85-15

Coherent Cooperation Among Communicating Problem Solvers

Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

September 30, 1985

This research was sponsored, in part, by the National Science Foundation under Grant MCS-8306327 and by the Defense Advanced Research Projects Agency (DOD), monitored by the Office of Naval Research under Contract NR049-041.

Abstract

When two or more computing agents work on interacting tasks, their activities should be coordinated so that they cooperate coherently. Coherence is particularly problematic in domains where each agent has only a limited view of the overall task, where communication between agents is limited, and where there is no "controller" to coordinate the agents. Our approach to coherent cooperation in such troublesome domains has been developed and implemented in a distributed problem solving network. This approach stresses the importance of sophisticated local control by which each problem solving node integrates knowledge of the problem domain with (meta-level) knowledge about network coordination. This allows nodes to make rapid, intelligent local decisions based on changing problem characteristics with only a limited amount of conferring with each other to coordinate these decisions.

In this article, we describe three mechanisms that improve local control decisions and enable nodes to cooperate coherently. These mechanisms are an organizational structure which provides a long-term framework for network coordination to guide each node's local control decisions, a planner at each node which develops sequences of problem solving activities based on the current situation, and meta-level communication about the current state of local problem solving which enables nodes to dynamically make short-term refinements to the long-term organization. We provide empirical results showing the benefits and limitations of these mechanisms in a variety of problem solving situations. Moreover, these mechanisms are not without cost, and we provide performance results showing the mechanisms to be particularly cost-effective in complex problem solving situations. Finally, we describe how these mechanisms might be of more general use in other distributed computing applications.

1. Introduction

Cooperation in a distributed computing environment requires two types of control decisions. One type is *network control*: tasks and responsibilities must be assigned to each of the computing agents. Typically, network control attempts to equally distribute the computing load among agents to maximize parallel computation. The other type is *local control*: each computing agent must choose a task to execute next from among its assigned tasks.

Much research in distributed computing systems has concentrated on network control algorithms that exchange tasks among agents based on protocols such as bidding [22,25]. These systems tend to have unsophisticated local control; a simple scheduling algorithm (for example, based on task reception times or on task deadlines) is used. Since tasks are usually assumed to be independent, the local control component of one agent can be unaware of the local control decisions being made elsewhere.

As distributed computation is used in more diverse applications, however, the task independence assumption becomes invalid. Examples of how interactions among tasks affect local control decisions include:

- If tasks have precedence constraints, the local control decisions of agents with succeeding tasks depend on the decisions of the agents with the preceding tasks.
- To improve reliability, equivalent tasks may be assigned to several agents. If one agent successfully executes a task, the other agents should avoid redundantly executing equivalent tasks.
- In a distributed programming environment, each agent may have pieces of several distributed programs. Because the pieces of a distributed program may need to synchronize or exchange information, the local control components of agents must consider how the timing of their local decisions will affect activity on other agents.

