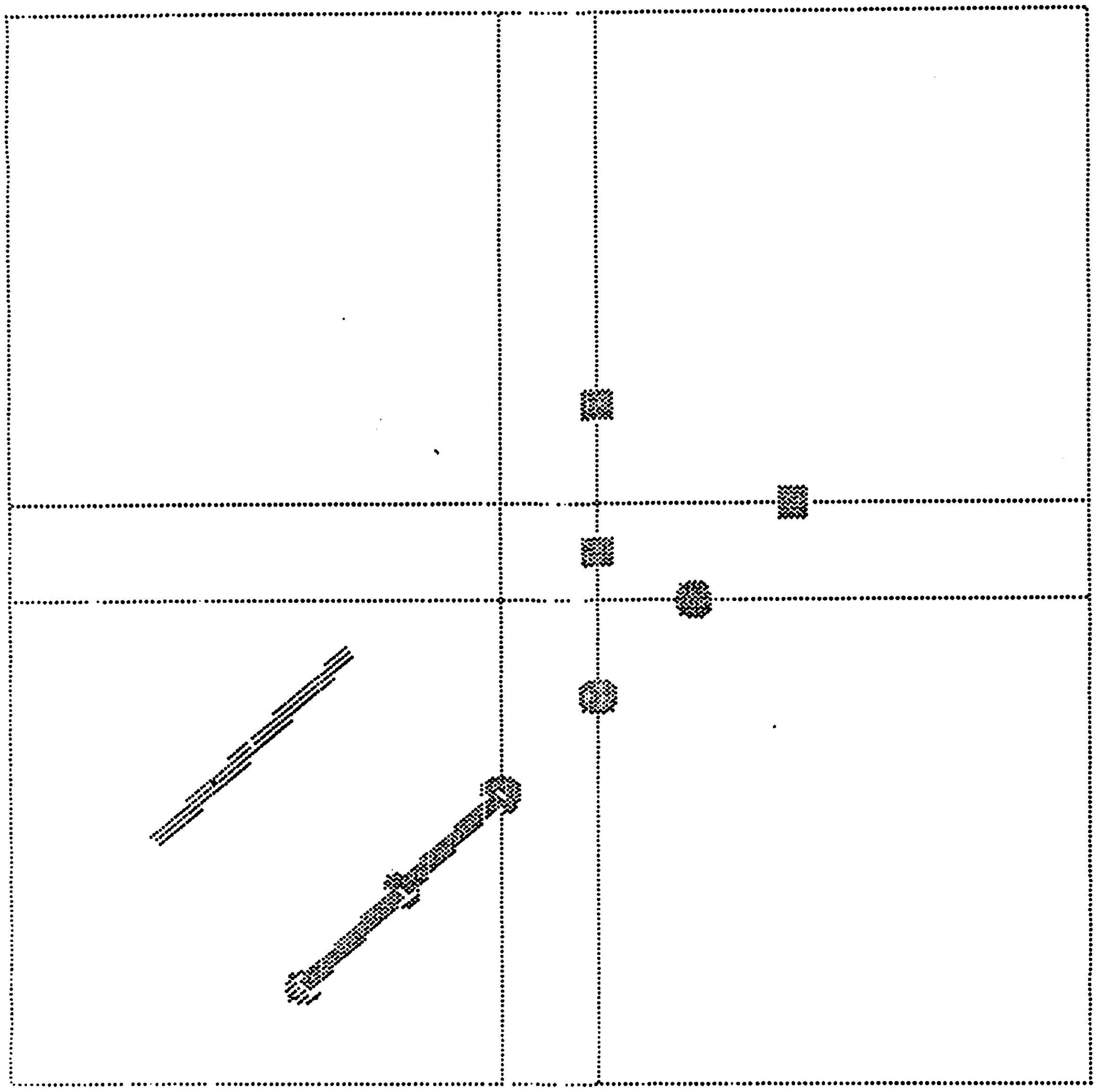


Figure 57: Real-Time Bent Vehicle Environment without Subgoaling: Cycle 18.

The system first synthesizes a pattern track hypothesis from the first strong portion of the actual track and, as new data arrives, synthesizes vehicle location hypotheses from the ghost extension. (Vehicle location hypotheses are indicated by circular symbols and signal location hypotheses by square symbols.)

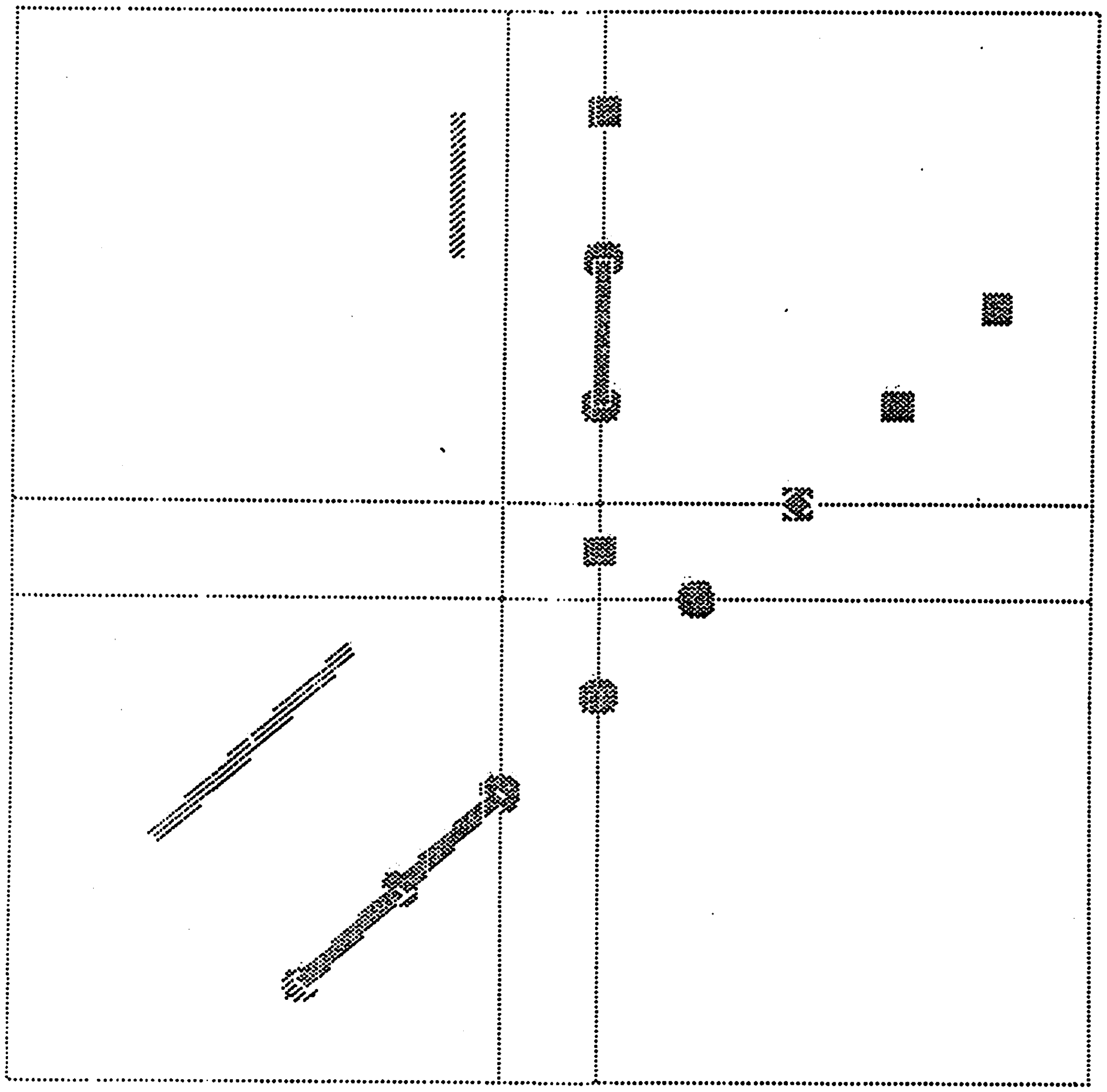


Figure 58: Real-Time Bent Vehicle Environment without Subgoaling: Cycle 29.

The second strong portion of the actual track is sensed and the system drops work on the ghost extension to synthesize a pattern track hypothesis for part of the second strong portion.

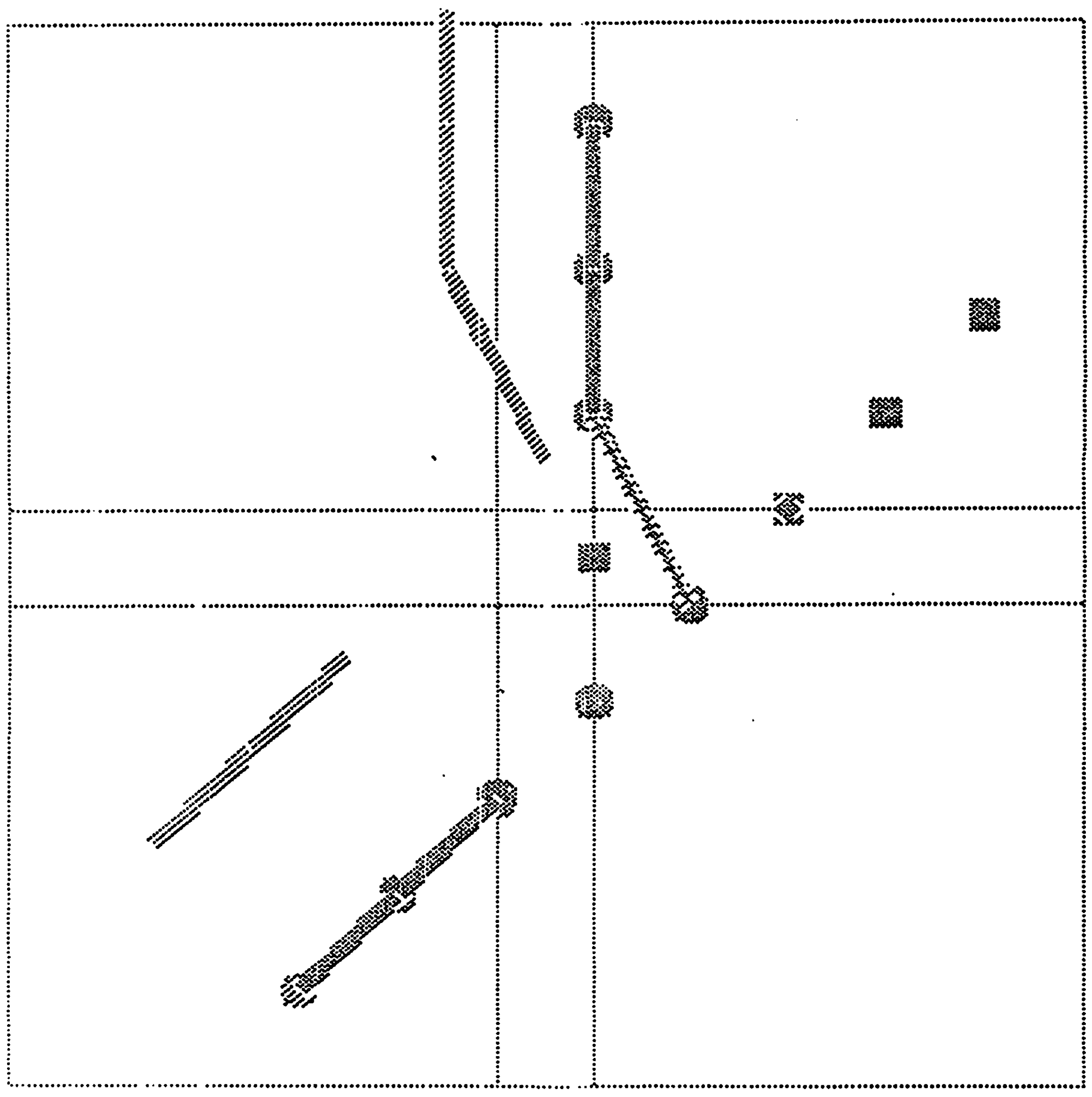


Figure 59: Real-Time Bent Vehicle Environment without Subgoaling: Cycle 36.

The system forms a pattern track hypothesis connecting the second strong portion of the actual track with part of the ghost extension and, once the last time frame has been received, synthesizes a pattern track hypothesis for the remainder of the second strong portion.

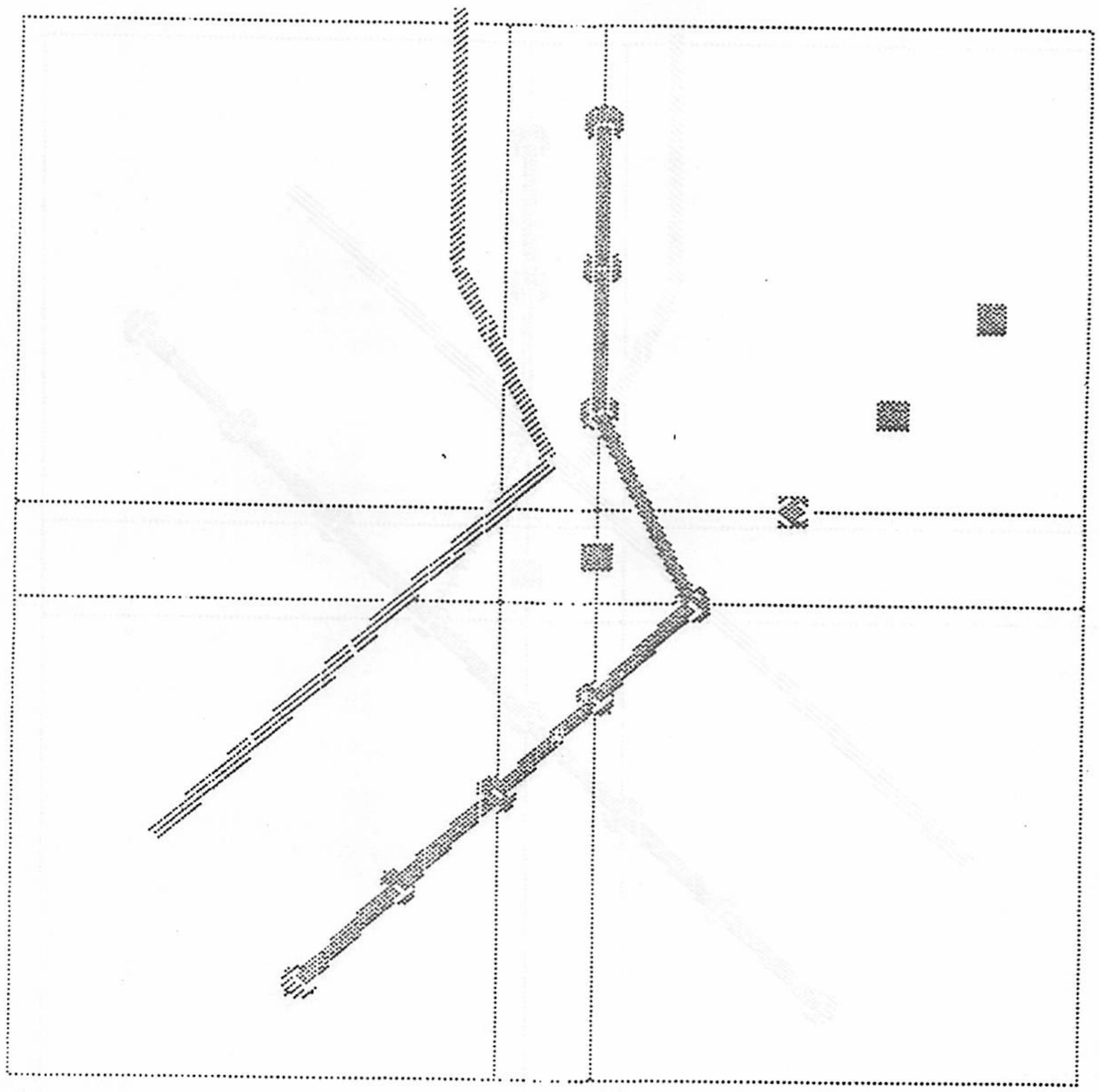


Figure 60: Real-Time Bent Vehicle Environment without Subgoaling: Cycle 41.

The system forms pattern track hypotheses for a complete track which include part of the actual track data and part of the ghost extension.

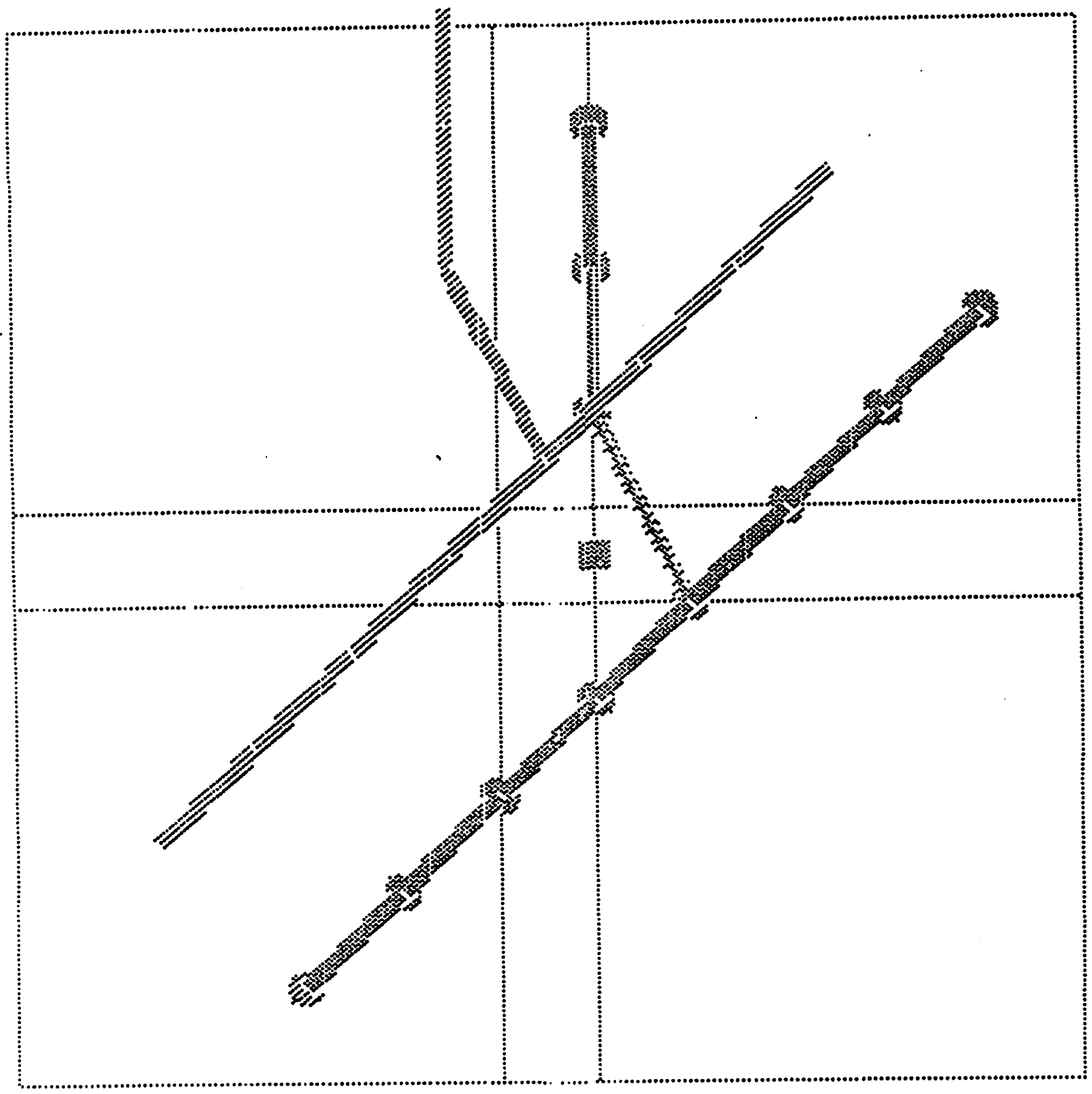


Figure 61: Real-Time Bent Vehicle Environment without Subgoaling: Cycle 74.

The system forms pattern track hypotheses for the remainder of the ghost extension.

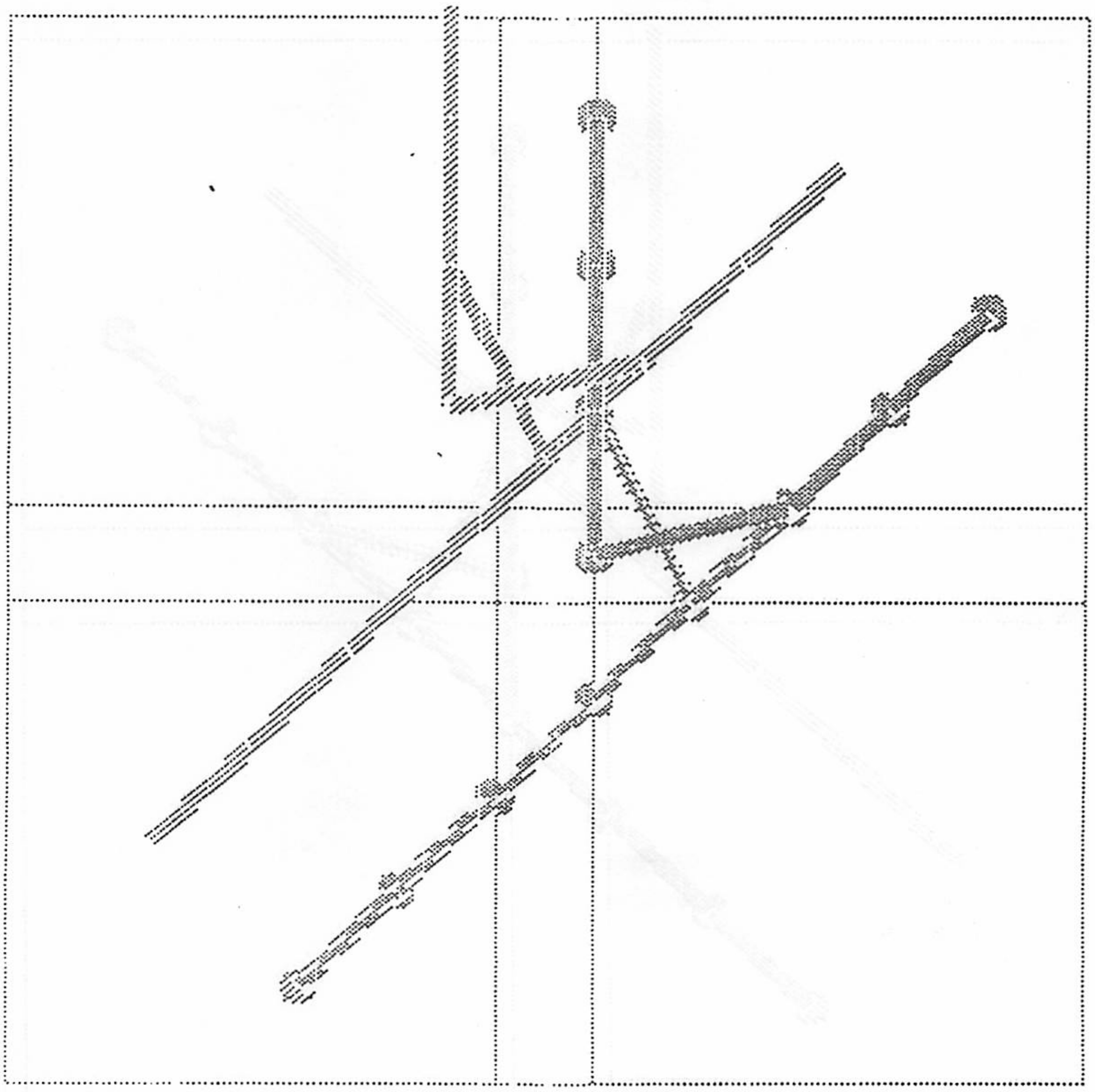


Figure 62: Real-Time Bent Vehicle Environment without Subgoaling: Cycle 97.