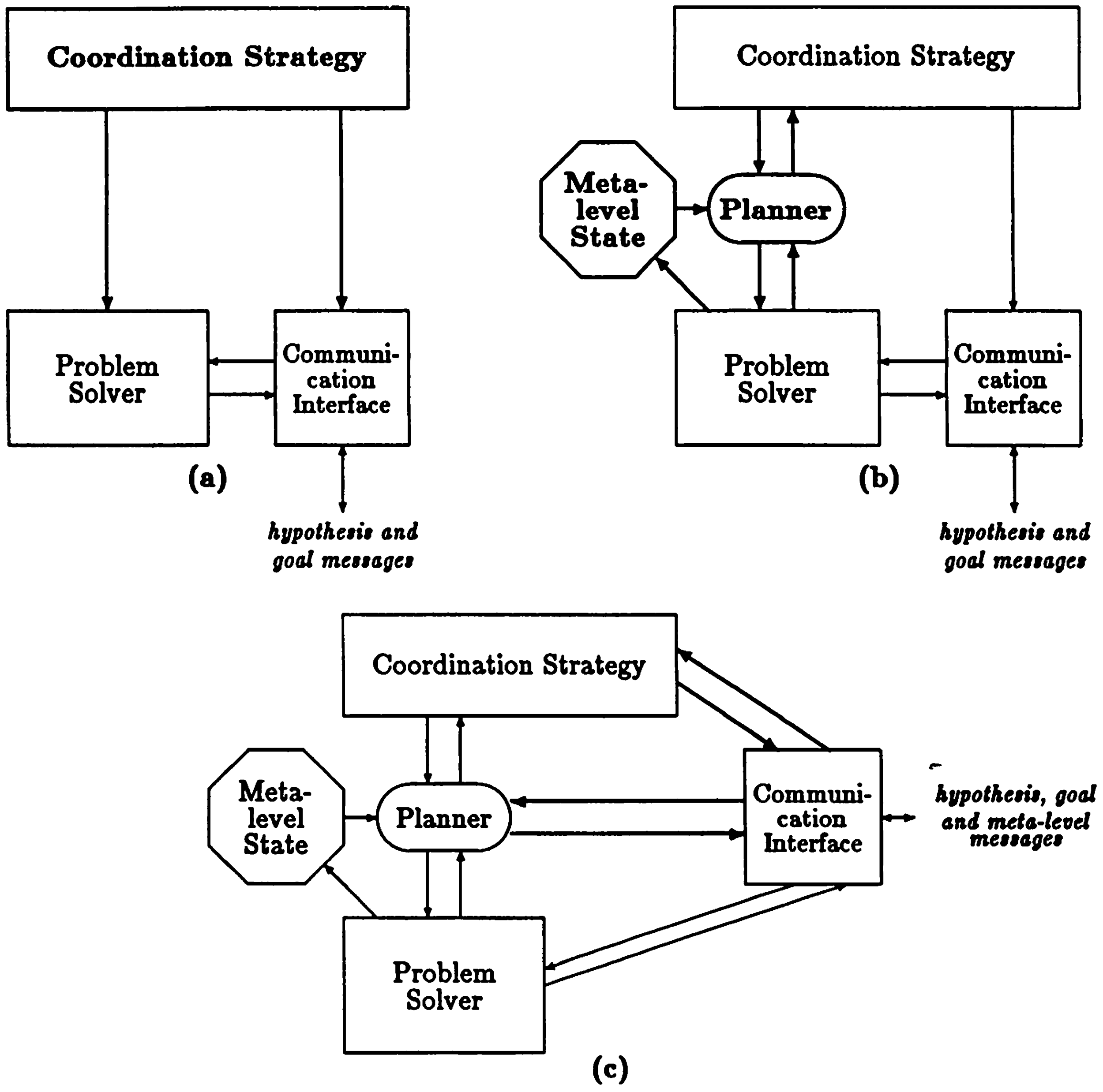
In this article, we enable agents with interacting tasks to cooperate effectively by giving each agent knowledge about the actions and intentions of the other agents. Achieving such *network awareness* is difficult in a distributed environment because of communication delays and bandwidth limitations. We explore mechanisms for improving network awareness without excessive computation and communication overhead. Since full network awareness is seldom possible in large networks, we develop mechanisms that allow an agent to make local control decisions despite potentially incomplete, inaccurate, and inconsistent knowledge about other agents. We have implemented and experimentally tested these mechanisms in a simulated distributed problem solving network.

In a *distributed problem solving network*, each agent is a semi-autonomous problem solving *node* that can communicate with other nodes. Nodes work together to solve a single problem by individually solving interacting subproblems and integrating their subproblem solutions into an overall solution. These networks are typically used in applications such as distributed sensor networks [17,25], distributed air traffic control [1], and distributed robot systems [11], where there is a natural spatial distribution of information but where each node has insufficient local information to completely and accurately solve its subproblems. To improve their local information, nodes must share subproblem solutions; cooperation thus requires intelligent local control decisions so that each node performs tasks which generate useful subproblem solutions. The use of a global "controller" to make these decisions for the nodes is not an option because it would be a severe communication and computational bottleneck and would make the network susceptible to complete collapse if it fails. Because nodes must make these decisions based only on their local information, well-coordinated or *coherent* cooperation is difficult to achieve [7,18].

In the *functionally-accurate, cooperative* (FA/C) approach to distributed problem solving [18], nodes cooperate by generating and exchanging tentative, partial solutions based on their local views. By iteratively exchanging their potentially incomplete, inaccurate, and inconsistent partial solutions, the nodes eventually converge on an overall network solution. Coherent cooperation thus requires that, at any given time, each node performs a sequence of tasks to generate a tentative partial solution that is compatible with the solutions being generated at other nodes. The nodes often have overlapping views of the problem and alternative methods for forming partial solutions to insure both that problem solving can occur despite significant amounts of error in local data and that network performance will degrade gracefully if a subset of the nodes fail. Coherent cooperation among these *overlapping nodes* is particularly troublesome, requiring the nodes to work together to cover the overlapping area without duplicating each other's work.