The system uses the weaker portion of the actual track in forming a track with the second strong portion of the actual track and with the ghost extension.

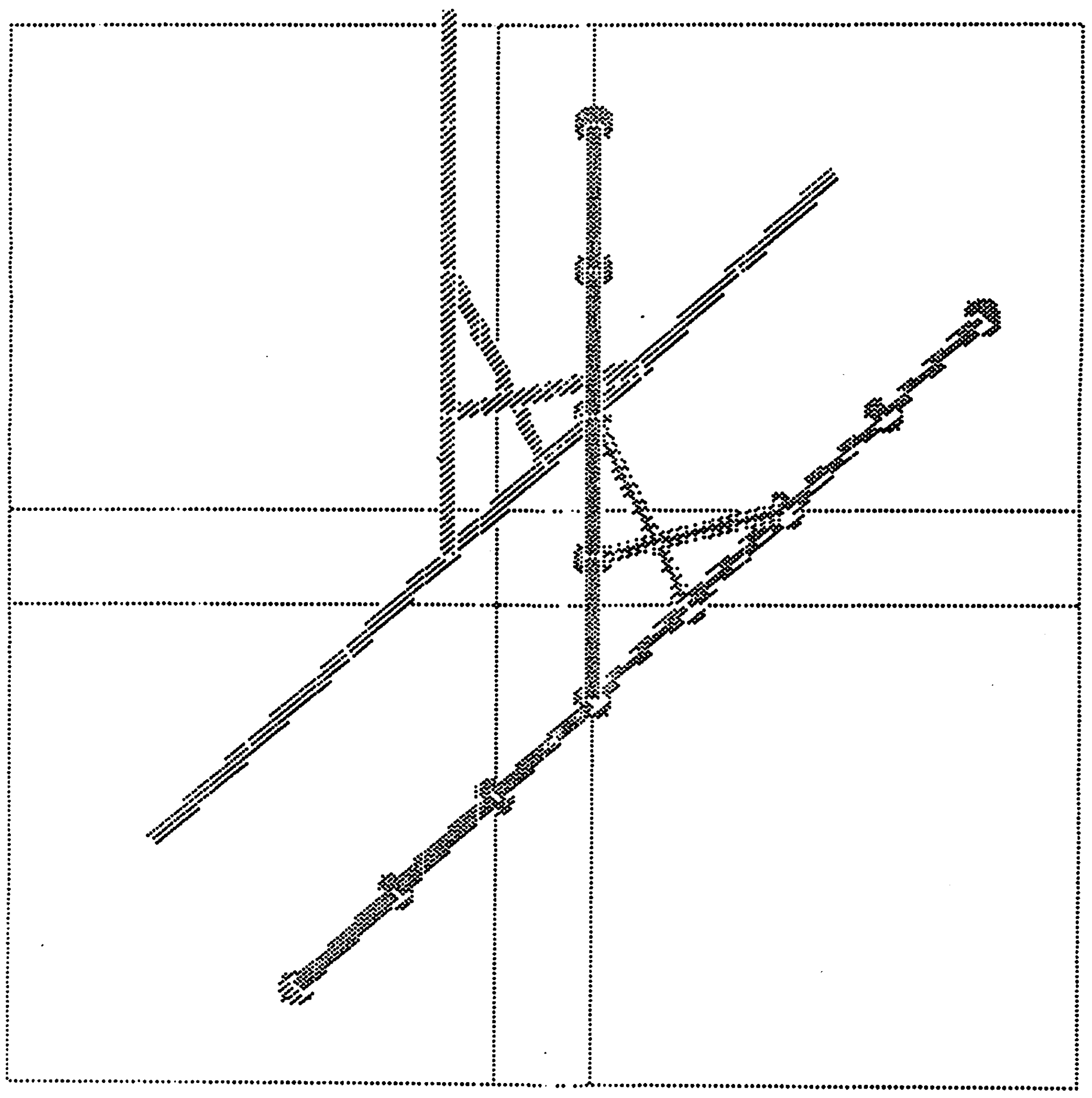


Figure 63: Real-Time Bent Vehicle Environment without Subgoaling: Solution at Cycle 114.

Working backward in time, the system eventually includes the remaining weak time frame of the actual track in the complete actual pattern track hypothesis. The real-time experiment required two less knowledge source executions than the batch mode experiment.

ghost pattern track hypothesis before the system is stopped (Figures 64 through 69). The delay in sensing the second strong portion of the actual track allows the extension of the beginning of the actual track along the ghost extension. The strong depth-first predictive effect of subgoaling has, in this case, lead the system away from the actual track. However, even in this difficult environment the system with subgoaling significantly outperforms the system without subgoaling.

5.2.5 The cost of goal processing.

A continuing issue in artificial intelligence is whether quickly searching many possibilities is more effective than making more informed (and expensive) decisions about where to spend problem solving effort. The balance between "speed" and "smarts" is basically one of optimization and depends on the specifics of the problem solving situation. The goal-directed Hearsay-II architecture and planner provide the capability of making more informed local coordination decisions than the data-directed Hearsay-II architecture. These decisions are not without cost, and this section considers if this expenditure is more than rewarded with a lower total problem solving and coordination cost.

Various metrics can be developed for comparing the cost of subgoaling in these experiments. The most obvious comparison would be to compare the execution time required by the testbed in the runs with and without subgoaling. However, this metric would be misleading, since knowledge sources are simulated by syntactic candidate generators whose powers are varied using information on the consistency blackboard and

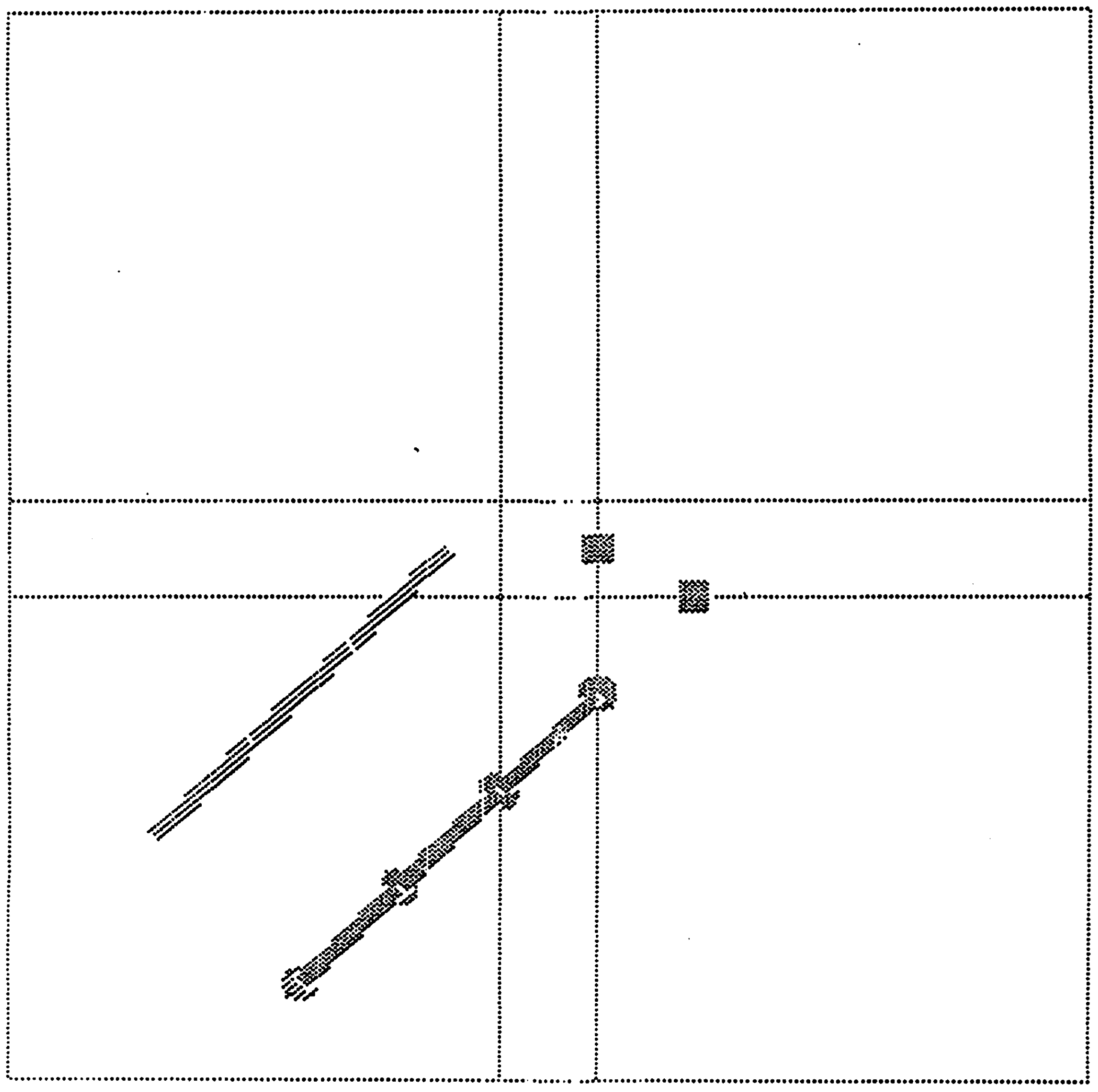


Figure 64: Real-Time Bent Vehicle Environment with Subgoaling: Cycle 16.

The system first synthesizes a pattern track hypothesis from the first strong portion of the actual track and, as the next time frame arrives, extends a pattern track hypothesis to include the first weak location in the actual track (the only possibility).

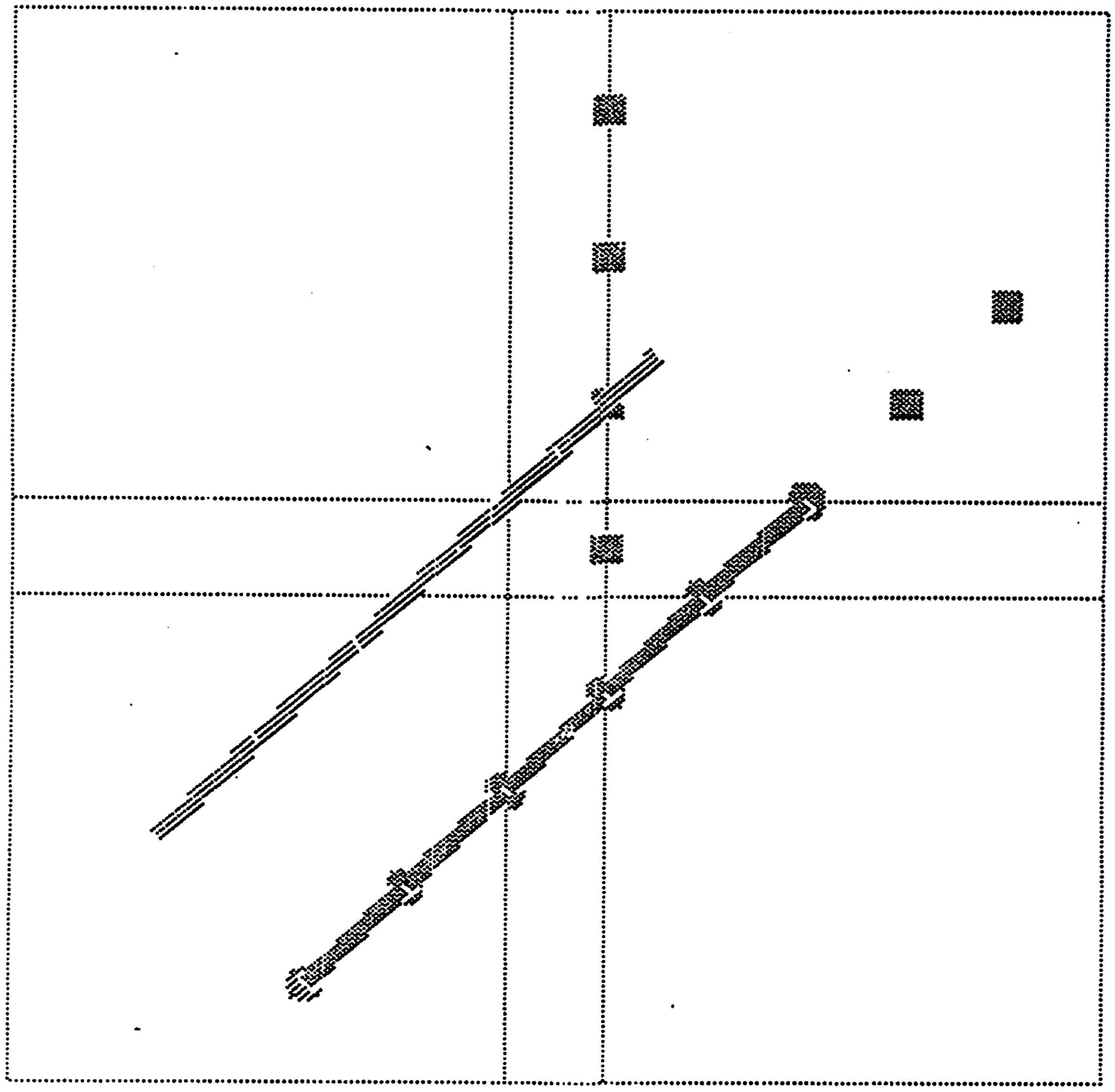


Figure 65: Real-Time Bent Vehicle Environment with Subgoaling: Cycle 29.

As additional time frames arrive, subgoaling the high level extension goals of the pattern track hypotheses causes the system to continue with the ghost extension even though stronger signal location hypotheses at the end of the actual track are available.

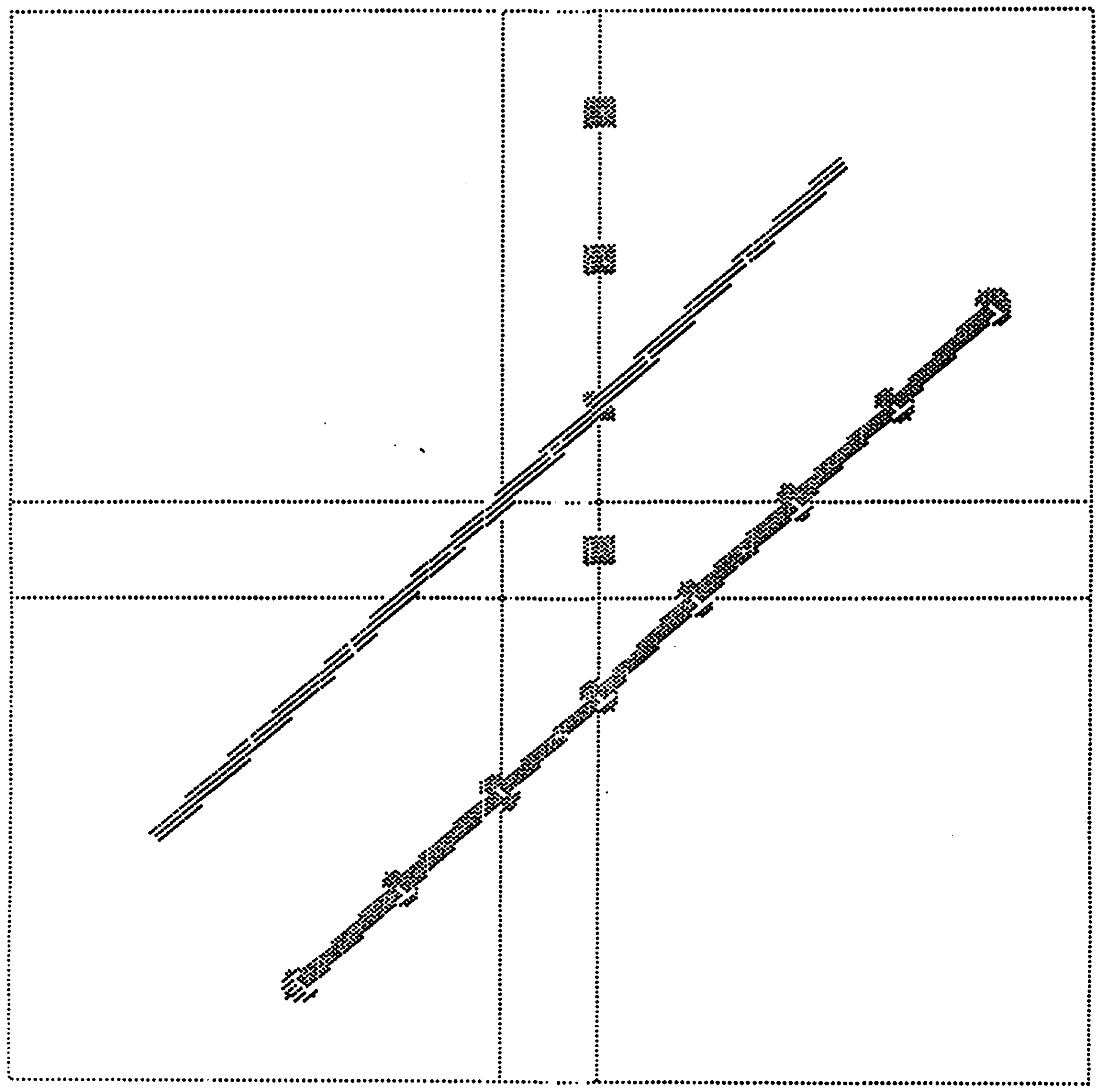


Figure 66: Real-Time Bent Vehicle Environment with Subgoaling: Cycle 39.

Synthesis of the ghost track extension continues through the last time frame.

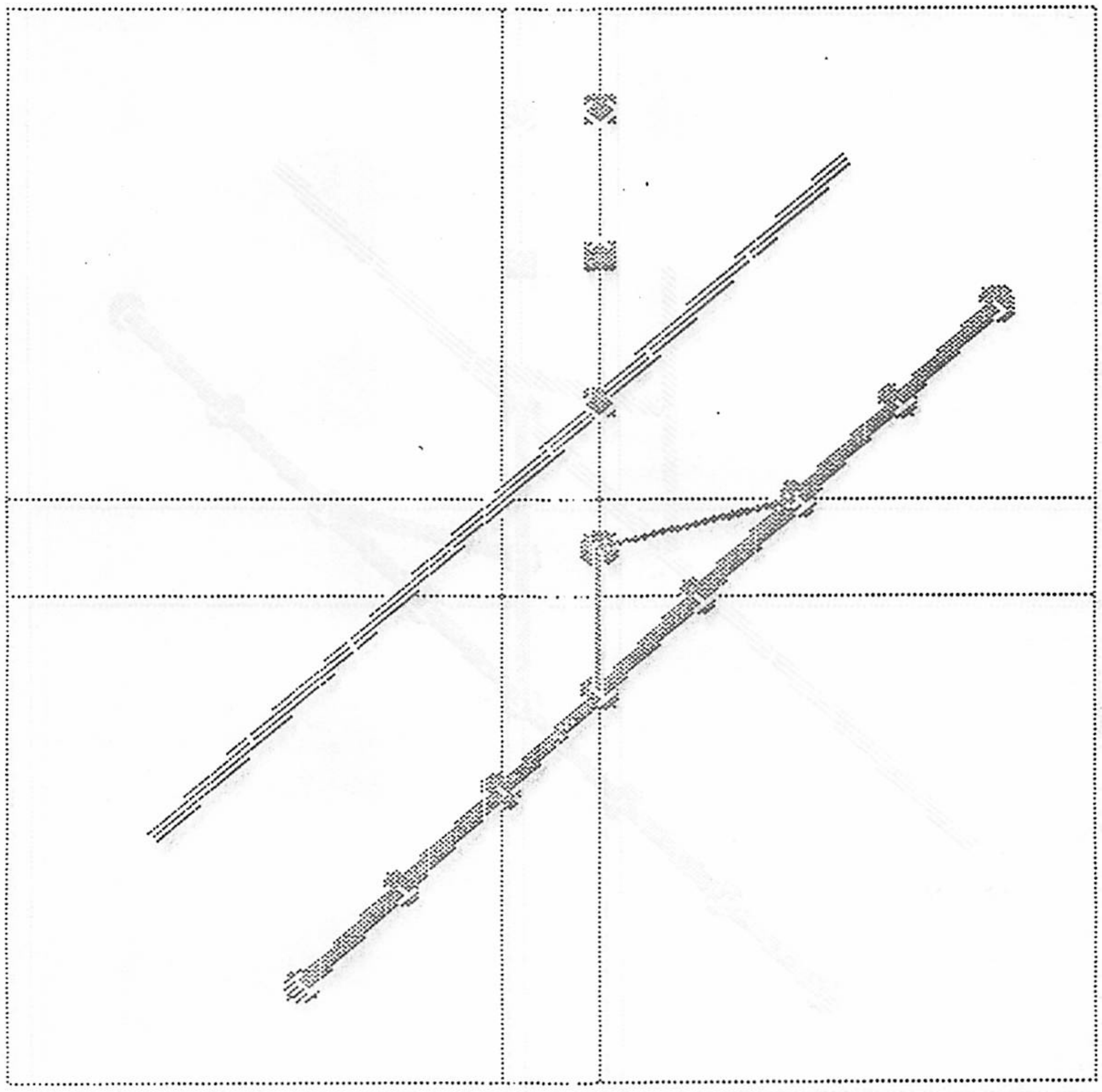


Figure 67: Real-Time Bent Vehicle Environment with Subgoaling: Cycle 49.

The system begins synthesizing higher level location hypotheses from the remaining signal location data. Since the second weak location of the actual track is also within the extension goals generated from the existing pattern track hypotheses, vehicle track hypotheses including it are created.

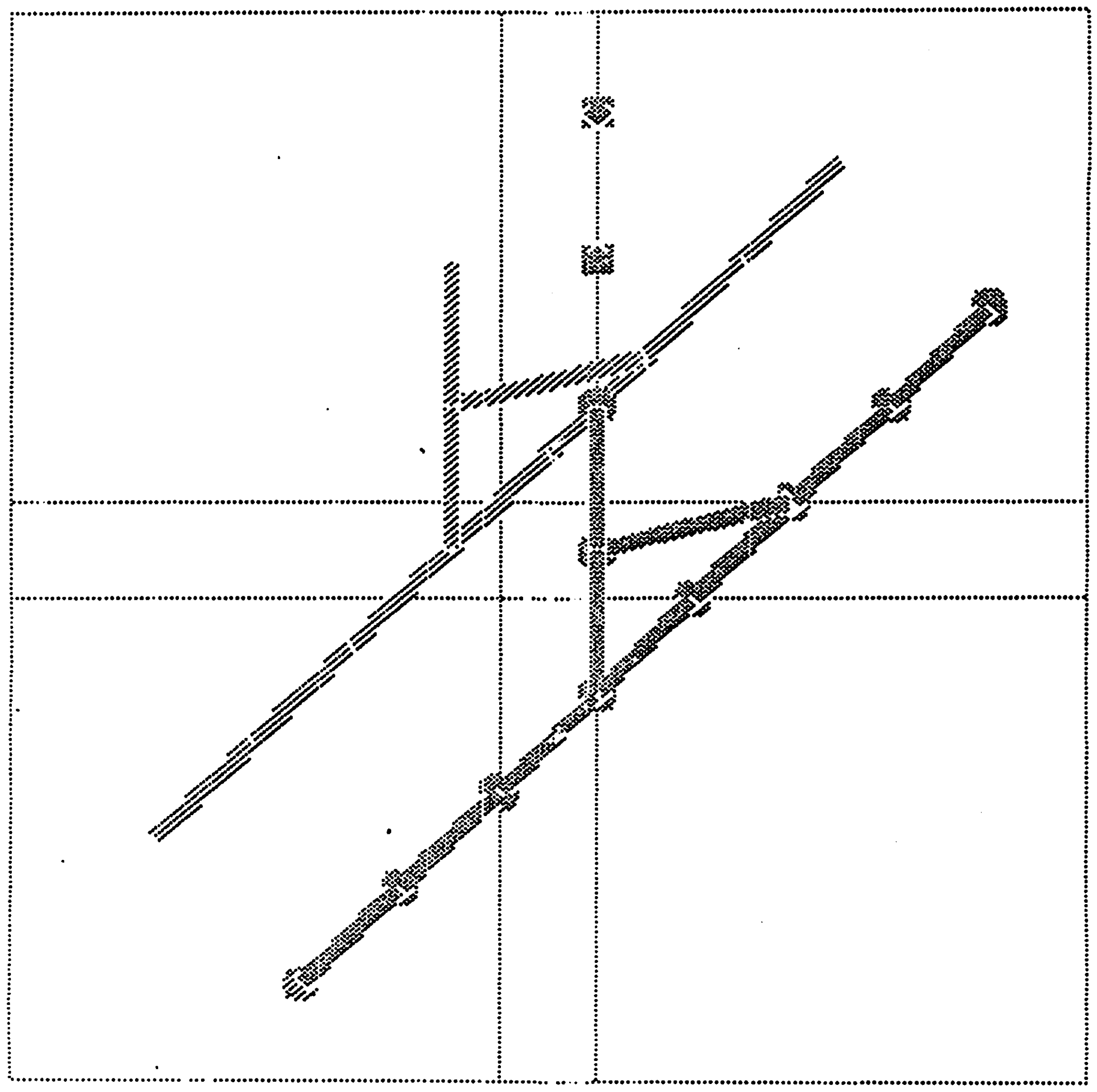


Figure 68: Real-Time Bent Vehicle Environment with Subgoaling: Cycle 63.

The system continues forming pattern track hypotheses with the remaining actual track data.

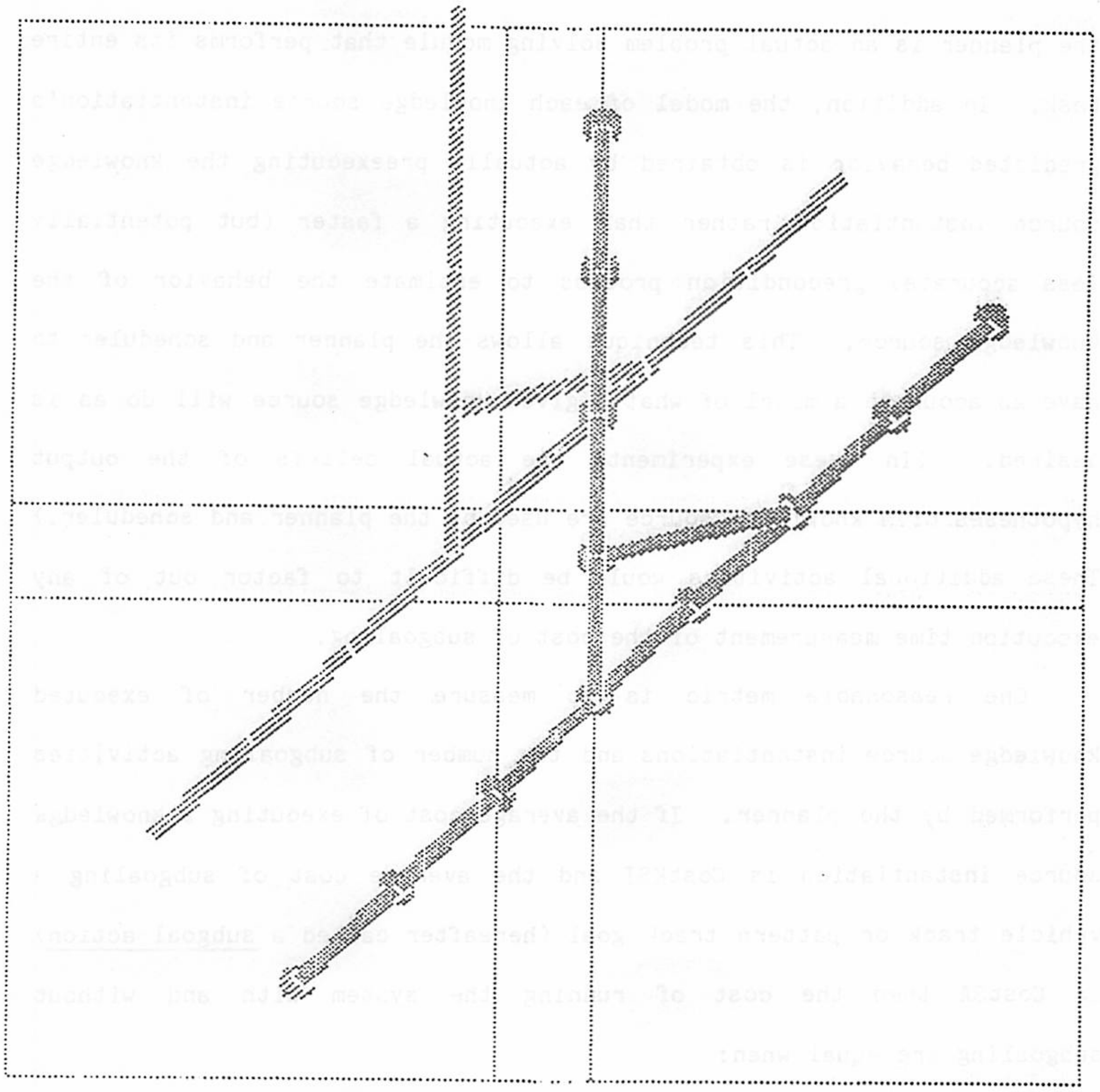


Figure 69: Real-Time Bent Vehicle Environment with Subgoaling: Solution at Cycle 76.

The complete actual pattern track hypothesis is generated in cycle 76. This is 11 cycles longer than the 65 cycles required in the "batch" mode experiment with subgoaling.

the planner is an actual problem solving module that performs its entire task. In addition, the model of each knowledge source instantiation's predicted behavior is obtained by actually preexecuting the knowledge source instantiation rather than executing a faster (but potentially less accurate) precondition process to estimate the behavior of the knowledge source. This technique allows the planner and scheduler to have as accurate a model of what a given knowledge source will do as is desired. (In these experiments the actual beliefs of the output hypotheses of a knowledge source are used by the planner and scheduler.) These additional activities would be difficult to factor out of any execution time measurement of the cost of subgoaling.

One reasonable metric is to measure the number of executed knowledge source instantiations and the number of subgoaling activities performed by the planner. If the average cost of executing a knowledge source instantiation is CostKSI and the average cost of subgoaling a vehicle track or pattern track goal (hereafter called a <u>subgoal action</u>) is CostSA then the cost of running the system with and without subgoaling are equal when:

$$\text{CostKSI} * \#\text{KSIs}_{\text{no subgoaling}} = \text{CostKSI} * \#\text{KSIs}_{\text{subgoaling}} + \text{CostSA} * \#\text{SAs}.$$

The ratio of the costs is then:

$$\frac{\text{CostSA}}{\text{CostKSI}} = \frac{\#\text{KSIs}_{\text{no subgoaling}} - \#\text{KSIs}_{\text{subgoaling}}}{\#\text{SAs}}.$$

This ratio has the following values in the above experiments:

Environment	Sensory Data	Reduction in KSIs	Subgoal Actions	Cost Ratio
straight	batch	105	39	2.7
straight	real-time	113	27	4.2
bent	batch	51	55	0.9
bent	real-time	38	66	0.6

Even in the bent, real-time environment where subgoaling is the least effective, the cost of a subgoaling action would have to be 60 percent of the cost of executing an average knowledge source instantiation for subgoaling to lose in total execution time. If knowledge source instantiations perform significant computation (as is the norm with Hearsay-II systems) the cost of a subgoaling action will tend to be a small percentage of the cost of executing a knowledge source instantiation. This analysis assumes that the saved cost of knowledge source instantiations eliminated by subgoaling is close to the average cost of executing a knowledge source instantiation.

5.2.6 The balance between data-directed and goal-directed control.

To explore the balance between data-directed and goal-directed control a set of experiments was performed in which the weight given by the scheduler to the beliefs of created hypotheses versus the importance of created goals was varied.

Straight vehicle environment.

The results with the straight vehicle environment were:

Goal Weight	Executed KSIs	Created Hyps	Actual Track Belief	Ghost Track Belief
0	123	300	4222	4100
2000	130	304	4222	4100
4000	52	153	4221	-
6000	52	152	4041	-
8000	52	157	4666	-
10000	75	207	4777	-

When the system is heavily data-directed or goal-directed it performed worse than when there is a balance of the two. This behavior stems from the predictions provided by subgoaling vehicle track hypotheses. With the goal weight set to zero the system is essentially operating without subgoaling.[3] As the goal weight is increased, predictions from the strong portion of the actual track sharply reduce the number of executed knowledge source instantiations needed to generate the actual vehicle track. When activity becomes entirely goal-directed, false predictions made from the ghost track lead the system to again work on the ghost track before completing the actual track. With the straight vehicle environment there is a wide range of intermediate goal weights with similar system performances.

3. There is a slight speed-up over no subgoaling due to the restimulation of lower level data-directed goals by newly-created subgoals. This reactivates the planner and can cause it to schedule a different knowledge source for the goal.

<u>Bent vehicle environment.</u>

The results with the bent vehicle environment were:

Goal Weight	Executed KSIs	Created Hyps	Actual Track Belief	Ghost Track Belief
0	89	219	4520	4296
2000	86	223	4520	4496
4000	90	202	4680	4304
6000	95	207	4632	4376
8000	65	173	4416	-
10000	76	208	4656	3800*

* This track includes time frames 1 through 5 only.

With the bent vehicle environment, the system is more sensitive to the setting of the goal weight. The weightings of 6000 and below all performed poorly with respect to the 8000 setting. At 10000 the system again performed worse than at 8000, although better than the lower weightings. It may seem surprising that a heavily goal-directed weighting would perform so much better than a data-directed weighting in the bent vehicle environment where predictions can falsely lead the system to use the ghost track extension. The explanation is that a fairly strong prediction component from the second strong portion of the actual vehicle track is required to avoid work on the ghost extension. The lower goal weightings simply do not provide enough of this prediction.

5.2.7 Summary of single node experiments.

The single node experiments reported in this section demonstrate that the goal-directed Hearsay-II architecture, and particularly the use of subgoaling as a focusing technique, can be used to direct the problem solving activities of a node through the use of goals. For direct comparison, the results of the single node experiments are summarized in Table 10.

The goal-directed Hearsay-II architecture was used in the single node problem solving experiments to plan interpretation activities that are both coordinated and purposeful. However, the architecture also implements network coordination decisions in distributed testbed configurations. This is the subject of the experiments reported in the next section.

5.3 Multinode Experiments

This section describes experiments performed with a distributed testbed network configuration. The purpose of these experiments is to compare the performance of the single node experiments with different distributed network architectures and organizational problem solving strategies. First, the network architectures are presented.

5.3.1 Multinode architectures.

Two different distributed network architectures were used in these experiments: a laterally-organized, four-node network with broadcast communication among nodes at the vehicle track level and a hierarchically-organized, five-node network in which the fifth node acts

SUMMARY OF SINGLE NODE EXPERIMENTS

Environment	Subgoal Threshold	Goal Weight	Sensory Data	Solution Threshold	Executed KSIs	Created Hyps	Subgoaled Goals	Actual Track Belief	Ghost Track Belief
straight	10000	8000	batch	10000	323	552	0	4952	4098
straight	0	8000	batch	10000	322	572	313	4952	4100
straight	10000	8000	batch	4000	157	313	0	4499	4098
straight	0	0	batch	4000	123	300	136	4222	4100
straight	0	2000	batch	4000	130	304	135	4222	4100
straight	0	4000	batch	4000	52	153	37	4221	-
straight	0	6000	batch	4000	52	152	35	4041	-
straight	0	8000	batch	4000	52	157	39	4666	-
straight	0	10000	batch	4000	75	207	70	4777	-
straight	10000	8000	real-time	4000	152	301	0	4443	4100
straight	0	8000	real-time	4000	39	135	27	4712	-
bent	10000	8000	batch	10000	184	384	0	4912	4752
bent	0	8000	batch	10000	164	390	192	4416	4304
bent	10000	8000	batch	4000	116	258	0	4912	4752
bent	0	0	batch	4000	89	219	74	4520	4296
bent	0	2000	batch	4000	86	223	76	4520	4496
bent	0	4000	batch	4000	90	202	67	4680	4304
bent	0	6000	batch	4000	95	207	67	4632	4376
bent	0	8000	batch	4000	65	173	55	4416	-
bent	0	10000	batch	4000	76	208	67	4656	3800*
bent	10000	8000	real-time	4000	114	247	0	4704	4160
bent	0	8000	real-time	4000	76	175	66	4912	4056

* This track includes time frames 1 through 5 only.

Table 10: Summary of Single Node Experiments.

as an integrating node. In all of these experiments the balance between data-directed and goal-directed control is specified by a goal weight of 8000 and all sensory signal location hypotheses are available when other processing begins ("batch" mode). The network is stopped when one of the nodes generates the complete actual pattern track hypothesis (as specified by the consistency blackboard) with a belief above 4000.

Internode communication.

Communication among nodes in these experiments is modeled as a limited broadcast channel, although the limitation is relatively minor. During every second network cycle, a node can transmit up to two hypotheses and two goals to other nodes. A similar restriction is placed on the receiving end; every second cycle up to two hypotheses and two goals can be received from other nodes. A communication channel delay of one network cycle is placed between transmission and reception of a message (a hypothesis or a goal).

In these experiments messages are not lost, but are queued in order of belief (for hypotheses) or rating (for goals) until they can be transmitted or received. Knowledge sources involved in transmitting or receiving hypotheses or goals are not charged for their time in these experiments, so a node can send and receive up to its limit and still execute one knowledge source instantiation during its network cycle.

These parameters are specified as part of the environment file. The particular settings selected for these experiments are fairly unrestrictive, providing an estimate for the basic problem solving requirements of the network rather than its performance with limited communication resources.

The four-node, lateral network.

In the four-node network each node is positioned near one of the sensors and receives signal location hypotheses from that sensor only. The interest areas of each node specify that it is to synthesize its sensory data to the vehicle track level and transmit any of these vehicle track hypotheses that can be extended into the sensory area of another node to that node. Each node is also directed to attempt to generate hypotheses at both the vehicle track and pattern track levels which span the entire monitoring area. This means that each node is in a race with the other three to generate the complete answer map.

The five-node, hierarchical network.

In the five-node network four of the nodes are positioned near one of the sensors and receive signal location hypotheses only from that sensor. (Their signal location input is identical to the four-node network.) The fifth node receives no sensory data. Instead, it is instructed through its interest areas to work only at the vehicle track and pattern track levels with vehicle track hypotheses received from the other four nodes. The four nodes with sensory data are assigned the role of synthesizing their signal location hypotheses to the vehicle track level and transmitting them to the fifth node. In the five-node network configuration, these four nodes do not work outside the area of their sensory data at any blackboard level and do no work at the pattern track level.

5.3.2 Different four-node organizational problem solving strategies.

Within the four-node network configuration, a number of different organizational problem solving strategies were tried. Parameters that are varied include whether communication is voluntary (a node transmits vehicle track hypotheses at its pleasure) or requested (a node transmits vehicle track hypotheses only when that information is requested by another node), whether a node is self-directed or externally-directed in its activities, and whether hypotheses or goals are used for network coordination. Combined strategies are also tried.

Next, the particulars of the various four-node organizational problem solving strategies are described. The results of the experiments are presented last.

Voluntary hypothesis communication with self-directed control.

Voluntary communication with self-directed control is specified by enabling two additional knowledge sources at each node, HYP:SEND:VT:VT and HYP:RECEIVE:VT:VT. Each node is also given hypothesis transmission interest areas specifying transmission of any vehicle track hypothesis with a belief above zero (effectively all) that can be extended into the area covered by another node's sensor to that other node. Hypothesis reception interest areas to receive these hypotheses are also supplied to each node.

To keep the node entirely self-directed in its coordination decisions, the blackboard monitor and planner are instructed (using the focusing-weight interest area parameter) to reduce the rating of goals generated directly from received vehicle track hypotheses to 30 percent of their normal rating. (The actual percentage is not significant, only that it is small.) Since a goal weighting of 80 percent is used in the knowledge source instantiation rating calculation, it is unlikely that a node will use the receipt of a vehicle track hypothesis to directly change its scheduled activities. The beliefs of received vehicle track hypotheses, however, are not reduced. This means that the node can use received information in extending its own hypotheses without having to find local information that can be combined with the received hypotheses. This separation of belief in the data from focusing priority fits nicely into the goal-directed architecture.

Two variants of this strategy are tried, one with and one without subgoaling of internally-generated vehicle track and pattern track hypotheses, by setting the internal subgoal threshold at zero and 10000.

Voluntary communication with externally-directed control.

Voluntary communication with externally-directed control also requires the two additional knowledge sources, HYP:SEND:VT:VT and HYP:RECEIVE:VT:VT, at each node. Again, each node is given hypothesis transmission and hypothesis reception interest areas specifying communication of vehicle track hypotheses that can be extended into the area covered by another node's sensor to that other node.

Externally-directed control is obtained by instructing the planner to subgoal only those goals generated from the receipt of vehicle track hypotheses (by setting the subgoal threshold to 10000 for internally generated goals and to zero for goals generated from received hypotheses). In this strategy the ratings of goals generated from received vehicle track hypotheses are not reduced from their normal values. The receipt of a highly-believed hypothesis from another node causes the receiving node to try its best to find something that can be combined with the received hypothesis.

Voluntary communication with both self-directed and externally-directed control.

This strategy is a combination of the above strategies. It is obtained by setting both the internal and received hypothesis subgoaling thresholds to zero. Each node works on whatever goals have the highest ratings, without regard to whether they are internally or externally generated. Again goals generated from received vehicle track hypotheses are rated at their normal values.

Requested communication with self-directed control.

Instead of using the communication knowledge source, HYP-SEND:VT:VT, this strategy requires GOAL-SEND:VT:VT, GOAL-RECEIVE:VT:VT, HYP-REPLY:VT:VT in addition to HYP-RECEIVE:VT:VT. Each node processes its local sensory data to the vehicle track level, but rather than voluntarily transmitting vehicle track hypotheses, any vehicle track goals that are within the sensory area of another node are sent to that node. When a node creates a vehicle track hypothesis that satisfies one of these received goals it uses HYP-REPLY:VT:VT to transmit the hypothesis to the originator of the goal.

Two variants of this strategy, one with and one without internal subgoaling, are tried. Received vehicle track goals are not subgoaled.

Requested communication with externally-directed control.

This strategy is similar to the requested communication with self-directed control. The only difference is that internal subgoaling is disabled and received goal subgoaling is enabled by setting the threshold for received goals from 10000 to zero. This means that only requests for hypotheses from other nodes are used for local focusing.