A distributed problem solving network therefore involves issues in precedence among tasks (smaller partial solutions must precede larger, subsuming partial solutions), in redundancy among tasks (overlapping nodes may redundantly generate identical partial solutions), and in timing of tasks (a timely generation and exchange of certain partial solutions may guide nodes into behaving more effectively). Sophisticated local control at each node is thus paramount if nodes are to cooperate coherently. This article describes how such local control can be implemented; examines its costs, benefits, and limitations; and suggests how it might be introduced in other distributed computing systems.

In the next section, we present the conceptual issues in coherent cooperation among problem solvers, and in Section 3 we describe our experimental testbed. Section 4 introduces the first of three mechanisms to increase coherence through improved local control—a coordination strategy that provides each node with a general view of network responsibilities (including its own) to guide its problem solving and communication decisions (Figure 1a). In Section 5, the second mechanism, a local planner, is described. The planner increases the node's awareness of its local state and problem solving activities, and therefore allows it to make small, dynamic refinements to its copy of the coordination strategy (Figure 1b). Section 6 presents mechanisms that enable nodes to make more informed communication decisions and that allow nodes to exchange *meta-level* information—information specifically intended to increase coherence (Figure 1c). In Section 7, we examine experimental results from complex environments, citing the costs as well as the benefits of our mechanisms. Section 8 is devoted to a discussion of how our approach might be incorporated in other distributed computing systems. Finally, Section 9 summarizes our current ideas about coherent cooperation and describes our plans for future research.



The three new mechanisms we have developed are illustrated. Initially, a node has a problem solver and a communication interface for exchanging hypotheses (partial problem solutions) and goals (intentions to form partial problem solutions). In (a), a coordination strategy is added to guide problem solving and communication decisions. In (b), a local planner and a meta-level state representation are added to increase a node's problem solving and coordination awareness. Finally, in (c), meta-level messages (high-level views of node activities) can be exchanged through the communication interface to increase network awareness.

Figure 1: Evolutionary Phases of Sophisticated Local Control.

2. Problem Solving and Coherent Cooperation

A distributed problem solving network requires individual agents that are effective at problem solving and are able to cooperate as a coherent team. In this section, we present our ideas on problem solving and coherent cooperation. With this framework, we can begin to recognize the issues that must be addressed to achieve coherent cooperation in a distributed problem solving network.

2.1 Effective Problem Solving

Artificial intelligence researchers typically view problem solving as a search of the space of possible solutions to find the best one [27]. For example, given the problem of finding the shortest route between two cities, the solution space would consist of all possible routes, and problem solving would involve searching for the shortest of these. Solution spaces for non-trivial problems are extremely large and can neither be completely enumerated nor searched exhaustively. One way to make the search tractable is to transform the space of possible solutions into a space of partial solutions. A *partial solution* represents all complete solutions which contain it. Through the judicious selection of problem solving tasks, an effective problem solver controls the incremental construction of alternative partial solutions so that that one or more satisfactory complete solutions are found within a reasonable amount of time.

How search in this space of partial solutions should be controlled depends on what constitutes a satisfactory solution and how long a reasonable time is. If only the best solution will do, the control must be conservative so that it does not overlook any potentially optimal portions of the space. If the control must find a solution in a short amount of time, it may limit the search to a small part of the space without the guarantee that the best (or any) solution is in that part. When time is limited, therefore, a problem solver must have a sophisticated control structure to decide what part of the space should be searched. The decisions must often be based on potentially inaccurate or incomplete problem knowledge. As a result, the control decisions are uncertain because their full ramifications cannot be predicted. This *control uncertainty* can be reduced only by providing the problem solver with more accurate and complete problem information.

The control structure becomes significantly more complex in a distributed environment because a problem solver must then consider how its local searching actions complement those of the other nodes. More effective problem solving can occur if each node explores a different part of the space concurrently—within the same amount of time, the network more thoroughly searches the space, and so, can find more satisfactory solutions. Furthermore, a partial solution received from another node can provide additional information with which a node may generate partial solutions that it previously had been incapable of developing, or at least could narrow the part of the space that the node might search to find a compatible partial solution to combine with it. This subtask interdependence means that nodes must work together to generate an overall solution; effective distributed problem solving cannot be achieved without cooperation.