Requested communication with both self-directed and externally-directed control.

This strategy is a combination of the above two strategies. It is obtained by setting both the internal and received goal subgoaling thresholds to zero. Again each node works on whatever goals have the highest rating, without regard to whether they are internally or externally generated.

5.3.3 Results of the four-node network experiments on the straight vehicle environment.

Each of the organizational problem solving strategies was run using the straight vehicle environment. The results are shown in Table 11. The use of internal subgoaling (as shown in the two voluntary hypothesis communication with self-directed control strategies) results in a sizable improvement in the experiment with subgoaling. Not only is the number of network cycles reduced, but less than one-half the number of hypotheses are transmitted when subgoaling is used. In a communication limited situation, this reduction in communication requirements can be very important.

SUMMARY OF FOUR NODE EXPERIMENTS

Environment	Problem Solving Strategy	Internal Subgoal Threshold	Received Hyp Subgoal Threshold	Received Goal Subgoal Threshold	Network Cycles	Sent Hyps	Sent Goals	Actual Track Belief	Ghost Track Belief
straight	VH,SD	10000	10000	10000	54	21	0	4586(3)	**
straight	VH,SD	0	10000	10000	33	23	0	4246(3)	**
straight	VH,ED	10000	0	10000	86	39	0	4944(1)	**
straight	VH,S&ED	0	0	10000	79	45	0	4833(4)	4100(1-4)*
straight	RH,SD	10000	10000	10000	43	41	59	4912(3)	4100(2)
straight	RH,SD	0	10000	10000	32	32	80	4872(3)	**
straight	RH,ED	10000	10000	0	83	35	133	4928(2)	**
straight	RH,S&ED	0	10000	0	75	40	78	4850(1)	4100(2-4)*

Strategies:

VH	Voluntary Hypothesis Communication
RH	Requested Hypothesis Communication
SD	Self-Directed Control
ED	Externally-Directed Control
S&ED	Combined Self- and Externally-Directed Control

The numbers in parentheses following the actual and ghost track beliefs are the nodes where the hypothesis resides.

* This track is incomplete.
** Ghost track hypotheses less than 5 time frames long are not considered.

Table 11: Summary of Four-Node Network Experiments.

Comparing the performance of the four-node network in the voluntary, self-directed problem solving strategy with the performance of the single node network shows a speed-up of 66 percent without subgoaling and 37 percent with subgoaling. This means that the distributed interpretation without the use of subgoaling took 1.4 times as much processing as the centralized interpretation without subgoaling. With subgoaling the distributed interpretation took 2.5 times as much processing. While these figures would be disappointing in a parallel processing system, they are not unreasonable in a distributed problem solving network where the distribution of input data is not well-suited to parallelism. In fact, the distraction experiment (discussed below) shows that Node 1 is not contributing anything useful to the solution. Only two nodes (Nodes 2 and 3) receive large amounts of sensory data. It is the time needed for them to synthesize this data that limits the amount of parallelism in the network.

Whether the network used voluntary or requested communication of hypotheses had little effect on the number of network cycles required to generate an answer when subgoaling was used. When subgoaling was not used, the requested communication strategy resulted in fewer network cycles at the expense of additional hypothesis and goal communication.

Whether the strategy was self-directed or externally-directed had a much greater effect on network performance. The completely externally-directed strategies performed much worse than the completely data-directed strategies, with the combined strategies in between.

Distraction.

Why does externally-directed control perform so poorly in these experiments? A closer inspection reveals why. Node 1 (the node associated with Sensor 1, at the upper left of the figures) senses signal location hypotheses in only two time frames. Its signal location hypotheses are associated with the false ghost track. It does not sense the signal hypotheses associated with the actual vehicle track at all. Having no other work to perform, Node 1 quickly forms a two time-frame segment of the ghost track and transmits it to the other three nodes. Due to their bias to external direction and the higher belief associated with higher level hypotheses, these nodes suspend their lower level work on the actual track and attempt to extend the ghost track, resulting in inappropriate knowledge source activities and lost time. This is a prime example of distraction.

To verify that distracting information received from Node 1 is indeed the cause of the poor performance of the externally-directed strategies, the requested communication with both self-directed and externally-directed control was rerun with all knowledge sources at Node 1 disabled (simulating processor failure at Node 1). The number of network cycles was reduced from 75 with Node 1 to 38 without Node 1. The belief of the actual pattern track hypothesis was basically unchanged: 4850 with Node 1 and 4818 without it. In the straight vehicle environment, the network actually performs much better without Node 1, even though the remaining nodes still receive all signal location hypotheses associated with the ghost track (see Figure 35).

5.3.4 Different five-node organizational problem solving strategies.

Within the five-node network configuration, a number of different organizational problem solving strategies were tried. Parameters that are varied include whether communication with the integrating node is entirely voluntary or a mixed initiative combination of voluntary and requested hypotheses (a node volunteers only its highest rated hypotheses and awaits requests from the fifth node before transmitting any other hypotheses) and whether the four "worker" nodes are self-directed or externally-directed by the integrating node. In these experiments, only goals are used by the integrating node for coordinating the activities of the worker nodes.

The particulars of the various five-node network problem solving strategies are described, followed by the results of the experiments.

Voluntary communication with self-directed control.

Voluntary communication with self-directed control is specified by enabling the HYP-SEND:VT:VT knowledge source at each worker node and the HYP-RECEIVE:VT:VT knowledge source at the integrating node. Hypothesis transmission interest areas specifying the transmission of vehicle track hypotheses to the integrating node are also given to the worker nodes. The integrating node is given a hypothesis reception interest area specifying the reception of hypotheses from the other four nodes. The worker nodes only have processing interest areas for the signal location, group location, vehicle location, and vehicle track levels. These interest areas are restricted to the area of the sensor attached to the node. The integrating node has interest areas on the vehicle track and pattern track levels that span the entire monitoring area.

Since communication is only outward from a worker node, each worker node in this strategy decides what activities to perform based solely on its own sensory data. In effect, this means that the local activities of worker nodes are determined without a view of the developing answer map (as was the case in the four-node configuration).

This strategy is tried with and without internal subgoaling.

Mixed-initiative communication with self-directed control.

In this strategy, the integrating node transmits goals to the worker nodes informing them of its needs. These goals are not subgoaled, but their ratings are not lowered from their normal value. This means that the scheduling of vehicle track level knowledge sources can be biased by the integrating node, but lower level processing is still self-directed.

To implement this strategy the GOAL-RECEIVE:VT:VT and HYP-REPLY:VT:VT knowledge sources are enabled at each worker node and the GOAL-SEND:VT:VT knowledge source is enabled at the integrating node. Appropriate goal send and goal receive interest areas are also supplied to the integrating and worker nodes.

This strategy is run with internal subgoaling of locally generated vehicle track goals.

Mixed-initiative communication with externally-directed control.

This strategy is identical to the above strategy except that internal subgoaling is turned off and received goal subgoaling is turned on. This means that activities at all levels of worker nodes is controlled by the integrating node.

Mixed-initiative communication with both self-directed and externally-directed control.

This strategy is a combination of the above two strategies. Both internal and received goal subgoaling are enabled. Each worker node attempts to achieve whatever goals have the highest rating, regardless of their source.

5.3.5 Results of the five-node network experiments on the straight vehicle environment.

Each of the organizational problem solving strategies was run using the straight vehicle environment. The results are shown in Table 12. Internal subgoaling again results in a reduction in the number of network cycles and a significant reduction in the number of transmitted hypotheses.

Comparing the performance of the five-node network in the voluntary, self-directed problem solving strategy with the performance of the single node network shows a speed-up of 76 percent without subgoaling and 48 percent with subgoaling. This means that the distributed interpretation without the use of subgoaling took 1.2 times as much processing as the centralized interpretation without subgoaling. With subgoaling the distributed interpretation took 2.6 times as much

Table 12: Summary of Five-Node Network Experiments.

SUMMARY OF FIVE NODE EXPERIMENTS

Environment	Problem Solving Strategy	Internal Subgoal Threshold	Received Hyp Subgoal Threshold	Received Goal Subgoal Threshold	Network Cycles	Sent Hyps	Sent Goals	Actual Track Belief	Ghost Track Belief
straight	VH,SD	10000	10000	10000	37	43	0	4879	*
straight	VH,SD	0	10000	10000	27	20	0	4499	*
straight	MH,SD	0	10000	10000	25	18	14	4415	*
straight	MH,ED	10000	10000	0	40	33	30	4518	*
straight	MH,S&ED	0	10000	0	29	22	18	4499	*

Strategies:

VH	Voluntary Hypothesis Communication
MH	Mixed-Inititive Hypothesis Communication
SD	Self-Directed Control
ED	Externally-Directed Control
S&ED	Combined Self- and Externally-Directed Control

The complete actual track is generated at Node 5.

* Ghost track hypotheses less than 5 time frames long are not considered.

processing. These values are comparable with the four-node network.

Whether the network used voluntary or mixed-initiative communication of hypotheses had little effect on the number of network cycles required to generate an answer when subgoaling was used. As with the four-node network experiments, whether the strategy was self-directed or externally-directed had a much greater effect on network performance. The completely externally-directed strategies performed much worse than the completely data-directed strategies, with the combined strategies in between.

Distraction revisited.

Once again the more externally-directed organizational problem solving strategies performed poorly on the straight vehicle environment. In this case the information received by the integrating node (Node 5) from Node 1 causes it to make inappropriate coordination decisions for the other three worker nodes. Instead of distracting hypotheses directly from Node 1, this time the distraction takes the indirect form of distracting goals received from Node 5.

The mixed-initiative communication with externally-directed control experiment was rerun with the knowledge sources at Node 1 disabled. Again the loss of Node 1 improved the performance of the network by eliminating its distracting influence. The number of network cycles was reduced from 40 with Node 1 to 29 without Node 1. In this case the belief of the actual pattern track hypothesis increased from 4499 to 4796. The network again performed much better without the distractions from Node 1.

5.3.6 Comparing the four-node and five-node architectures.

When the additional processing provided by the fifth node is taken into account, the performance of the lateral four-node network was basically identical with the performance of the hierarchical five-node network in comparable self-directed experiments. Normalizing the number of network cycles by multiplying the four-node experiment cycles by four-fifths results in the side-by-side comparison of similar experiments shown in Table 13. The five-node network does appear to perform better than the four-node network in the externally-directed strategies. When a node in the four-node network receives distracting information it generally processes it to the pattern track level before resuming work on its own lower level hypotheses (due to the generally

Problem Solving Strategy	Internal Subgoaling	Normalized Four-Node Network Cycles	Five-Node Network Cycles
VH,SD	no	43.2	37
VH,SD	yes	26.4	27
RH,SD	yes	25.6	25
RH,ED	no	66.4	40
RH,S&ED	yes	60.0	29

Table 13: Network Cycle Comparison of the Four- and Five-Node Experiments.

Strategies:

VH	Voluntary Hypothesis Communication
RH	Requested/Mixed Initiative Hypothesis Communication
SD	Self-Directed Control
ED	Externally-Directed Control
S&ED	Combined Self- and Externally-Directed Control

higher belief associated with higher abstraction levels). A worker node in the five-node network only processes distracting information to the vehicle track level, and then sends the information on to the integrating node. Thus the worker node can resume its activities sooner than a node in the four-node architecture. The integrating node, while distracted at the vehicle track level, is not synthesizing low level data and is similarly less affected by the distracting information.

While the experiments reported in this chapter indicate the flexibility of the distributed vehicle monitoring testbed as a research tool for exploring varied problem solving network architectures and organizational structures, they do not provide sufficient data for drawing any conclusions on the particular benefits of particular organizational structures. These experiments were performed with a simple grammar in a single environmental scenario with fairly unrestricted communication. In the next section, the potential for using the testbed to begin to explore the effectiveness of different organizational structures in different problem solving situations is discussed as well as the problems facing such an investigation.

5.4 Future Directions in Evaluating the Effect of Organizational Structuring

The distributed vehicle monitoring testbed can be a useful tool in evaluating the effect of organizational structuring in different distributed problem solving situations. The simple experiments reported in this chapter are a tiny, first step in that direction. Additional experiments varying the amount of internode communication, the power of the knowledge sources, the synthesis paths through the blackboard, and the complexity of the grammar and environment are needed.

Particularly important is exploration of larger networks. A four or five node network simply has too few nodes for organizational structuring decisions to have a significant impact. Experiments with tens or even hundreds of nodes are needed before the full effect of organizational structuring will be seen.

Larger networks will involve more complex organizational structures and will provide sufficient redundancy to explore reliability issues when nodes and communication channels fail as well as the use of node skepticism. However, before an extensive set of experiments is performed, a means of quantifying the effectiveness of a particular organizational structure in a particular environmental setting is needed.

The bottom-line of any organizational effectiveness measure is whether the network is able to perform the task. There are at least three components to such a performance measure: response time, accuracy, and network robustness.

In the distributed vehicle monitoring task, <u>response time</u> determines the timeliness of the answer map. Figure 70 illustrates a typical response time acceptability function. If the answer map lags real-time by less than t:1, network performance is perfectly acceptable. As the response time goes past t:1, the acceptability drops. If the response time exceeds t:2, the network performance is unacceptable.

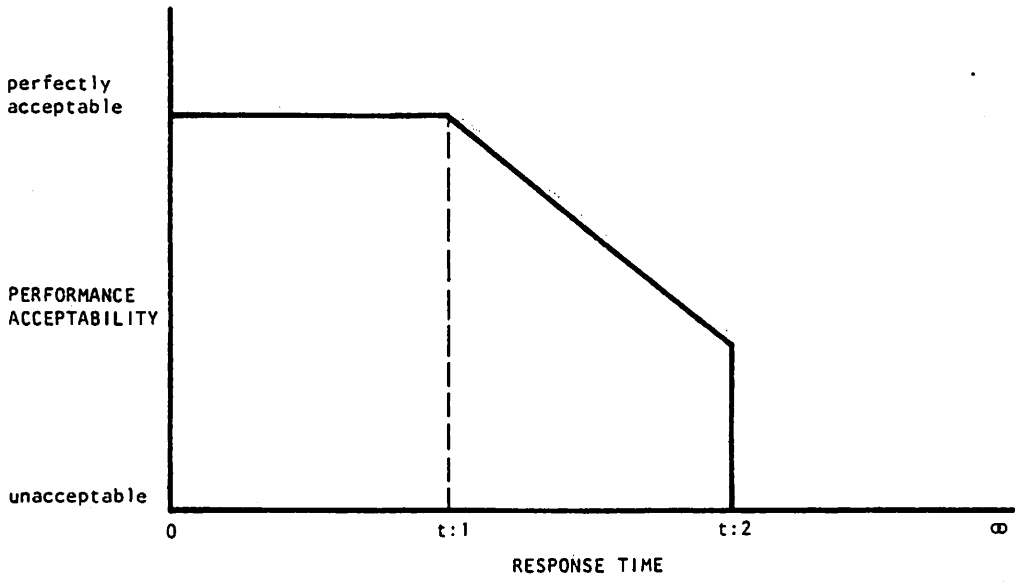


Figure 70: Typical Network Response Time Requirements.

A typical response time acceptability function. If the answer map lags real-time by less than t:1, network performance is perfectly acceptable. As the response time goes past t:1, the acceptability drops. If the response time exceeds t:2, the network performance is unacceptable.

In assessing the <u>accuracy</u> of the distributed vehicle monitoring network, four kinds of errors must be taken into account:

1. missing a pattern;
2. detecting a non-existent pattern;
3. incorrectly identifying a pattern;
4. inaccurately locating a pattern.

Each kind of error could have a different associated performance penalty.

<u>Network robustness</u> in distributed problem solving networks is somewhat like insurance; if it isn't needed, resources expended for network robustness have been wasted. An important aspect of the distributed vehicle monitoring task is that the network be able to withstand a certain degree of component failure. The network robustness

measure must take into account the potential robustness that is achieved. If the potential robustness is below design levels, the organizational performance should be penalized.

While the above components are useful for analyzing organizational performance in retrospect, a measure which can be applied while problem solving is underway is needed. One approach is to measure the expenditure of network resources and the progress of network problem solving.

Since limited node interaction is the driving characteristic of distributed problem solving, use of the communications channel has a major impact on the effectiveness of an distributed problem solving organization. Unfortunately, it is difficult to assess this impact. Tenny, discussing the problem in the context of formal decentralized control theory, states that "the inclusion of communication costs or constraints is extremely difficult to do analytically in a way which permits reasonable solutions to emerge" [TENN79]. The reason is that the communications channel is a <u>non-storable resource</u>. If the channel is unused during a period of time, that potential usage is lost forever; it cannot be saved for a future eventuality. Given that restricted communication is such a significant aspect of distributed problem solving, it might appear that the network should attempt to use the channel as much as possible to avoid wasting such a precious resource. On the other hand, if the communication channel is saturated with routine interactions, a critical message may not be communicated in a timely fashion.

One approach to measuring the "cost" of using the resource is to assign a usage cost to every message. However, such an approach does not recognize the non-storable nature of the communications resource. If the message can be sent without affecting the communications of other nodes its cost is essentially nil.

A contention costing approach where the organization is penalized for interfering messages would seem to be a better choice. One possible scheme would charge a fixed penalty for each interference. However, this scheme does not distinguish between interfering "routine" communications and interference with a message of critical importance. Another approach is to vary the penalty based upon some measure of the impact of each message. A low impact message is penalized for any high impact messages which it blocks. The penalty is the difference between the blocked messages' impacts and the low impact message. High impact messages are not penalized for blocking lower impact messages.

Processing is also a non-storable resource, and measuring processing expenditures requires a similar approach. A node should not be penalized for performing superfluous activities if it would otherwise be idle or for performing redundant activities that are needed to maintain specified network performance levels. Evaluating appropriate expenditures of processing resources requires detailed information about the relationship among the processing activities in the network and about the activities necessary to generate a solution effectively.

In short, evaluating the performance of an organizational structure in a functionally accurate, cooperative distributed problem solving network is not simply a matter of "metering" resource expenditures with a goal of minimizing their use. Instead, a useful evaluation requires understanding if all available resources are spent wisely in generating the solution. This requires a deep understanding of the relationship among activities in the network — an understanding that is central to not only the evaluation of organizational structures, but to the detection and correction of network hardware and problem solving errors and to load-balancing among nodes.

Goals, like dreams, are not always realized.

— Ya. Z. Tsypkin

CHAPTER VI
THE END OF A BEGINNING

In this chapter we first summarize what has been covered and then look at likely avenues for continued effort. We close the dissertation with a discussion of the problem of developing an organizational designer and the potential for transferring organizational structuring ideas from business and management organizations and other "natural distributed systems" to distributed problem solving networks.

6.1 A Look Back

We began with the problem of achieving coordinated activity among the nodes in a distributed problem solving network. Internode communication in these networks was seen as the major issue since it is both limited and potentially unreliable. Limited node interaction makes it infeasible to keep every node fully abreast of the information possessed by other nodes in the network or of their present and planned activities. Node activity is, by necessity, loosely-coupled.

The contract net and self-directed approaches to the network coordination problem were discussed. Both approaches attempt to deal with the problem of obtaining coordinated node activities with decentralized and interaction-limited decisionmaking. A general view of coordination that integrates both approaches was described. However, all these approaches lacked a shared high-level view of how the network is attempting to solve the problem and of the general roles and responsibilities of each node. Adequate coordination is difficult without such a shared view.

Organizational self-design was advanced as a multilevel approach to coordinating a distributed problem solving network. An organizational structure provides a shared high-level view of the information and control relationships among the nodes. Each node is responsible for elaborating these relationships into precise activities to be performed by the node. The idea is to include in the organizational structure those decisions that are not quickly outdated and that pertain to large numbers of nodes. The development and use of an organizational structure as a guide for coordination is considered to be less complex and dynamic than directly coordinating every activity in the network.

Organizational structuring decisions often have to be made using incomplete and inaccurate information about the problem solving situation. There can be times when the decisions are inappropriate for particular nodes or for the network as a whole. Therefore, the organizational structure is taken to be a guide rather than a rigid structure. The network is viewed as a society of skeptical nodes working within the framework of an organizational structure but always alert for signs of trouble.

A major theme throughout the dissertation is that sophisticated local control capabilities are necessary to perform these activities. Complex network behavior cannot be obtained from large numbers of simple nodes. The "magic" is in the glue that binds the nodes together. And that glue must come from the individual nodes themselves.

A framework providing the necessary local control capabilities was developed and implemented in the Distributed Vehicle Monitoring Testbed, a simulated distributed interpretation network designed for empirical evaluation of network coordination strategies. The control framework is based on the Hearsay-II architecture, extended to accommodate goal-directed control and a local node planner. This architecture allows each node to be responsive to both data and goals (representing requests to create or communicate particular types of data) and to plan sequences of activities to achieve its goals.

The activities of each node's local planner and communication knowledge sources are influenced by a set of interest area and subgoaling specifications. These nonprocedural specifications can be inspected and dynamically modified to change the behavior of the node. Interest area specifications form the interface between the node's local control machinery and an organizational designer.

The capabilities of the framework were illustrated with a number of testbed experiments using different organizational structures in one-, four-, and five-node distributed networks. The single node experiments demonstrated that the goal-directed Hearsay-II architecture and local node planner, and particularly the use of subgoaling as a focusing technique, can be used to direct the problem solving activities of a node through the use of goals.

The four-node and five-node network experiments indicated that different organizational structures and cooperation strategies do make a difference in network problem solving performance. These experiments also illustrated the problem of distraction, where one node communicated purely incorrect hypotheses that temporarily drew the other nodes away from working on the correct solution. The experiments suggested that organizational structures which divide the effort spent on distracting information among nodes (as was the case with the five-node hierarchical organization) are less affected by the distracting information.

6.2 A Look Ahead

Because this work represents only the first step toward a distributed problem solving network that is able to design, implement, and change its own organizational structure, a number of open research issues remain. Many of these issues involve the framework implemented in the Distributed Vehicle Monitoring Testbed.

Perhaps the most obvious area of future work involves applying the testbed and the framework to larger network experiments. As discussed in Chapter V, a four or five node network simply has too few nodes for organizational structuring decisions to have a significant impact. Experiments with tens or even hundreds of nodes are needed before the full effect of organizational structuring will be seen. Larger networks will require more complex organizational structures and will provide sufficient redundancy to explore reliability issues when nodes and communication channels fail as well as the use of node skepticism (see below).

Additional experiments also need to be performed in which the many parameters in the testbed are varied. In particular, the changes in problem solving behavior as internode communication is restricted should be very interesting. (Recall that the experiments presented in Chapter V used a fairly unrestricted communication channel.) Exploration of these issues using the testbed is just beginning.

Increasing the sophistication of the local node planner is an area with the potential for considerable rewards. Major improvements in distributed problem solving activity were gained from a very simple planner. The goal-directed Hearsay-II architecture provides a rich source of planning information that could be used to plan the interpretation activities of each node. Of course as planning capabilities are increased, techniques for adequately costing and controlling the overhead associated these activities are needed.

The goal satisfaction problem (and at the macro level the distributed "stopping" problem) is a related area requiring additional work. These problems were discussed in Section 4.3.3.

A limitation in the testbed architecture is the inability of a node to represent negative evidence for a particular hypothesis internally or to communicate negative evidence to other nodes. There is an important distinction between information indicating that a particular hypothesis is inconsistent with other hypotheses and the lack of information altogether. The use of a single belief value in Hearsay-II (and other single belief systems) leads to a confused representation of this distinction. The ability to communicate negative information appears to be very useful in a distributed problem solving environment where nodes can develop incorrect hypotheses due to incomplete local information.

For example, suppose a testbed node has knowledge that it is very unlikely for vehicles to move through a particular region of its immediate environment and that this information is not known by other nodes. (For now, we ignore how this information might be represented and applied within the node.) This node receives a vehicle track hypothesis from another node that is contained within this "unlikely" region. How can the node inform the other node that, from its local perspective, this track appears to be incorrect?

One approach is to use a two-valued belief system for hypotheses, one value representing the support for a hypothesis and the other representing negative evidence for the hypothesis's existence [BARN81, GARV81, LOWR82]. Use of such a belief system in the testbed would require extensions to the basic testbed architecture, the network problem solving state measures, the knowledge sources, and the planner but would allow negative evidence to be developed and exchanged among nodes.

As was illustrated in the simple example in Section 3.2, it is not always sufficient to communicate a hypothesis's characteristics and its belief to another node. A description of the reasoning involved in creating the hypothesis can also be needed for the receiving node to decide what new, independent information is represented in the hypothesis. Although dependent versus independent information is an issue for centralized problem solving systems, the overlapping and partially dependent local views that arise in distributed problem solving networks increase its importance.

A neglected area of testbed development has been the communication knowledge sources. The communication knowledge sources in the testbed are very primitive. Transmission knowledge sources do not modify their transmission criteria based on channel loads and do not evaluate whether or not a particular candidate for transmission could be deduced by the receiving node from previous communications with that node. Since the communications resource is so critical to a distributed problem solving network, efforts spent here should result in substantial improvements in network problem solving.

As noted by Sacerdoti, language understanding research is directly relevant to distributed problem solving networks due to the limitations on communication [SACE78]. Particularly appropriate is the computer implementation of Searle's model of speech acts [SEAR70] by Allen and Cohen. Allen's work focused on the recognition of the intended goals by the listener, and Cohen's work focused on the planning of what to say to achieve particular communication goals [ALLE79, COHE78]. Communication knowledge sources that implement parsimonious high-level internode communication have considerable potential in the distributed environment. In fact, much of the planning activity required for high-level message generation and recognition may already exist on the data and goal blackboards.

There were occasions (in the early morning hours of testbed implementation) when the functionally accurate, cooperative capabilities of the testbed worked all too well, and the network generated reasonable solutions despite major conceptual and programming errors. Once gross "holes" among the knowledge sources were eliminated, the network did not crash or fail to develop a solution but rather tended to use its functionally accurate capabilities to work around conceptual and programming errors. Instead of obviously failing, the network simply took longer to generate the solution. Identifying such errors required detailed (and tedious) inspection of the individual activities of each the node. When a number of functionally accurate, cooperative nodes are combined to form a large network, the problem of monitoring and debugging network activities becomes a major practical issue. Bates, Wileden, and Lesser describe an event monitoring facility (in which high-level events can be specified using combinations of primitive individual node activities) and its potential application in the testbed [BATE81].

The major motivation for this dissertation has been the eventual development of an organizational self-designer for a distributed problem solving network. Now that the computational framework to support an organizational self-designer has been built, the problem of providing an organizational designer can be addressed. A discussion of organizational design for distributed problem solving networks concludes this chapter.

The last area of future work on this list is exploration of the advantages, disadvantages, and control of node skepticism in large distributed problem solving network networks.

6.3 Some Thoughts on Organizational Design for Distributed Problem Solving Networks

You have to tell the truth the way you see it. And yet you have to be tolerant of the fact that neither you nor the man you are arguing with is going to get it right.

— Jacob Bronowski

During the process of developing the computational machinery described in the preceding chapters, the problem of actually performing the organizational design was not completely forgotten. Now that that machinery is in place, it is tempting to consider the issues facing organizational design for distributed problem solving networks. In the remaining pages, we give in to this temptation and briefly speculate on the future of organizational design in distributed problem solving networks.

A major obstacle in organizational design for distributed problem solving networks is that networks of significant size and complexity have not yet been constructed. We are in the difficult situation of developing an organizational structuring repertoire for networks whose properties can only be estimated.

Fortunately, organizational structuring is not unique to distributed problem solving networks. In the next sections, organizational structuring concepts used in business and management organizations and identified in natural and social systems is briefly reviewed, with an emphasis on the potential application of those concepts to distributed problem solving networks.[1] As we will see, an important issue is identifying the similarities and diversities between the organizational requirements of these systems and distributed problem solving networks. Some speculations on the similarities and diversities are made in the concluding section.

6.3.1 Business and management organizations.

Organizational design decisions are faced regularly in business organizations where the pressures of efficiency can be severe. In response to the need to develop effective organizational structures, a substantial body of knowledge has been developed. Galbraith in Chapter 2 of Organization Design presents a concise review of the development of management theory [GALB77]. We begin with a brief summary of his review.

Classical management theory has its roots in the concept of division of labor, where the overall task of the organization is divided into subtasks, each to be assigned to an individual. Rather than having each individual perform all the organization's activities on a portion of the overall task (the approach suggested for distributed problem solving networks in Chapter I), it was usually recommended that the task be partitioned such that each individual performed only a portion of the activities on the entire task. Such horizontal division of labor was based on the premise that each individual would become highly skilled at his smaller, more specialized task, that the transition time between

1. Fox presents an extended discussion of the application of organizational theory to the problem of developing large and complex software systems [FOX79].

activities would be minimized, and that more efficient and that specialized tools would be developed for these smaller tasks.

While horizontal division of labor often increased the production per individual, it also increased the interdependence among individuals. Rather than only affecting his own production, each individual affects the production of those who rely upon his activities. To coordinate the various activities, a second form of division of labor, termed vertical division of labor, was developed. In this structure, the task of coordinating the basic activities involved in the overall task was assigned to separate, managerial specialists. As Galbraith notes, "the horizontal division of labor must meet the economic constraint of increasing output by an amount sufficient enough to pay for someone who does none of the work but must be present to coordinate the work" [GALB77].

Overspecialization is also a danger of horizontal division of labor. While some specialization can lead to increased overall productivity, overspecialization causes the productivity increases to be offset by motivational problems (a lack of job satisfaction) and idle time (if the specialty is not in constant demand).

An important issue with vertical division of labor is how to partition the managerial activities and decisionmaking responsibilities. Two fundamental principles were developed. The first principle, unity of command, states that each individual should have a single superior responsible for resolving conflicts in his activities. The second, scalar, principle states that authority should flow in a unbroken line from the chief executive to the lowest worker. These two principles led to a representation of the organization as a hierarchy of authority. An important variable in such a hierarchy is the <u>span of control</u> of a supervisor, the number of subordinates that can be coordinated by a single supervisor.

A problem with the purely hierarchical authority structure is that it stresses responsibility relationships and not the sharing of expertise (such as an expert in energy savings making energy-related decisions throughout the organization). Classical management theory's solution was the formation of the <u>line-staff</u> organization. Experts (staff) were made available as advisers to the (line) managers who remained responsible for making the decisions. (In practice the distinction between line and staff roles tends to be fuzzy.) An important design variable in the line-staff organization is where the expertise is to be made available to the line managers. If the expertise is made available at the top of the authority pyramid, the organization is termed "centralized". Organizations where the expertise is made available at the lower levels are termed "decentralized". The appropriate degree of centralization/decentralization was dependent on the organizational situation. However, decisionmaking in decentralized organizations tended to be more fragmented and uncoordinated. Partitioning the organization into departments was viewed as a means of localizing the uncoordinated effects of decentralized decisionmaking.

So what does classical management theory have to say about organizational structuring in distributed problem solving networks? The basic division of labor and availability of skilled expertise are sharply different in distributed problem solving networks and the organizations addressed by classical management theory. In addition, it is much more likely that each node in the distributed problem solving network will spend a portion of its effort dealing with the organizational coordination rather than having a clear division between worker nodes and manager nodes.[2] The key issue is how closely the distributed vehicle monitoring task fits the organizational tasks considered by classical management theory. Unlike many organizations that are concerned with producing goods or services, the testbed's productive energies are spent attempting to acquire a sufficiently global view of activity in the environment for deciding "what is out there". In a sense, the testbed has a relatively trivial production component and a substantial internal information processing component. It is the essence of the internal information and decision processes of business organizations operating in uncertain task environments.

Galbraith lists five design strategies that can be used by an organization to handle the increased information processing requirements of decision making caused by uncertainty [GALB77]:

<u>Environmental Management</u>	-- An organization can reduce its need for information processing by modifying the environment in which it resides.

2. An intriguing question is whether distributed problem solving networks should be constructed with sensorless "manager" nodes whose computational responsibility is coordinating the other nodes or whether all nodes should have the capability of receiving sensory information and of coordinating the network. The latter approach would appear to make more use of system resources since nodes with little sensory information could undertake greater responsibility for coordinating and monitoring the performance of the network. In this structure the notion of redundancy of potential command [McCU65] applies not only to the problem solving but to organizational coordination as well.

Creation of Slack Resources	-- An organization can reduce its need for information processing by decreasing its level of performance by using additional, slack, resources (such as time, equipment, and personnel) or by reducing the overall quality of its output.
Creation of Self-Contained Tasks	-- An organization can reduce its need for information processing by choosing another decomposition in which tasks are more self-contained.
Improving Vertical Information Flow	-- An organization can increase its capacity to process information by collecting information at the points of origin and directing it to the appropriate individuals in the organization and by the use of abstraction.
Creation of Lateral Relations	-- An organization can increase its capacity to process information by placing in direct contact individuals which share a common problem.

Galbraith draws upon Simon's work which recognized the limited information processing capabilities of individuals [SIMO57, SIMO69]. Termed bounded rationality, this limitation applies to both the amount of input information which can be effectively used to make decisions and the amount of control which can be effectively exercised by an individual. Bounded rationality has severe implications on the quality of decision making when a large amount of uncertainty is present, for "the greater the task uncertainty, the greater the amount of information that must be processed ... to achieve a given level of performance" [GALB73]. A motivation for variations in organizational structures is to provide additional information processing capacity to handle the greater uncertainty within the bounded rationality of the organization's individual members.

While several of Galbraith's design strategies are inappropriate in the distributed vehicle monitoring domain (in particular environmental management and the procurement of additional network resources), they do prescribe the information processing and communication options available to the organizational designer. Particularly relevant is Galbraith's warning that "the organization must adopt at least one of the five strategies when faced with greater uncertainty. If it does not consciously choose one of the five, the slack, reduced performance standards will happen automatically" [GALB77].

An important issue in organizational design for the testbed is the speed at which the vehicles move in the environment. Such rapid changes in the environment require a matching speed of organizational change in the testbed. (unless sufficient slack resources are allocated throughout the network to handle congested vehicle traffic without organizational change -- an uninteresting and potentially expensive approach). An interesting research question is whether an organizational structure can be developed for the testbed that can be adapted to fit the changing vehicle traffic or whether new organizational structures have to be designed and implemented when the environment changes substantially. Another question is whether the organizational situations encountered in the distributed vehicle monitoring domain are sufficiently recurrent that programmed organizational decisions can be used. It has been recognized that organizations operating in uncertain environments perform poorly if they rely too heavily on programming [LAWR67]. Organizations that perform well in uncertain environments tend to have a high level of general knowledge throughout the organization and a low reliance on predetermined formal rules. Hagafors presents a concise survey of the issues related to programmed versus unprogrammed organizational decisionmaking [HAGA82].

In the next section, we briefly look at the organization of biological and social systems with particular attention to the problem of organizational design in uncertain and rapidly changing environments.

6.3.2 Biological and social systems.

The work of Beer on "managerial cybernetics" [BEER78, BEER79, BEER81] forms a transition between business and management organizational systems and biological and social systems. Beer's goal is to bring cybernetic validity to the design of new managerial control techniques that allow the organizational structure of the firm to be adapted on a "second-by-second" basis. He suggests that the human nervous system provides a model for coordinating and regulating a complex organization and that management organizations should embody this same logical structure. Although Beer cautions against simply using the neural model as a metaphor for organizing, we survey here only a few of the model's high-level principles and do not suggest that the structure of any organizational self-designer for distributed problem solving networks should be directly patterned after the model.

Much of the organizational philosophy presented in the preceding chapters is in the style of Beer's model. In particular, he stresses the need for autonomous elements (our network nodes) that are able to function with minimal intervention from higher (organizational) level elements. While our main motivation for autonomous nodes stemmed from the limited and unreliable nature of internode communication, autonomy in Beer's model stems from the need to reduce the control required of higher level organizational elements.

Beer also discusses the need for interpreting higher-level (organizational) instructions into more detailed patterns of activity at the lower levels. This requires some sophistication at the lower levels along the same lines as the coordination framework developed for the testbed. He describes the need for organizing the system, not in full, but only somewhat (our high-level organizational structure) and then letting the lower level components tune the dynamics of the system in the directions it needs to go. Beer also notes the importance of lower-level components in initiating feedback information to the higher-level (organizational) components to be used in making predictive plans. This aspect was not developed in the testbed coordination framework. As mentioned in Chapter III, Beer describes the need for balancing the tension between the performance of potential elemental actions (which he terms the <u>operational force</u>) and the need for system viability and coordination (which he terms the <u>coherence force</u>). This tension serves as the basis for node skepticism.

Beer's cybernetic organization model also suggests some characteristics for an organizational self-designer. In particular, the organizational self-design component is viewed as an autonomous system in its own right and not as a component of the basic problem solving network. In his model, each level in the organization is an autonomous system and has the same basic structure. The model is recursively applied at all organizational levels, with the coordination components of one level being the basic working components at the next higher level. Rather than constructing a classical hierarchical organizational structure, Beer advocates the recursive application of his model at each level.[3] Where interaction in a hierarchy tends to be greater between levels, Beer suggests that interaction in the recursive structure is greater laterally, within each level. In addition, each level is an autonomous system in its own right, able to cope with changes either above or below, rather than a mere component in the overall organization.

Beer's approach is, to say the least, interesting. Even more exciting is the potential for using the testbed to experiment with his ideas and to compare and contrast their performance with the performance of more traditional organizing techniques.

The work of Crane suggests that neural organization and functioning can be better understood through the use of sound analogies with social organization and functioning [CRAN78]. In a sense, the direction of

3. Beer does not speak highly of organizational charts, stating that they "specify 'responsibility' or the 'chain of command', instead of the machinery that makes the firm tick" [BEER81].

knowledge transfer in Crane's work is in the opposite direction of Beer's approach. However, many concepts are common to both approaches. Autonomy of individual elements is central to Crane's approach as is the tension between local and system-wide activity decisions. Crane states that "an individual may undertake an activity because he is capable of doing it, or because it is in the best interests of society that he do it; at the same time he may not want to do it" [CRAN78]. We have discussed previously (in Section 3.2) Crane's notion of dual-directed control, the important balance between top-down and bottom-up control, and its relationship to node skepticism. Crane also stresses the use of information brokers to bring together producers and consumers of information [CRAN80]. Such "broker nodes" might prove useful in very large distributed problem solving network organizations.

6.3.3 Parting thoughts.

Local autonomy and the elaboration of high-level decisions under the tension of local and organizational demands is a common theme in these approaches to organizational structuring. Local elements are not simpletons carrying out orders received from above, but are sophisticated systems in their own right, making sophisticated decisions and understanding the reasoning behind these decisions. The coordination framework developed for the testbed, with its decisionmaking machinery provided by the goal-directed architecture and planner, appears to be on target with this viewpoint.

But what of the future development of an organizational self-designer? Crane states that "computers, brains, and societies are each intimately involved with information, language, and meaning, and insight into any one area can potentially help to illuminate the other two" and he predicts that there will be a "lively exchange of insights among the organizational structures of individuals, societies, and computers" [CRAN78]. At the level of general insights, Crane's statement is undeniably accurate. The important question, however, is how deeply does this relationship hold. Are the organizational problems and constraints facing business organizations similar enough to those facing society or distributed problem solving networks or those addressed in the brain that detailed techniques can be transferred from one area to the others? Can an organizational self-designer for a distributed problem solving network be constructed from borrowed parts?

Each area should look to the others -- not for the ultimate solutions, for they do not yet exist -- but for fresh and illuminating ways of looking at common problems and for identifying and understanding the unique characteristics of each area. The designers of an organizational designer for distributed problem solving networks do not need to reinvent the principles of organizational design, they are fortunate indeed to have a rich source of ideas to draw from, but they do need to identify the problems unique to coordinating these systems and to add their own insights and contributions to the science of organization.