2.2 Coherent Cooperation

Achieving cooperation in a distributed problem solving network is a difficult problem. Although the individual nodes are predisposed to work together toward network goals, they may compete or conflict with each other because each must locally interpret the network goals. Since the nodes may have different local views, incompatible local interpretations of network goals may be created. Even if nodes have compatible goals, network performance depends on how coherently the nodes work together as a team.

Global coherence means that the activities of the nodes should make sense given overall network goals. Nodes should avoid unnecessarily duplicating the work of others, sitting idle while others are swamped with work, or transmitting information that will not improve overall network performance. Because network coordination must be decentralized to improve reliability and responsiveness, the amount of global coherence in the network depends on the degree to which each node makes coherent local decisions based on its local view of network problem solving. In particular, local control decisions about what part of the partial solution space to search and how to search it (breadth-first, depth-first, etc.) must make sense when compared to the search occurring at other nodes.

At any given time, a node may be capable of searching several parts of the solution space. Since search at one node can form partial solutions that influence search at another node, a node ranks its pending, interacting search tasks based on how it believes each will improve *network* problem solving. A node's decision to execute its top ranked task is thus more or less coherent depending on how highly ranked the task would have been if the node had complete network awareness. We can guarantee full global coherence only if each node has a complete and accurate view of the problem solving activities and intentions of all other nodes. One way to achieve this would be to globally predefine a schedule of activities for each node at network creation, but such a network would be inflexible to changing problem solving situations and network characteristics. Another way to achieve a global view would require nodes to broadcast all changes to their states, an infeasible scheme because of bandwidth limitations and communication delays. Therefore, we have no practical means to insure full global coherence. The FA/C approach develops a framework which tolerates incorrect control decisions, so that network goals can be achieved with only partial global coherence. However, less than full coherence can waste resources and therefore can degrade performance.

Our research in distributed problem solving is focused on the effective use of all computation and communication resources. By increasing node sophistication, we improve communication resource usage since nodes make more intelligent communication decisions. We also improve computation resource usage: although better local control decisions add computation overhead, they may focus search so that better partial solutions can be found with less search. Our approach has thus been directed toward increasing the coherence of local control decisions by using sophisticated local processing and selective communication to reduce control uncertainty. We have implemented and tested our approach in a (simulated) distributed problem solving network.