A P P E N D I X   A

ENVIRONMENT FILE DESCRIPTION

This appendix contains a description of an environment file, the input file for a distributed vehicle monitoring testbed run.

```
                         ##ENVIRONMENT-FILE##
;
; *********************************************************************
; *********************************************************************
; **                                                                 **
; **                          TEMPLATE                               **
; **                      ENVIRONMENT FILE                           **
; **                                                                 **
; *********************************************************************
; *********************************************************************
;
;     This template file, last modified APR-26-1982, defines the
; format of environment files.  An environment file contains the
; following major sections:
;
; DATA DEFINITIONS      defines the basic data types used in
;                       environment files;
;
; ENVIRONMENT-ID AND COMMENTS
;                       specifies the name of the environment file and
;                       a brief description;
;
; SYSTEM DATA           specifies some basic parameters of the testbed;
;
; GRAMMAR DEFINITION    specifies the grammer: its name, the event
;                       relationship among event classes at different
;                       blackboard levels, and the maximum velocity and
;                       acceleration of vehicles;
;
; NODE DEFINITIONS      specifies the node configuration: where each
;                       node is located and how it behaves;
;
; SENSOR DEFINITIONS    specifies the sensor configuration: where each
;                       sensor is located and how it behaves;
;
; COMMUNICATION RELIABILITY DATA
;                       specifies the chance of message loss between
;                       nodes;
;
```

```
; CONSISTENCY DATA      specifies what hypotheses are considered
;                       consistent by the testbed (defines the
;                       consistency blackboard);
;
; ENVIRONMENT DATA      specifies what is sensed in the environment.
;
;
; *********************************************************************
;
;                          DATA DEFINITIONS
;
; *********************************************************************
;
;                          INTEGER CONSTANTS
;
;
;     begin-coordinate
;     begin-time
;     end-coordinate
;     end-time
;     max-acceleration
;     max-velocity
;     reception-latency
;     receptions-per-period
;     transmission-latency
;     transmissions-per-period
;
; The following constants are not explicitly included in the
; environment file but are computed from the supplied information:
;
;     #-event-classes
;     #-interest-areas
;     #-knowledge-source-sets
;     #-node-classes
;     #-nodes
;     #-sensor-classes
;     #-sensors
;     #-topological-roles
;
```

```
;                              BASE DATA TYPES
;
;   belief            = an integer in the range [0..10000]
;   consistency       = an integer in the range [0..10000]
;   coordinate        = an integer in the range
;                           [begin-coordinate..end-coordinate]
;   credibility       = an integer in the range [-10000..10000]
;   event-class       = an integer in the range [1..#-event-classes]
;   event-class-list  = a list of event-classes or *all
;   focusing-weight   = an integer in the range [-10000..10000]
;   group-class       = an event-class
;   id                = a string of characters
;   interest-area     = an integer in the range [1..#-interest-areas]
;   knowledge-source-set
;                     = an integer in the range
;                           [1..#-knowledge-source-sets]
;   level             = an element of the set
;                           {sl st gl gt vl vt pl pt}
;   level-list        = a list of levels or *all
;   level-size-weight-list
;                     = a level and a size-weight-list
;   location          = a pair of coordinates,
;                           (x-coordinate y-coordinate)
;   node              = an integer in the range [1..#-nodes]
;   node-class        = an integer in the range [1..#-node-classes]
;   node-list         = a list of nodes or *all
;   pattern-class     = an event-class
;   power             = an integer in the range [-10000..10000]
;   probability       = an integer in the range [0..10000]
;   region            = a quadruple of coordinates,
;                           (x-min y-min x-max y-max)
;   region-list       = a list of regions or *all
;   seed              = an integer in the range [0..131072]
;   sensor            = an integer in the range [1..#-sensors]
;   sensor-class      = an integer in the range [1..#-sensor-classes]
;   sensor-list       = a list of sensors or *all
;   signal-class      = an event-class
;   size              = an integer in the range
;                           [1..system-location-range]
;   size-weight-list  = a list of pairs (size weight) or nil
;   threshold         = an integer in the range [-10000..10000]
;   time              = an integer in the range
;                           [begin-time..end-time]
;   time-list         = a list of times or *all
;   time-location     = a pair consisting of a time and a location,
;                           (time (x-location y-location))
;   time-location-list
;                     = a list of time-locations
;   topological-role  = an integer in the range [1..#-topological-roles]
;   vehicle-class     = an event-class
;   weight            = an integer in the range [0..10000]
```

```
;
; ***********************************************************************
;
;                        ENVIRONMENT-ID AND COMMENTS
;
; ***********************************************************************
;
[  "environment-id"
   "first line of comments"
   ...
   "last line of comments"  ]
;
; Comments are printed out at the beginning of testbed output.
;
; ***********************************************************************
;
;                               SYSTEM DATA
;
; ***********************************************************************
;
;                               RANDOM SEED
;
[  seed  ]
;
; Initializes the pseudo-random number generator at the start of the
; run.
;
;                        GLOBAL TESTBED PARAMETERS
;
[  scheduler-power
   scheduler-belief-weight
   reflection-threshold
   scheduler-threshold
   modification-threshold
   knowledge-source-instantiation-threshold
   goal-weight
   internal-subgoal-threshold
   received-hypothesis-subgoal-threshold
   received-goal-subgoal-threshold
   hypothesis-insertion-threshold
   goal-creation-threshold
   transmission-latency
   transmission-rate
   reception-latency
   reception-rate
  ]
;
```

```
; **********************************************************************
;
;                           GRAMMAR DEFINITION
;
; **********************************************************************
;
;                               GRAMMAR-ID
;
[  "grammar-id"  ]
;
; The name of this grammar.
;
;                  PATTERN-CLASS TO VEHICLE-CLASS RELATION
;
[  (pattern-class ((vehicle-class location)
                    (vehicle-class location)
                    ...
                    (vehicle-class location)))
   (pattern-class ((vehicle-class location)
                    (vehicle-class location)
                    ...
                    (vehicle-class location)))
   ...
   (pattern-class ((vehicle-class location)
                    (vehicle-class location)
                    ...
                    (vehicle-class location)))  ]
;
; The location of a vehicle in a pattern is specified relative
; to the center of the pattern.
;
;                   VEHICLE-CLASS TO GROUP-CLASS RELATION
;
[  (vehicle-class (group-class group-class ... group-class))
   (vehicle-class (group-class group-class ... group-class))
   ...
   (vehicle-class (group-class group-class ... group-class))  ]
;
;                   GROUP-CLASS TO SIGNAL-CLASS RELATION
;
[  (group-class (signal-class signal-class ... signal-class))
   (group-class (signal-class signal-class ... signal-class))
   ...
   (group-class (signal-class signal-class ... signal-class))  ]
;
```

```
;                         MAX VELOCITY AND ACCELERATION
;
;
[  max-velocity max-acceleration  ]
;
;
; specify the tracking component of the grammar
;
; *****************************************************************
;
;
;                              NODE DEFINITIONS
;
;
; *****************************************************************
;
;
;                           INTEREST-AREA DEFINITIONS
;
;
[  (interest-area (level-list event-class-list time-and-region-lists)
                  (level-list event-class-list time-and-region-lists)
                  ...
                  (level-list event-class-list time-and-region-lists))
   (interest-area (level-list event-class-list time-and-region-lists)
                  (level-list event-class-list time-and-region-lists)
                  ...
                  (level-list event-class-list time-and-region-lists))
   ...
   (interest-area (level-list event-class-list time-and-region-lists)
                  (level-list event-class-list time-and-region-lists)
                  ...
                  (level-list event-class-list time-and-region-lists)) ]

; and time-and-region-lists is a list of pairs,
;
;            ((time-list region-list)
;             (time-list region-list)
;             ...
;             (time-list region-list))
;
; Note: a region is specified relative to a node's center as a list.
;
```

```
;                          KNOWLEDGE-SOURCE-SET DEFINITIONS
;
[  (knowledge-source-set
          (knowledge-source-name goodness resolving-power runtime)
          (knowledge-source-name goodness resolving-power runtime)
          ...
          (knowledge-source-name goodness resolving-power runtime))
   (knowledge-source-set
          (knowledge-source-name goodness resolving-power runtime)
          (knowledge-source-name goodness resolving-power runtime)
          ...
          (knowledge-source-name goodness resolving-power runtime))
   ...
   (knowledge-source-set
          (knowledge-source-name goodness resolving-power runtime)
          (knowledge-source-name goodness resolving-power runtime)
          ...
          (knowledge-source-name goodness resolving-power runtime))  ]
;
; Knowledge sources in a knowledge-source-set are specified in order
;    of decreasing priority for the planner.
; goodness is an integer in the range [0..10000] used to weight the
;    knowledge source rating calculated by the scheduler.
; runtime is a list of two constants (const-1 const-2) used to
;    determine the runtime of a knowledge source as:
;           (const-1) * (#-stimulus-hyps) + (const-2).
;
;                          TOPOLOGICAL-ROLE DEFINITIONS
;
[  (topological-role node-list up-data same-level-data down-data)
   (topological-role node-list up-data same-level-data down-data)
   ...
   (topological-role node-list up-data same-level-data down-data)  ]
;
; where up-data, same-level-data, and down-data are lists of the form:
;
;       ((level-list avg-number-paths)...(level-list avg-number-paths))
;
; and avg-number-paths is a floating-point number.
;
; Topological roles are used to classify nodes for measurement purposes.
;
```

```
;                          NODE-CLASS DEFINITIONS
;
[  (node-class knowledge-source-set cset-region
                      local-processing-interest-areas
                      topological-role subgoal-data)
   (node-class knowledge-source-set cset-region
                      local-processing-interest-areas
                      topological-role subgoal-data)
   ...
   (node-class knowledge-source-set cset-region
                      local-processing-interest-areas
                      topological-role subgoal-data)  ]

; where:
; cset-region is a region specified relative to a node's location;
; local-processing-interest-areas is a list:
;
;          ((interest-area weight)...(interest-area weight))
;
;     of pairs of an interest-area and associated weights;
; subgoal-data is a list of pairs:
;
;     (subgoal-level-list level-size-weight-list)
;
; subgoal-level-list is a list of levels from which subgoals will be
;     created;
; level-size-weight-list is a pair:
;
;     (level-list (list of size-weights));
;
; level-list is a list of levels at which the subgoals will be created
;     for each of the top-level goals;
; size-weight is a pair of size-shrink-factor and a weight;
; size-shrink-factor determines by how much a region will be reduced
;     for the subgoal at that level and the weight multiplies the
;     original top-goal rating to get the new subgoal rating.
;
; Note: If the level-size-weight-list is nil then all the subgoals will
;       be created at all the levels below the top-goal-level which are
;       in the interest area with the same region and rating as the
;       top-goal.
;
```

```
;                                NODE DATA
;
[  (node node-class location sensor-list hyp-send-interest-areas
         hyp-receive-interest-areas goal-send-interest-areas
         goal-help-interest-areas goal-receive-interest-areas
         sensed-times-list)
   (node node-class location sensor-list hyp-send-interest-areas
         hyp-receive-interest-areas goal-send-interest-areas
         goal-help-interest-areas goal-receive-interest-areas
         sensed-times-list)
   ...
   (node node-class location sensor-list hyp-send-interest-areas
         hyp-receive-interest-areas goal-send-interest-areas
         goal-help-interest-areas goal-receive-interest-areas
         sensed-times-list)
] .
; where hyp-send-interest-areas, goal-send-interest-areas and
; goal-help-interest-areas are lists:
;
;     ((node-list (interest-area threshold weight)
;                 ...
;                 (interest-area threshold weight))
;      (node-list (interest-area threshold weight)
;                 ...
;                 (interest-area threshold weight))
;      ...
;      (node-list (interest-area threshold weight)
;                 ...
;                 (interest-area threshold weight)))
;
```

```
; and hyp-receive-interest-areas is a list:
;
;      ((node-list (interest-area threshold weight credibility
;                   focusing-weight)
;                  ...
;                  (interest-area threshold weight credibility
;                   focusing-weight))
;       (node-list (interest-area threshold weight credibility
;                   focusing-weight)
;                  ...
;                  (interest-area threshold weight credibility
;                   focusing-weight))
;       ...
;       (node-list (interest-area threshold weight credibility
;                   focusing-weight)
;                  ...
;                  (interest-area threshold weight credibility
;                   focusing-weight)))
;
; and goal-receive-interest-areas is a list:
;
;      ((node-list (interest-area threshold weight credibility)
;                  ...
;                  (interest-area threshold weight credibility))
;       (node-list (interest-area threshold weight credibility)
;                  ...
;                  (interest-area threshold weight credibility))
;       ...
;       (node-list (interest-area threshold weight credibility)
;                  ...
;                  (interest-area threshold weight credibility)))
;
; and sensed-times-list is a list:
;
;       ((node-clock-time time-frame-list) ...)
;
```

```
; ********************************************************************
;
;                             SENSOR DEFINITIONS
;
; ********************************************************************
;
;                         SENSOR-CLASS DEFINITIONS
;
[  (sensor-class tolerance location-mask signal-mask)
   (sensor-class tolerance location-mask signal-mask)
   ...
   (sensor-class tolerance location-mask signal-mask)  ]
;
; where:
; tolerance is an integer specifying the spatial range of a sensor
; location-mask is a list:
;
;        ((lm11 lm12 ... lm1L)
;         (lm21 lm22 ... lm2L)
;         ...
;         (lmL1 lmL2 ... lmLL))
;
;    of location-mask-values in the range 0..8 (L = 2*tolerance + 1);
; signal-mask is a list, (sm1 sm2 ... smS), of signal-mask-values in
;    the range 0..2 (S = #-event-classes).
;
;                               SENSOR DATA
;
[  (sensor sensor-class location
             prob-true-location prob-true-signal sensor-weight)
   (sensor sensor-class location
             prob-true-location prob-true-signal sensor-weight)
   ...
   (sensor sensor-class location
             prob-true-location prob-true-signal sensor-weight)  ]
;
```

```
; ***********************************************************************
;
;                        COMMUNICATION RELIABILITY DATA
;
; ***********************************************************************
;
;                          HYP COMMUNICATION ERRORS
;
[  (from-node-list to-node-list probability-of-error)
   (from-node-list to-node-list probability-of-error)
   ...
   (from-node-list to-node-list probability-of-error)  ]
;
; Communication links not mentioned have probability-of-error = 0.
;
;                          GOAL COMMUNICATION ERRORS
;
[  (from-node-list to-node-list probability-of-error)
   (from-node-list to-node-list probability-of-error)
   ...
   (from-node-list to-node-list probability-of-error)  ]
;
; Communication links not mentioned have probability-of-error = 0.
;
; ***********************************************************************
;
;                              CONSISTENCY DATA
;
; ***********************************************************************
;
;                                   CBB PT
;
[  (event-class time-location-list true?)
   (event-class time-location-list true?)
   ...
   (event-class time-location-list true?)  ]
;
; true? has a value of t if data is true, nil if false but consistent.
;
;                                   CBB PL
;
[  (event-class time-location-list)
   (event-class time-location-list)
   ...
   (event-class time-location-list)  ]
;
```

```
;                                         CBB VT
;
[  (event-class time-location-list)
   (event-class time-location-list)
   ...
   (event-class time-location-list)  ]
;
;                                         CBB VL
;
[  (event-class time-location-list)
   (event-class time-location-list)
   ...
   (event-class time-location-list)  ]
;
;                                         CBB GT
;
[  (event-class time-location-list)
   (event-class time-location-list)
   ...
   (event-class time-location-list)  ]
;
;                                         CBB GL
;
[  (event-class time-location-list)
   (event-class time-location-list)
   ...
   (event-class time-location-list)  ]
;
;                                         CBB ST
;
[  (event-class time-location-list)
   (event-class time-location-list)
   ...
   (event-class time-location-list)  ]
;
;                                         CBB SL
;
[  (event-class time-location-list)
   (event-class time-location-list)
   ...
   (event-class time-location-list)  ]
;
```

```
; ********************************************************************
;
;                         ENVIRONMENT DATA
;
; ********************************************************************
;
;                          PATTERN DATA
;
[  (event-class ( (time-location-list belief sensor-list)
                  (time-location-list belief sensor-list)
                   ...
                  (time-location-list belief sensor-list)
                ) )
   (event-class((time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list)))
   ...
   (event-class ((time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list)))  ]
;
;                          VEHICLE DATA
;
[  (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )

   (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )
   ...
   (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )  ]
;
```

```
;                                GROUP DATA
;
[  (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )

   (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )
   ...
   (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )  ]
;
;                                SIGNAL DATA
;
[  (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )

   (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )
   ...
   (event-class (time-location-list belief sensor-list)
                (time-location-list belief sensor-list)
                ...
                (time-location-list belief sensor-list) )  ]
;
;
;                   ***** End of Environment File *****
;
                          ##ENVIRONMENT-FILE##
```

APPENDIX B

DISTRIBUTED VEHICLE MONITORING TESTBED ATTRIBUTE DESCRIPTIONS

The major data structures in the distributed vehicle monitoring testbed are: competitor sets, global-hypotheses, goals, hypotheses, interest areas, knowledge sources, knowledge source instantiations, information about each node, and information about each sensor. This appendix lists most of the attributes in each of these data structures with a brief description of each attribute.

B.1 Competitor Set Attributes

Competitor sets are used for calculating the node and network performance measures used in the testbed. A competitor set contains all alternative global-hypotheses (see below) describing possible interpretations for mutually-exclusive aspects of the sensory data. The competitor set structure for all sensor level hypotheses in the testbed is precomputed by the frontend at the beginning of an experimental run using consistency information supplied in the environment file.

ghyps: [link]
An ordered set of global-hypothesis names which comprise this competitor set.

reliabilities:
A vector containing the reliability measures for this competitor set at each node and for the entire network.

B.2 Global-Hypothesis Attributes

Global-hypotheses are also used in calculating the node and network performance measures in the testbed as well as the effect of transmitting a hypothesis to another node. There is one global-hypothesis for each unique hypothesis in the network. Global-hypotheses connect hypotheses at different nodes that have identical level, time-belief-list, and event-class attributes with their appropriate competitor sets.

<u>csets</u>: [link]
An ordered set of competitor set names which contain this global-hypothesis.

<u>event-class</u>:
The event-class number of the global-hypothesis.

<u>node-hyps</u>: [link]
A vector of ordered sets of hypothesis names which have this global-hypothesis as a global-hypothesis attribute or as a hidden-support attribute.

<u>time-location-list</u>:
An ordered set of time-location pairs of this global-hypothesis.

B.3 Goal Attributes

<u>active-time-region-list</u>:
An ordered set of the active time-region pairs of the goal (see Section 4.2.3).

<u>creating-ksis</u>: [link]
An ordered set of the names of all knowledge source instantiations which created (stimulated) this goal.

<u>event-classes</u>:
An ordered set of event-class numbers indicating the desired hypothesis event-classes represented by this goal.

<u>extension-direction</u>:
The direction of extension (forward, backward, neither) for this goal (track extension goals only).

inactive-time-region-list:
An ordered set of the active time-regions of this goal (see Section 4.2.4).

level:
The level (sl, st, gl, gt, vl, vt, pl, pt) indicating the desired hypothesis level represented by the goal.

next-ks:
The index of the next knowledge source at the node which the planner should use to attempt to satisfy this goal.

node:
The node number at which this goal resides.

overlapping-goals: [link]
An ordered set of goal names which spatially overlap the active-time-region-list of this goal during the same time frames.

rating:
The rating of this goal.

received-from:
An ordered set of node numbers which have sent this goal to the node.

result-to:
An ordered set of node numbers which desire reception of hypotheses satisfying this goal.

satisfying-hyps: [link]
An ordered set of hypothesis names which contribute to the satisfaction of this goal.

seen-by:
An ordered set of node numbers which have seen this goal.

stimulated-ksis: [link]
An ordered set of knowledge source instantiation names which have been stimulated by this goal.

stimulus-hyps: [link]
An ordered set of hypothesis names which have stimulated this goal.

subgoals: [link]
An ordered set of goal names which are subgoals of this goal.

supergoals: [link]
An ordered set of goal names of which this goal is a subgoal.

B.4 Hypothesis Attributes

backward-velocity:
The (x,y) component velocity of this hypothesis at its earliest time frame (track hypotheses only).

belief:
The belief value of this hypothesis.

blackboard:
The name of the blackboard on which this hypothesis is located or nil if it is not yet on a blackboard.

consistency-hyp: [link]
The name of the hypothesis which indicates this hypothesis to be consistent.

consistent-hyps: [link]
An ordered set of hypothesis names which are indicated as consistent by this hypothesis (consistency blackboard hypotheses only).

creating-ksis: [link]
An ordered set of the names of all knowledge source instantiations which created this hypothesis or created a hypothesis which was merged with this hypothesis.

event-class:
The event class number of this hypothesis.

forward-velocity:
The (x,y) component velocity of this hypothesis at its latest time frame (track hypotheses only).

ghyp: [link]
The name of the global-hypothesis of this hypothesis (signal location hypotheses only).

hidden-supports: [link]
An ordered set of global-hypothesis names which indicate the hidden supports of this hypothesis (received hypotheses only).

level:
The level (sl, st, gl, gt, vl, vt, pl, pt) on which this hypothesis is located.

local-reflected-reliability:
The [0-10,000] local reflected reliability of this hypothesis (used in computing local knowledge source power measures).

node:
The node number at which this hypothesis is located.

received-from:
An ordered set of node numbers which transmitted this hypothesis to the node.

reflected-reliability:
The [0-10,000] reflected reliability of this hypothesis (used in computing the state of problem solving in the node and network).

satisfied-goals: [link]
An ordered set of goal names which are satisfied (at least in part) by this hypothesis.

seen-by:
An ordered set of node numbers which have seen this hypothesis.

sensor-id:
The sensor identification number of the sensor which produced the hypothesis (signal location hypotheses only).

stimulated-goals: [link]
An ordered set of the goal names which have been stimulated by the creation/modification of this hypothesis.

stimulated-ksis: [link]
An ordered set of the knowledge source instantiation names which have been stimulated by the insertion of this hypothesis.

supported-hyps: [link]
An ordered set of the hypothesis names which are supported by this hypothesis.

supporting-hyps: [link]
An ordered set of the hypothesis names which support this hypothesis.

time-location-list:
An ordered set of the time/location pairs of this hypothesis.

time-beliefs:
An ordered list of the beliefs of this hypothesis for each time frame (used to represent non-uniform belief within a track hypothesis).

true?:
t if this hypothesis is true, nil if it is false.

B.5 Knowledge Source Attributes

consistent-instantiation-counts:
A vector containing the number of instantiations at each node of this knowledge source creating at least one false, but consistent, output hypothesis and no true output hypotheses.

consistent-invocation-counts:
A vector containing the number of invocations at each node of this knowledge source creating at least one false, but consistent, output hypothesis and no true output hypotheses.

false-instantiation-counts:
A vector containing the number of instantiations at each node of this knowledge source creating only false output hypotheses.

false-invocation-counts:
A vector containing the number of invocations at each node of this knowledge source creating only false output hypotheses.

global-powers:
A vector of the total (accumulated) global power measures of this knowledge source at each node.

goodnesses:
A vector of knowledge source efficiency weightings at each node for this knowledge source (used in calculating the ratings of knowledge source instantiations).

instantiations: [link]
A vector of ordered sets of the names of instantiations of this knowledge source at each node.

instantiation-counts:
A vector of the number of instantiations of this knowledge source at each node.

invocations: [link]
A vector of ordered sets of the names of executions of this knowledge source at each node.

invocation-counts:
A vector of the number of invocations of this knowledge source at each node.

local-powers:
A vector of the total (accumulated) local power measures of this knowledge source at each node.

precondition-function:
The name of the precondition function for this knowledge source (or nil, if no precondition function is to be executed for this knowledge source).

resolver-powers:
A vector of the resolver powers to be used for this knowledge source at each node.

true-instantiation-counts:
A vector containing the number of instantiations at each node of this knowledge source creating only true hypotheses.

true-invocation-counts:
A vector containing the number of invocations at each node of this knowledge source creating only true hypotheses.

type:
The type of this knowledge source (communication, extension, external, or synthesis).

B.6 Knowledge Source Instantiation Attributes

created-goals: [link]
An ordered set of goal names which were created by this knowledge source instantiation.

created-hyps: [link]
An ordered set of hypothesis names which were created by this knowledge source instantiation or which were the result of a merger with a hypothesis created by this knowledge source instantiation.

global-power:
The instantaneous global power measure of this knowledge source instantiation.

invocation-time:
The internal clock time when this knowledge source instantiation began execution.

ks: [link]
The name of the knowledge source of which this knowledge source instantiation is an instance.

local-power:
The instantaneous local power measure of this knowledge source instantiation.

next-ksi: [link]
The name of the knowledge source instantiation which follows this knowledge source instantiation on a scheduling queue.

node:
The node number at which this knowledge source instantiation is located.

output-set:
The response frame for this knowledge source instantiation (since knowledge source instantiations are actually executed in the precondition procedure in the testbed, the output-set contains an specification of the hypotheses to be created by this knowledge source instantiation).

rating:
The [-10,000-10,000] rating of this knowledge source instantiation.

stimulus-goals: [link]
An ordered set of goal names which stimulated this knowledge source instantiation.

stimulus-hyps: [link]
An ordered set of hyp names which stimulated this knowledge source instantiation.

B.7 Node Attributes

area-csets: [link]

consistent-goal-counts:
A vector containing the number of false, but consistent, goals created at each level at this node.

consistent-hyp-counts:
A vector containing the number of false, but consistent, hypotheses created at each level at this node.

consistent-ks-instantiation-counts:
A vector containing the number of false, but consistent, knowledge sources instantiated at each level at this node.

consistent-ks-invocation-counts:
A vector containing the number of false, but consistent, knowledge sources invoked at each level at this node.

consistent-received-goal-counts:
A vector containing the number of false, but consistent, goals received at each level at this node.

consistent-received-hyp-counts:
A vector containing the number of false, but consistent, goals received at each level at this node.

consistent-transmitted-goal-counts:
A vector containing the number of false, but consistent, goals transmitted at each level from this node.

consistent-transmitted-hyp-counts:
A vector containing the number of false, but consistent, hypotheses transmitted from each level at this node.

cset-time-region-list:
The time-frame/region pairs defining what competitor sets are used to measure the problem solving state for this node.

current-time:
The current internal clock time at the node.

false-goal-counts:
A vector containing the number of false goals created at each level at this node.

false-hyp-counts:
A vector containing the number of false hypotheses created at each level at this node.

false-ks-instantiation-counts:
A vector containing the number of false knowledge sources instantiated at each level at this node.

false-ks-invocation-counts:
A vector containing the number of false knowledge sources invoked at each level at this node.

false-received-goal-counts:
A vector containing the number of false goals received at each level at this node.

false-received-hyp-counts:
A vector containing the number of false hypotheses received at each level at this node.

false-transmitted-goal-counts:
A vector containing the number of false goals transmitted from each level at this node.

false-transmitted-hyp-counts:
A vector containing the number of false hypotheses transmitted from each level at this node.

goal-directed-goal-counts:
A vector containing the number of subgoals created at each level at this node.

goal-help-levels:
A list of level names for transmitting help goals (used as a filter for goal-help-interest-areas).

goal-help-interest-areas:
The help goal transmission interest areas for this node.

goal-message-buffer:
The message buffer for received goals.

goal-receive-interest-areas:
The goal reception interest areas for this node.

goal-send-levels:
A list of level names for transmitting goals (used as a filter for goal-send-interest-areas).

goal-send-interest-areas:
The goal transmission interest areas for this node.

hyp-message-buffer:
The message buffer for received hypotheses.

hyp-receive-interest-areas:
The hypothesis reception interest areas for this node.

hyp-send-levels:
A list of level names for transmitting hypotheses (used as a filter for goal-help-interest-areas).

hyp-send-interest-areas:
The hypothesis transmission interest areas for this node.

initial-measure:
The initial problem solving state for this node (based solely on the quality of the sensory data).

interest-area-weight-list:

internal-subgoal-data:
Describes to the planner where and how subgoaling is to be performed.

ks-instantiation-counts:
A vector containing the total number of knowledge sources instantiated at each level at this node.

ks-invocation-counts:
A vector containing the total number of knowledge sources invoked at each level at this node.

kss: [link]
The knowledge sources available at this node (in priority order).

location:
The spatial (x, y) location of this node.

last-reception-time:
The time when the most recent hypothesis or goal was received at this node.

last-transmission-cycle:
The time when the most recent hypothesis or goal was transmitted at this node.

measure:
The current problem solving state measure at this node.

merged-goal-counts:
A vector containing the number of merged goals at each level at this node.

merged-hyp-counts:
A vector containing the number of merged hypotheses at each level at this node.

pending-local-ksi: [link]
The next non-communication knowledge source instantiation to be executed at this node.

pending-receive-ksi: [link]
The next reception knowledge source instantiation to be executed at this node.

pending-send-ksi: [link]
The next transmission knowledge source instantiation to be executed at this node.

scheduler-threshold:
The minimum knowledge source instantiation rating required for a knowledge source instantiation to be executed at this node.

sensed-times:
A list of time frames that have been sensed by sensors reporting to this node.

sensor-list:
An ordered set of sensor numbers that report to this node.

true-goal-counts:
A vector containing the number of true goals created at each level at this node.

true-hyp-counts:
A vector containing the number of true hypotheses created at each level at this node.

true-ks-instantiation-counts:
A vector containing the number of true knowledge sources instantiated at each level at this node.

true-ks-invocation-counts:
A vector containing the number of true knowledge sources invoked at each level at this node.

true-received-goal-counts:
A vector containing the number of true goals received at each level at this node.

true-received-hyp-counts:
A vector containing the number of true hypotheses received at each level at this node.

true-transmitted-goal-counts:
A vector containing the number of true goals transmitted from each level at this node.

true-transmitted-hyp-counts:
A vector containing the number of true hypotheses transmitted from each level at this node.

B.8 Sensor Attributes

consistent-signal-count:
The number of false, but consistent, signals generated by this sensor.

false-signal-count:
The number of false signals generated by this sensor.

frequency-mask:
A vector specifying the number of signals (at most one being correct) generated from a simulated signal of each possible frequency event class.

location:
The (x, y) location of this sensor.

location-mask:
An array specifying the number of signals (at most one being correct) generated from a simulated signal located at a particular location relative to the sensor itself.

nodes:
An ordered set of node numbers to which this sensor reports.

probability-true-location:
The probability that a signal generated by the sensor is correctly positioned with respect to the simulated vehicle.

probability-true-frequency:
The probability that a signal generated by the sensor has the correct frequency with respect to the vehicle.

true-signal-count:
The number of true signals generated by this sensor.

weight:
A multiplier applied to the beliefs of signal location hypotheses produced by this sensor.

APPENDIX C

AN ANNOTATED PORTION OF A NETWORK TRACE

This appendix contains a short portion of a trace listing produced by the Distributed Vehicle Monitoring Testbed. Major activities have been annotated. The trace is from the straight vehicle environment with subgoaling experiment described in Chapter V, and the portion reproduced here shows the activities involved in forming and extending vehicle track hypotheses as well as the subgoaling activities performed by the local node planner.

Each line in the trace represents a primitive action in the testbed. (Not all primitive actions are included in this annotated segment.) Each line begins with an asterisk, a plus sign, or a minus sign denoting whether the entity is dealing with true, false but consistent, or false data, respectively (as indicated by the information on the consistency blackboard). The various fields associated with each traced action are indicated below:[1]

1. In the trace "dd" means "data-directed" and "gd" means "goal-directed".

```
* BLACKBOARD EVENT --> event-type level stimulus-hyps/stimulus-goals

* CREATED DD GOAL ---> goal-name level active-time-region-list
                       event-classes {goal-rating}

* CREATED GD GOAL ---> goal-name level active-time-region-list
                       event-classes {goal-rating}

* CREATED HYP -------> hyp-name level time-location-list event-class
                       {belief}

* INSTANTIATED KSI --> ksi-name ks-type stimulus-goals stimulus-hyps
                       <goal-directed-rating data-directed-rating>
                       {overall-ksi-rating}

* INVOKED KSI -------> ksi-name ks-type stimulus-goals stimulus-hyps
                       {ksi-rating}

* MERGED GD GOAL ----> goal-name level active-time-region-list
                       event-classes {goal-rating}

* MERGED KSI --------> ksi-name ks-type stimulus-goals stimulus-hyps

* RERATED GD GOAL ---> goal-name level active-time-location-list
                       event-classes stimulus-hyps stimulated-ksis
                       {old-rating to new-rating}

* RERATED KSI -------> ksi-name ks-type stimulus-goals stimulus-hyps
                       <new-gd-rating-component
                       new-dd-rating-component>
                       {old-rating to new-rating}

* SUPPORTING HYP ----> supporting-hyp-name level time-location-list
                       event-class {belief}
```

```
   -------- Executing Node 1 -- Time Frame 8 -- System Cycle 26 --------

          ; the vehicle track formation knowledge source begins execution
          ; with a vehicle location hypothesis at time frame 2 as its
          ; stimulus hypothesis.
* INVOKED KSI -------> ksi0070 s:vl:vt (g0101 g0105) (h0254) {6602}
          ; a vehicle track in time frames 1 and 2 is created
* CREATED HYP -------> h0258 vt ((1 (6 2)) (2 (8 4))) 1 {9766}
          ; with two supporting hypotheses (one in each time frame).
* SUPPORTING HYP ----> h0246 vl ((1 (6 2))) 1 {9844}
* SUPPORTING HYP ----> h0254 vl ((2 (8 4))) 1 {9844}
          ; a hypothesis creation blackboard event is signalled.
* BLACKBOARD EVENT --> hyp-creation vt (h0258)
          ; a goal specifying extension of the newly created track into
          ; the next time frame is created.
* CREATED DD GOAL ---> g0117 vt ((3 (8 4 12 8))) (1) {4883}
          ; a knowledge source instantiation is scheduled to attempt
          ; to achieve the goal.
* INSTANTIATED KSI --> ksi0072 ef:vt:vt (g0117) (h0258) <6791 9792>
                       {7390}
          ; the extension goal is subgoaled to lower location levels.
          ; at the vehicle location level an identical goal is found,
          ; and so the existing goal is rerated.
* RERATED GD GOAL ---> g0054 vl ((3 (9 5 11 7))) (1) (h0217 h0242)
                       (ksi0047) {875 to 4883}
          ; the remaining subgoals are created.
  CREATED GD GOAL ---> g0118 gl ((3 (9 5 11 7))) (1 2 3 5 6 7) {4883}
  CREATED GD GOAL ---> g0119 sl ((3 (9 5 11 7))) (1 2 3 5 6 7 9 10 11
                       13 14 15 17 18 19) {4883}
          ; returning to the original vehicle track hypothesis creation
          ; event, a goal to synthesize a single vehicle pattern track
          ; hypothesis is created
* CREATED DD GOAL ---> g0120 pt ((1 (2 1 10 6)) (2 (4 1 12 8))) (1)
                       {9766}
          ; and a knowledge source is instantiated to achieve that goal.
* INSTANTIATED KSI --> ksi0073 s:vt:pt (g0120) (h0258) <9766 9766>
                       {9765}
          ; a two-vehicle pattern track goal is also created.
* CREATED DD GOAL ---> g0121 pt ((1 (1 1 7 9)) (2 (1 3 9 11))) (3)
                       {9766}
          ; a pending knowledge source instantiation is found that
          ; can also achieve this goal (the knowledge source instantiation
          ; scheduled to achieve the single vehicle goal).
* MERGED KSI --------> ksi0073 s:vt:pt (g0120 g0121) (h0258)
          ; a second vehicle track hypothesis is formed by the executing
          ; knowledge source instantiation using the stimulus hypothesis
          ; in time frame 2, this time in time frames 2 and 3.
* CREATED HYP -------> h0259 vt ((2 (8 4)) (3 (10 6))) 1 {9766}
* SUPPORTING HYP ----> h0254 vl ((2 (8 4))) 1 {9844}
* SUPPORTING HYP ----> h0248 vl ((3 (10 6))) 1 {9844}
          ; the hypothesis creation event is signalled
```

```
* BLACKBOARD EVENT --> hyp-creation vt (h0259)
       ; and a goal to extend the track backward in time is created.
* CREATED DD GOAL ---> g0124 vt ((1 (4 1 8 4))) (1) {4883}
       ; a knowledge source instantiation is scheduled to achieve it.
* INSTANTIATED KSI --> ksi0074 eb:vt:vt (g0124) (h0259) <6799 9792>
                       {7397}
       ; and the goal is subgoaled.
  CREATED GD GOAL ---> g0125 vl ((1 (5 2 7 3))) (1) {4883}
  CREATED GD GOAL ---> g0126 gl ((1 (5 2 7 3))) (1 2 3 5 6 7) {4883}
  CREATED GD GOAL ---> g0127 sl ((1 (5 2 7 3))) (1 2 3 5 6 7 9 10 11 13
                       14 15 17 18 19) {4883}
       ; a goal to extend the track forward in time is created and
       ; subgoaled.
* CREATED DD GOAL ---> g0128 vt ((4 (10 6 14 10))) (1) {4883}
  CREATED GD GOAL ---> g0129 vl ((4 (11 7 13 9))) (1) {4883}
  CREATED GD GOAL ---> g0130 gl ((4 (11 7 13 9))) (1 2 3 5 6 7) {4883}
       ; subgoaling raises the rating of two pending knowledge source
       ; instantiations (slightly).
* RERATED KSI -------> ksi0022 s:sl:gl (g0019 g0087 g0130) (h0122
                       h0124) <4883 3600> {4595 to 4626}
* RERATED KSI -------> ksi0023 s:sl:gl (g0020 g0021 g0087 g0130) (h0126
                       h0128 h0130) <4883 4880> {4851 to 4882}
  CREATED GD GOAL ---> g0131 sl ((4 (11 7 13 9))) (1 2 3 5 6 7 9 10 11
                       13 14 15 17 18 19) {4883}
       ; a goal to synthesize a single vehicle pattern track is
       ; created.
* CREATED DD GOAL ---> g0132 pt ((2 (4 1 12 8)) (3 (6 2 14 10))) (1)
                       {9766}
* INSTANTIATED KSI --> ksi0075 s:vt:pt (g0132) (h0259) <9766 9766>
                       {9765}
       ; a goal to synthesize a two-vehicle pattern track is
       ; created.
* CREATED DD GOAL ---> g0133 pt ((2 (1 3 9 11)) (3 (3 5 11 13))) (3)
                       {9766}
       ; and the previous knowledge source instantiation is found to
       ; also achieve it.
* MERGED KSI --------> ksi0075 s:vt:pt (g0132 g0133) (h0259)

  -------- Executing Node 1 -- Time Frame 8 -- System Cycle 27 --------

       ; another vehicle track formation knowledge source begins
       ; execution.
* INVOKED KSI -------> ksi0071 s:vl:vt (g0109 g0113) (h0256) {6602}
       ; a vehicle track in time frames 6 and 7 is created
* CREATED HYP -------> h0260 vt ((6 (16 12)) (7 (18 14))) 1 {9766}
* SUPPORTING HYP ----> h0250 vl ((6 (16 12))) 1 {9844}
* SUPPORTING HYP ----> h0256 vl ((7 (18 14))) 1 {9844}
* BLACKBOARD EVENT --> hyp-creation vt (h0260)
       ; a goal specifying extension of the newly created track into
       ; the previous time frame is created
* CREATED DD GOAL ---> g0136 vt ((5 (12 8 16 12))) (1) {4883}
```

```
        ; and subgoaled.
  CREATED GD GOAL ---> g0137 vl ((5 (13 9 15 11))) (1) {4883}
  CREATED GD GOAL ---> g0138 gl ((5 (13 9 15 11))) (1 2 3 5 6 7) {4883}
* RERATED KSI -------> ksi0026 s:sl:gl (g0025 g0091 g0138) (h0132
                       h0133) <4883 3600> {4595 to 4626}
* RERATED KSI -------> ksi0027 s:sl:gl (g0026 g0027 g0091 g0138) (h0136
                       h0138 h0140) <4883 4880> {4851 to 4882}
  CREATED GD GOAL ---> g0139 sl ((5 (13 9 15 11))) (1 2 3 5 6 7 9 10 11
                       13 14 15 17 18 19) {4883}
        ; a goal specifying extension of the newly created track into
        ; the next time frame is created
* CREATED DD GOAL ---> g0140 vt ((8 (18 14 21 18))) (1) {4883}
* INSTANTIATED KSI --> ksi0076 ef:vt:vt (g0140) (h0260) <6791 9792>
                       {7390}
        ; and subgoaled.
  CREATED GD GOAL ---> g0141 vl ((8 (19 15 20 17))) (1) {4883}
  CREATED GD GOAL ---> g0142 gl ((8 (19 15 20 17))) (1 2 3 5 6 7)
                       {4883}
  CREATED GD GOAL ---> g0143 sl ((8 (19 15 20 17))) (1 2 3 5 6 7 9 10
                       11 13 14 15 17 18 19) {4883}
        ; a goal to synthesize a single vehicle pattern track
        ; hypothesis is created.
* CREATED DD GOAL ---> g0144 pt ((6 (12 8 20 16)) (7 (14 10 21 18)))
                       (1) {9766}
* INSTANTIATED KSI --> ksi0077 s:vt:pt (g0144) (h0260) <9766 9766>
                       {9765}
        ; a goal to synthesize a two-vehicle pattern track
        ; hypothesis is created.
* CREATED DD GOAL ---> g0145 pt ((6 (9 11 17 19)) (7 (11 13 19 21)))
                       (3) {9766}
* MERGED KSI --------> ksi0077 s:vt:pt (g0144 g0145) (h0260)
        ; a second vehicle track hypothesis is formed from the time
        ; frame 7 location hypothesis, this time in time frames 7
        ; and 8.
* CREATED HYP -------> h0261 vt ((7 (18 14)) (8 (20 16))) 1 {9766}
* SUPPORTING HYP ----> h0256 vl ((7 (18 14))) 1 {9844}
* SUPPORTING HYP ----> h0252 vl ((8 (20 16))) 1 {9844}
* BLACKBOARD EVENT --> hyp-creation vt (h0261)
        ; a goal to extend the track backward in time is created
* CREATED DD GOAL ---> g0148 vt ((6 (14 10 18 14))) (1) {4883}
* INSTANTIATED KSI --> ksi0078 eb:vt:vt (g0148) (h0261) <6799 9792>
                       {7397}
        ; and subgoaled.
* RERATED GD GOAL ---> g0056 vl ((6 (15 11 17 13))) (1) (h0219 h0243)
                       (ksi0049) {875 to 4883}
  CREATED GD GOAL ---> g0149 gl ((6 (15 11 17 13))) (1 2 3 5 6 7)
                       {4883}
  CREATED GD GOAL ---> g0150 sl ((6 (15 11 17 13))) (1 2 3 5 6 7 9 10
                       11 13 14 15 17 18 19) {4883}
        ; a goal to synthesize a single vehicle pattern track is
        ; created.
```

```
* CREATED DD GOAL ---> g0151 pt ((7 (14 10 21 18)) (8 (16 12 21 20)))
                       (1) {9766}
* INSTANTIATED KSI --> ksi0079 s:vt:pt (g0151) (h0261) <9766 9766>
                       {9765}
       ; a goal to synthesize a two-vehicle pattern track is
       ; created.
* CREATED DD GOAL ---> g0152 pt ((7 (11 13 19 21)) (8 (13 15 21 21)))
                       (3) {9766}
* MERGED KSI --------> ksi0079 s:vt:pt (g0151 g0152) (h0261)

  -------- Executing Node 1 -- Time Frame 8 -- System Cycle 28 --------

       ; the knowledge source instantiated in cycle 26 to synthesize
       ; pattern track hypotheses in time frames 1 and 2 begins
       ; execution.
* INVOKED KSI -------> ksi0073 s:vt:pt (g0120 g0121)
                       (h0258) {9953}
       ; a two-vehicle pattern track hypothesis is created.
- CREATED HYP -------> h0262 pt ((1 (6 2)) (2 (8 4))) 3 {4883}
* SUPPORTING HYP ----> h0258 vt ((1 (6 2)) (2 (8 4))) 1 {9766}
- BLACKBOARD EVENT --> hyp-creation pt (h0262)
       ; a forward extension goal is created and subgoaled.
- CREATED DD GOAL ---> g0155 pt ((3 (8 4 12 8))) (3) {4883}
  CREATED GD GOAL ---> g0156 vt ((3 (6 8 8 10))) (2) {4883}
  CREATED GD GOAL ---> g0157 vt ((3 (12 2 14 4))) (1) {4883}
  CREATED GD GOAL ---> g0158 vl ((3 (6 8 8 10))) (2) {4883}
       ; this knowledge source instantiation's rating was dramatically
       ; increased due to subgoaling (unfortunately it will only
       ; produce false hypotheses).
- RERATED KSI -------> ksi0058 s:gl:vl (g0065 g0158) (h0228) <4883 683>
                       {244 to 4042}
  CREATED GD GOAL ---> g0159 gl ((3 (6 8 8 10))) (9 10 11 13 14 15)
                       {4883}
  CREATED GD GOAL ---> g0160 sl ((3 (6 8 8 10))) (17 18 19 21 22 23 25
                       26 27 29 30 31 33 34 35) {4883}
  CREATED GD GOAL ---> g0161 vl ((3 (12 2 14 4))) (1) {4883}
  CREATED GD GOAL ---> g0162 gl ((3 (12 2 14 4))) (1 2 3 5 6 7) {4883}
  CREATED GD GOAL ---> g0163 sl ((3 (12 2 14 4))) (1 2 3 5 6 7 9 10 11
                       13 14 15 17 18 19) {4883}
       ; the single-vehicle pattern track hypothesis is created.
* CREATED HYP -------> h0263 pt ((1 (6 2)) (2 (8 4))) 1 {9766}
* SUPPORTING HYP ----> h0258 vt ((1 (6 2)) (2 (8 4))) 1 {9766}
* BLACKBOARD EVENT --> hyp-creation pt (h0263)
       ; a forward extension goal is created and subgoaled.
* CREATED DD GOAL ---> g0164 pt ((3 (8 4 12 8))) (1) {9766}
  CREATED GD GOAL ---> g0165 vt ((3 (9 5 11 7))) (1) {9766}
* RERATED KSI -------> ksi0072 ef:vt:vt (g0105 g0117 g0165) (h0258)
                       <9766 9792> {7390 to 9770}
* RERATED GD GOAL ---> g0054 vl ((3 (9 5 11 7))) (1) (h0217 h0242)
                       (ksi0047) {4883 to 9766}
  RERATED GD GOAL ---> g0118 gl ((3 (9 5 11 7))) (1 2 3 5 6 7) nil nil
```

```
                          {4883 to 9766}
  RERATED GD GOAL ---> g0119 sl ((3 (9 5 11 7))) (1 2 3 5 6 7 9 10 11
                          13 14 15 17 18 19) nil nil {4883 to 9766}

  -------- Executing Node 1 -- Time Frame 8 -- System Cycle 29 --------

        ; the knowledge source instantiated in cycle 26 to synthesize
        ; pattern track hypotheses in time frames 1 and 2 begins
        ; execution.
* INVOKED KSI -------> ksi0075 s:vt:pt (g0132 g0133)
                          (h0259) {9953}
        ; a two-vehicle pattern track hypothesis is created.
- CREATED HYP -------> h0264 pt ((2 (8 4)) (3 (10 6))) 3 {4883}
* SUPPORTING HYP ----> h0259 vt ((2 (8 4)) (3 (10 6))) 1 {9766}
- BLACKBOARD EVENT --> hyp-creation pt (h0264)
        ; a backward extension goal is created and subgoaled.
- CREATED DD GOAL ---> g0166 pt ((1 (4 1 8 4))) (3) {4883}
- INSTANTIATED KSI --> ksi0080 mb:pt (g0166) (h0264) <10000 4896>
                          {8979}
  CREATED GD GOAL ---> g0167 vt ((1 (2 5 4 6))) (2) {4883}
  CREATED GD GOAL ---> g0168 vt ((1 (8 1 10 0))) (1) {4883}
  CREATED GD GOAL ---> g0169 vl ((1 (2 5 4 6))) (2) {4883}
  CREATED GD GOAL ---> g0170 gl ((1 (2 5 4 6))) (9 10 11 13 14 15)
                          {4883}
  CREATED GD GOAL ---> g0171 sl ((1 (2 5 4 6))) (17 18 19 21 22 23 25
                          26 27 29 30 31 33 34 35) {4883}
  CREATED GD GOAL ---> g0172 vl ((1 (8 1 10 0))) (1) {4883}
  CREATED GD GOAL ---> g0173 gl ((1 (8 1 10 0))) (1 2 3 5 6 7) {4883}
  CREATED GD GOAL ---> g0174 sl ((1 (8 1 10 0))) (1 2 3 5 6 7 9 10 11
                          13 14 15 17 18 19) {4883}
        ; a forward extension goal is created and subgoaled.
- CREATED DD GOAL ---> g0175 pt ((4 (10 6 14 10))) (3) {4883}
  CREATED GD GOAL ---> g0176 vt ((4 (8 10 10 12))) (2) {4883}
  CREATED GD GOAL ---> g0177 vt ((4 (14 4 16 6))) (1) {4883}
  CREATED GD GOAL ---> g0178 vl ((4 (8 10 10 12))) (2) {4883}
- RERATED KSI -------> ksi0060 s:gl:vl (g0067 g0178) (h0230) <4883 683>
                          {244 to 4042}
  CREATED GD GOAL ---> g0179 gl ((4 (8 10 10 12))) (9 10 11 13 14 15)
                          {4883}
  CREATED GD GOAL ---> g0180 sl ((4 (8 10 10 12))) (17 18 19 21 22 23
                          25 26 27 29 30 31 33 34 35) {4883}
  CREATED GD GOAL ---> g0181 vl ((4 (14 4 16 6))) (1) {4883}
  CREATED GD GOAL ---> g0182 gl ((4 (14 4 16 6))) (1 2 3 5 6 7) {4883}
  CREATED GD GOAL ---> g0183 sl ((4 (14 4 16 6))) (1 2 3 5 6 7 9 10 11
                          13 14 15 17 18 19) {4883}
        ; the single-vehicle pattern track hypothesis is created.
* CREATED HYP -------> h0265 pt ((2 (8 4)) (3 (10 6))) 1 {9766}
* SUPPORTING HYP ----> h0259 vt ((2 (8 4)) (3 (10 6))) 1 {9766}
* BLACKBOARD EVENT --> hyp-creation pt (h0265)
        ; a backward extension goal is created and subgoaled.
* CREATED DD GOAL ---> g0184 pt ((1 (4 1 8 4))) (1) {9766}
* INSTANTIATED KSI --> ksi0081 mb:pt (g0184) (h0265) <10000 9792>
                       {9958}
  CREATED GD GOAL ---> g0185 vt ((1 (5 2 7 3))) (1) {9766}
* RERATED KSI -------> ksi0074 eb:vt:vt (g0101 g0124 g0185) (h0259)
                       <9766 9792> {7397 to 9770}
  RERATED GD GOAL ---> g0125 vl ((1 (5 2 7 3))) (1) nil nil {4883 to
                       9766}
  RERATED GD GOAL ---> g0126 gl ((1 (5 2 7 3))) (1 2 3 5 6 7) nil nil
                       {4883 to 9766}
  RERATED GD GOAL ---> g0127 sl ((1 (5 2 7 3))) (1 2 3 5 6 7 9 10 11 13
                       14 15 17 18 19) nil nil {4883 to 9766}
        ; a forward extension goal is created and subgoaled.
* CREATED DD GOAL ---> g0186 pt ((4 (10 6 14 10))) (1) {9766}
  CREATED GD GOAL ---> g0187 vt ((4 (11 7 13 9))) (1) {9766}
  RERATED GD GOAL ---> g0129 vl ((4 (11 7 13 9))) (1) nil nil {4883 to
                       9766}
  RERATED GD GOAL ---> g0130 gl ((4 (11 7 13 9))) (1 2 3 5 6 7) nil
                       (ksi0022 ksi0023) {4883 to 9766}
        ; here knowledge source instantiations that will produce
        ; correct output hypotheses have their ratings increased
        ; via subgoaling.
* RERATED GD KSI ----> ksi0022 s:sl:gl (g0019 g0087 g0130) (h0122
                       h0124) <9766 3600> {4626 to 8532}
* RERATED GD KSI ----> ksi0023 s:sl:gl (g0020 g0021 g0087 g0130) (h0126
                       h0128 h0130) <9766 4880> {4882 to 8788}
  RERATED GD GOAL ---> g0131 sl ((4 (11 7 13 9))) (1 2 3 5 6 7 9 10 11
                       13 14 15 17 18 19) nil nil {4883 to 9766}

  -------- Executing Node 1 -- Time Frame 8 -- System Cycle 30 --------

        ; the knowledge source instantiated in cycle 27 to synthesize
        ; pattern track hypotheses in time frames 6 and 7 begins
        ; execution.
* INVOKED KSI -------> ksi0077 s:vt:pt (g0144 g0145)
                       (h0260) {9953}
        ; a two-vehicle pattern track hypothesis is created.
- CREATED HYP -------> h0266 pt ((6 (16 12)) (7 (18 14))) 3 {4883}
* SUPPORTING HYP ----> h0260 vt ((6 (16 12)) (7 (18 14))) 1 {9766}
- BLACKBOARD EVENT --> hyp-creation pt (h0266)
        ; a backward extension goal is created and subgoaled.
- CREATED DD GOAL ---> g0188 pt ((5 (12 8 16 12))) (3) {4883}
  CREATED GD GOAL ---> g0189 vt ((5 (10 12 12 14))) (2) {4883}
  CREATED GD GOAL ---> g0190 vt ((5 (16 6 18 8))) (1) {4883}
  CREATED GD GOAL ---> g0191 vl ((5 (10 12 12 14))) (2) {4883}
- RERATED KSI -------> ksi0062 s:gl:vl (g0069 g0191) (h0232) <4883 683>
                       {244 to 4042}
  CREATED GD GOAL ---> g0192 gl ((5 (10 12 12 14))) (9 10 11 13 14 15)
                       {4883}
  CREATED GD GOAL ---> g0193 sl ((5 (10 12 12 14))) (17 18 19 21 22 23
                       25 26 27 29 30 31 33 34 35) {4883}
  CREATED GD GOAL ---> g0194 vl ((5 (16 6 18 8))) (1) {4883}
  CREATED GD GOAL ---> g0195 gl ((5 (16 6 18 8))) (1 2 3 5 6 7) {4883}
  CREATED GD GOAL ---> g0196 sl ((5 (16 6 18 8))) (1 2 3 5 6 7 9 10 11
                       13 14 15 17 18 19) {4883}
        ; a forward extension goal is created and subgoaled.
- CREATED DD GOAL ---> g0197 pt ((8 (18 14 21 18))) (3) {4883}
  CREATED GD GOAL ---> g0198 vt ((8 (16 18 17 20))) (2) {4883}
  CREATED GD GOAL ---> g0199 vt ((8 (22 12 21 14))) (1) {4883}
  CREATED GD GOAL ---> g0200 vl ((8 (16 18 17 20))) (2) {4883}
  CREATED GD GOAL ---> g0201 gl ((8 (16 18 17 20))) (9 10 11 13 14 15)
                       {4883}
  CREATED GD GOAL ---> g0202 sl ((8 (16 18 17 20))) (17 18 19 21 22 23
                       25 26 27 29 30 31 33 34 35) {4883}
  CREATED GD GOAL ---> g0203 vl ((8 (22 12 21 14))) (1) {4883}
  CREATED GD GOAL ---> g0204 gl ((8 (22 12 21 14))) (1 2 3 5 6 7)
                       {4883}
  CREATED GD GOAL ---> g0205 sl ((8 (22 12 21 14))) (1 2 3 5 6 7 9 10
                       11 13 14 15 17 18 19) {4883}
        ; the single-vehicle pattern track hypothesis is created.
* CREATED HYP -------> h0267 pt ((6 (16 12)) (7 (18 14))) 1 {9766}
* SUPPORTING HYP ----> h0260 vt ((6 (16 12)) (7 (18 14))) 1 {9766}
* BLACKBOARD EVENT --> hyp-creation pt (h0267)
        ; a backward extension goal is created and subgoaled.
* CREATED DD GOAL ---> g0206 pt ((5 (12 8 16 12))) (1) {9766}
  CREATED GD GOAL ---> g0207 vt ((5 (13 9 15 11))) (1) {9766}
  RERATED GD GOAL ---> g0137 vl ((5 (13 9 15 11))) (1) nil nil {4883 to
                       9766}
  RERATED GD GOAL ---> g0138 gl ((5 (13 9 15 11))) (1 2 3 5 6 7) nil
                       (ksi0026 ksi0027) {4883 to 9766}
* RERATED GD KSI ----> ksi0026 s:sl:gl (g0025 g0091 g0138) (h0132
                       h0133) <9766 3600> {4626 to 8532}
* RERATED GD KSI ----> ksi0027 s:sl:gl (g0026 g0027 g0091 g0138) (h0136
                       h0138 h0140) <9766 4880> {4882 to 8788}
  RERATED GD GOAL ---> g0139 sl ((5 (13 9 15 11))) (1 2 3 5 6 7 9 10 11
                       13 14 15 17 18 19) nil nil {4883 to 9766}
        ; a forward extension goal is created and subgoaled.
* CREATED DD GOAL ---> g0208 pt ((8 (18 14 21 18))) (1) {9766}
  CREATED GD GOAL ---> g0209 vt ((8 (19 15 20 17))) (1) {9766}
        ; another beneficial knowledge source rating increase due to
        ; subgoaling.
* RERATED KSI -------> ksi0076 ef:vt:vt (g0113 g0140 g0209) (h0260)
                       <9766 9792> {7390 to 9770}
  RERATED GD GOAL ---> g0141 vl ((8 (19 15 20 17))) (1) nil nil {4883
                       to 9766}
  RERATED GD GOAL ---> g0142 gl ((8 (19 15 20 17))) (1 2 3 5 6 7) nil
                       nil {4883 to 9766}
  RERATED GD GOAL ---> g0143 sl ((8 (19 15 20 17))) (1 2 3 5 6 7 9 10
                       11 13 14 15 17 18 19) nil nil {4883 to 9766}
```

SELECTED BIBLIOGRAPHY

Abbreviations.

AAAI-80	Proceedings of the First Annual National Conference on Artificial Intelligence. Available from Garcia-Robinson, Inc., 301 Menlo Oaks, Menlo Park, California, 94025.
AAAI-82	Proceedings of the National Conference on Artificial Intelligence. Available from William Kaufmann, Inc., 95 First Street, Los Altos, California, 94022.
IJCAI-75	Advance Papers of the Fourth International Joint Conference on Artificial Intelligence. Zerographic or microfilm copies available from University Microfilms, 300 North Zeeb Road, Ann Arbor, Michigan, 48106.
IJCAI-77	Proceedings of the Fifth International Joint Conference on Artificial Intelligence. Available from Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15213.
IJCAI-79	Proceedings of the Sixth International Joint Conference on Artificial Intelligence. Available from Computer Science Department, Stanford University, Stanford, California, 94305.
IJCAI-81	Proceedings of the Seventh International Joint Conference on Artificial Intelligence. Available from American Association for Artificial Intelligence, 445 Burgess Drive, Menlo Park, CA, 94025.

References.

ALLE79 James F. Allen.
A Plan-Based Approach to Speech Act Recognition.
PhD thesis, University of Toronto, 1979.
Available as Technical Report 131/79, Department of Computer Science, University of Toronto, Toronto, Canada.

ANTH65 Robert N. Anthony.
Planning and Control Systems: A framework for analysis.
Division of Research, Harvard Business School, Harvard University, 1965.

BARN81 Jeffrey A. Barnett.
Computational methods for a mathematical theory of evidence.
In IJCAI-81, pages 868-875.

BARN82 Jeffrey A. Barnett and Lee D. Erman.
Making control decisions in an expert system is a problem-solving task.
Unpublished working paper, USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California, 90291, April 1982.

BATE81 Peter C. Bates, Jack C. Wileden, and Victor R. Lesser.
A language to support debugging in distributed systems.
Technical Report 81-7, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, April 1981.

BAUD78 Gerard M. Baudet.
Asynchronous iterative methods for multiprocessors.
Journal of the ACM 25(2):226-244, April 1978.

BEER78 Stafford Beer.
Platform for Change.
Wiley, corrected reprint 1978.

BEER79 Stafford Beer.
The Heart of Enterprise.
Wiley, 1979.

BEER81 Stafford Beer.
Brain of the Firm.
Wiley, second edition 1981.

BROO79 Richard S. Brooks and Victor R. Lesser.
Distributed problem solving using iterative refinement.
Technical Report 79-14, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, May 1979.

BROO83 Richard S. Brooks.
Experiments in Distributed Problem Solving with Iterative Refinement.
PhD Thesis, University of Massachusetts, 1983.
Available as Technical Report 82-25, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, October 1982.

CHAP75 Alphonse Chapanis
Interactive human communication.
Scientific American 232(3):36-42, March 1975.

CHU77 W. W. Chu.
Advances in Computer Communications.
Artech House, Dedham, Massachusetts, 1977.

COHE78 Philip R. Cohen.
On Knowing What to Say: Planning Speech Acts.
PhD thesis, University of Toronto, 1978.
Available as Technical Report 118, Department of Computer Science, University of Toronto, Toronto, Canada, January 1978.

CRAN78 Hewitt D. Crane.
Beyond the seventh synapse: The neural marketplace of the mind.
Research memorandum, SRI International, Menlo Park, California, December 1978.

CRAN80 Hewitt D. Crane. The New Social Marketplace: Notes on effecting social change in America's third century.
Ablex Publishing, 1980.

DAVI80 Randall Davis.
Meta-rules: Reasoning about control.
Artificial Intelligence 15(3):179-222, December 1980.

DAVI81 Randall Davis and Reid G. Smith.
Negotiation as a metaphor for distributed problem solving.
AI Memo 624, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1981.

ENGE77 Robert S. Engelmore and H. Penny Nii.
A knowledge based system for the interpretation of protein x-ray crystallographic data.
Technical Report STAN-CS-77-589, Computer Science Department, Stanford University, Stanford, California, February 1977.

ERMA75 Lee D. Erman and Victor R. Lesser.
A multi-level organization for problem solving using many diverse cooperating sources of knowledge.
In IJCAI-75, pages 483-490.

ERMA80 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.

ERMA81 Lee D. Erman, Philip E. London, and Stephen F. Fickas.
The design and an example use of Hearsay-III.
In IJCAI-81, pages 409-415.

FELD77 Jerome A. Feldman and Robert F. Sproull.
Decision theory and artificial intelligence II: The hungry monkey.
Cognitive Science 1(2):158-192, April 1977.

FELD79 Jerome A. Feldman.
High level programming for distributed computing.
Communications of the ACM 22(6):353-368, June 1979.

FENN77 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.

FOX79 Mark S. Fox.
Organization structuring: Designing large complex software.
Technical Report CMU-CS-79-155, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, December 1979.

FOX81 Mark S. Fox.
An organizational view of distributed systems.
IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1): 70-80, January 1981.

GALB73 Jay Galbraith.
Designing Complex Organizations.
Addison-Wesley, 1973.

GALB77 Jay R. Galbraith.
Organization Design.
Addison-Wesley, 1977.

GARV81 Thomas D. Garvey, John D. Lowrance, and Martin A. Fischler.
An inference technique for integrating knowledge from disparate sources.
In IJCAI-81, pages 319-325.

GREE82 Peter E. Green
Distributed acoustic surveillance and tracking.
In Proceedings of the Distributed Sensor Networks Workshop, pages 117-141, January 1982.
Copies may be available from MIT Lincoln Laboratory, Lexington, Massachusetts, 02173.

HAGA82 Roger Hagafors.
The character of organizational problems: A classification system for organizational decision-making.
Technical Report 317, Department of Psychology, University of Uppsala, Uppsala, Sweden, 1982.

HANS78 Allen R. Hanson and Edward M. Riseman.
VISIONS: A computer system for interpreting scenes.
In Allen R. Hanson and Edward M. Riseman, editors, Computer Vision Systems, pages 303-333, Academic Press, 1978.

HAYE77 Frederick Hayes-Roth and Victor R. Lesser.
Focus of attention in the Hearsay-II speech understanding system.
In IJCAI-77, pages 27-35.

HAYE79 Barbara Hayes-Roth and Frederick Hayes-Roth.
A cognitive model of planning.
Cognitive Science 3(4):275-310, October-December 1979.

HEWI77 Carl Hewitt.
Viewing control structures as patterns of passing messages.
Artificial Intelligence 8(3):323-364, Fall 1977.

KAHN78 R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman.
Advances in packet radio technology.
Proceedings of the IEEE 66(11):1468-1496, November 1978.

KAST74 F. E. Kast and J. E. Rosenzweig.
Organization and Management.
McGraw-Hill, second edition 1974, pages 574-575.

KILM69 W. L. Kilmer, W. S. McCulloch, and J. Blum.
A model of the vertebrate central command system.
International Journal of Man-Machine Studies 1(3):279-309, July 1969.

KIMB75 S. R. Kimbleton and G. M. Schneider.
Computer communication networks: Approaches, objectives, and performance considerations.
Computer Surveys 7(3):129-173, September 1975.

KOHL81 Walter H. Kohler
A survey of techniques for synchronization and recovery in decentralized computer systems.
Computer Surveys 13(2):149-183, June 1981.

KORN79 William A. Kornfeld.
EITHER: A parallel problem solving system.
In IJCAI-79, pages 490-492.

LACO78 R. Lacoss and R. Walton.
Strawman design of a DSN to detect and track low flying aircraft.
In Proceedings of the Distributed Sensor Nets Workshop, pages 41-52, December 1978.
Copies may be available from the Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15213.

LAWR67 P. R. Lawrence and J. W. Lorsch.
Organization and Environment: Managing differentiation and integration.
Division of Research, Harvard Business School, Harvard University, 1967.

LENA75 Douglas B. Lenat.
Beings: Knowledge as interacting experts.
In IJCAI-75, pages 126-133.

LESS77 Victor R. Lesser and Lee D. Erman.
A retrospective view of the Hearsay-II architecture.
In IJCAI-77, pages 790-800.

LESS80a Victor R. Lesser, Jasmina Pavlin, and Scott Reed.
Quantifying and simulating the behavior of knowledge-based interpretation systems.
In AAAI-80, pages 111-115, August 1980.

LESS80b Victor R. Lesser and Lee D. Erman.
An experiment in distributed interpretation.
IEEE Transactions on Computers C-29(12):1144-1163, December 1980.

LESS81 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.

LOWR82 John Douglas Lowrance.
Dependency-Graph Models of Evidential Support.
PhD Thesis, University of Massachusetts, 1982.
Available as Technical Report 82-26, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, September 1982.

MANN79 William C. Mann.
Design for dialogue comprehension.
In Proceedings of the Seventeenth Annual Meeting of the Association for Computational Linguistics, August 1979.

MARC58 James G. March and Herbert A. Simon.
Organizations.
Wiley, 1958.

McCU65 Warren S. McCulloch.
What's in the brain that ink may character?
In Warren S. McCulloch, Embodiments of Mind, pages 387-397, MIT Press, 1965.

NADL77 David A. Nadler and Edward E. Lawler, III.
Motivation: A diagnostic approach.
In Richard Hackman, Edward E. Lawler, III, and Lyman W. Porter, editors, Perspectives on Behavior in Organizations, pages 26-34, McGraw-Hill, 1977.

NII78 H. Penny Nii and Edward A. Feigenbaum.
Rule based understanding of signals.
In D. A. Waterman and Frederick Hayes-Roth, editors, Pattern-Directed Inference Systems, pages 483-501, Academic Press, 1978.

NII82 H. Penny Nii, Edward A. Feigenbaum, John J. Anton, and A. J. Rockmore.
Signal-to-symbol transformation: HASP/SIAP case study.
AI Magazine 3(2):23-35, Spring 1982.

NILS79 N. J. Nilsson.
A production system for automatic deduction.
In J. E. Hayes, Donald Michie, and L. I. Mikulich, editors, Machine Intelligence 9, pages 101-126, Halsted Press, 1979.

NILS80a Nils J. Nilsson.
Principles of Artificial Intelligence.
Tioga, Palo Alto, California, 1980.

NILS80b Nils J. Nilsson.
Two heads are better than one.
SIGART Newsletter (73):43, October 1980.

REED80 S. Reed and V. R. Lesser.
Division of labor in honey bees and distributed focus of attention.
Technical Report 80-17, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, September 1981.

RUME76 D. E. Rumelhart.
Toward an interactive model of reading.
Technical Report 56, Center for Human Information Processing, University of California, San Diego, California, 1976.

SACE78 Earl D. Sacerdoti.
What language understanding research suggests about distributed artificial intelligence.
In Proceedings of the Distributed Sensor Nets Workshop, pages 8-11, December 1978.
Copies may be available from the Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15213.

SCIE77 Scientific American.
Special issue on microelectronics.
Scientific American 237(3):62-245, September 1977.

SEAR70 John R. Searle.
Speech Acts: An Essay in the Philosophy of Language.
Cambridge University Press, 1970.

SIMO57 Herbert A. Simon.
Models of Man.
Wiley, 1957.

SIMO69 Herbert A. Simon.
The Sciences of the Artificial.
MIT Press, 1969.

SMIT78 Reid Garfield Smith.
A Framework for Problem Solving in a Distributed Processing Environment.
PhD thesis, Stanford University, 1978.
Available as Technical Report STAN-CS-78-800, Computer Science Department, Stanford University, Stanford, California, December 1978.

SMIT80 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.

SMIT81 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.

SOLO77 Elliot M. Soloway and Edward M. Riseman.
Levels of pattern description in learning.
In IJCAI-77, pages 801-811.

SPRO77 Robert F. Sproull.
Strategy Construction using a Synthesis of Heuristic and Decision-Theoretic Methods.
PhD thesis, Stanford University, 1977.
Available as Technical Report CSL-77-2, Xerox Palo Alto Research Center, Palo Alto, California, July 1977.

STEF80 Mark Jeffrey Stefik.
Planning with Constraints.
PhD thesis, Stanford University, 1980.
Available as Technical Report STAN-CS-80-784, Computer Science Department, Stanford University, Stanford, California, January 1980.

TENN79 Robert R. Tenny.
Distributed Decision Making using a Distributed Model.
PhD thesis, Massachusetts Institute of Technology, 1979.
Available as Technical Report LIDS-TD-938, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1979.

ZISM78 Michael D. Zisman.
Use of production systems for modeling asynchronous, concurrent processes.
In D. A. Waterman and Frederick Hayes-Roth, editors, Pattern-Directed Inference Systems, pages 53-68, Academic Press, 1978.

CITATION INDEX