3. The Distributed Vehicle Monitoring Testbed

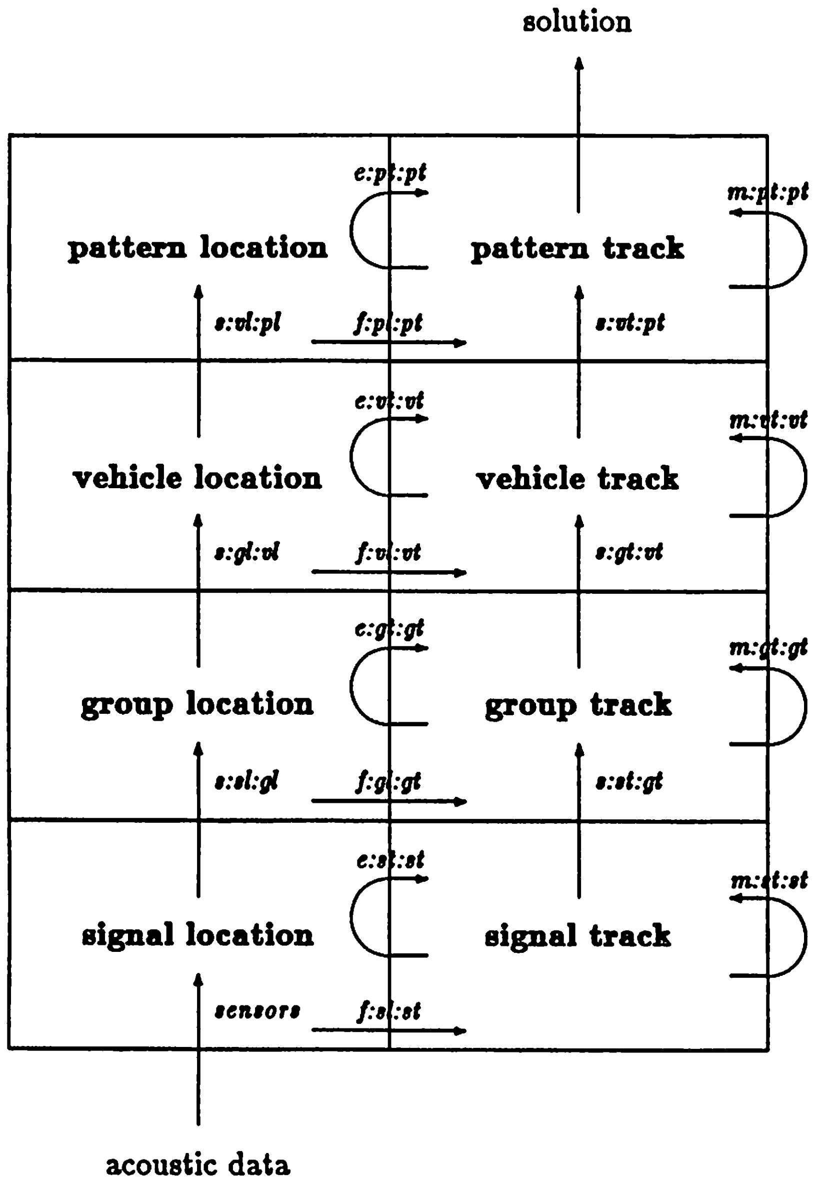
By simulating a network of problem solving nodes, the Distributed Vehicle Monitoring Testbed (DVMT) provides a framework where general approaches for distributed problem solving can be developed and evaluated [19,20]. Each simulated node applies simplified signal processing knowledge to acoustically sensed data in an attempt to identify, locate, and track patterns of vehicles moving through a two-dimensional space. By varying parameters in the DVMT that specify the accuracy and range of the acoustic sensors, the acoustic signals that are to be grouped together to form patterns of vehicles, the power and distribution of knowledge among the nodes in the network, and the node and communication topology, a wide variety of cooperative distributed problem solving situations can be modeled. Furthermore, using simplified signal processing knowledge reduces the processing complexity and knowledge engineering effort required in the DVMT without significantly changing the fundamental network coordination characteristics of the distributed vehicle monitoring task.

3.1 The DVMT Nodes

Each problem solving node has a Hearsay-II, blackboard-based architecture [10], with knowledge sources and levels of abstraction appropriate for vehicle monitoring. A *knowledge source* (KS) performs the basic problem solving tasks of extending and refining *hypotheses* (partial solutions). A hypothesis is characterized by one or more *time-locations* (where the vehicle was at certain times), by an *event-class* (classifying the frequency or vehicle type), and by a *belief* (the confidence in the accuracy of the hypothesis).

The hypotheses are organized on a blackboard with four levels of abstraction: **signal** (for low-level analyses of the sensory data), **group** (for collections of harmonically related signals), **vehicle** (for collections of groups that correspond to given vehicle types), and **pattern** (for collections of spatially related vehicle types such as vehicles moving in a formation). Each of these levels is split into a level for location hypotheses (which have one time-location) and a level for track hypotheses (which have a sequence of time-locations). In total, nodes have eight blackboard levels with appropriate KSs for combining hypotheses on one level to generate more encompassing hypotheses on the same or on a higher level (Figure 2).

A *knowledge source instantiation* (KSI) represents the potential application of a particular KS to specific hypotheses. Each node maintains a queue of pending KSIs and, at any given time, must rank the KSIs to decide which one to invoke next. To improve these decisions, we have extended the Hearsay-II architecture (Figure 3) so that nodes can reason more fully about the intentions or *goals* of the KSIs [3,4]. As explicit representations of the node's intentions to abstract and extend hypotheses, goals are stored on a separate goal blackboard and are given importance ratings. A goal processing component recognizes interactions between goals and adjusts their ratings appropriately (for example, subgoals of an important goal might have their ratings boosted). The scheduler ranks a KSI based both on the estimated beliefs of the hypotheses it may produce and on the ratings of the



KSs combine hypotheses to form more encompassing hypotheses on the same or higher levels. Synthesis (s:) KSs generate higher level hypotheses out of compatible lower level hypotheses. Formation (f:) KSs form a track hypothesis from two combinable location hypotheses. Extension (e:) KSs combine a track hypothesis with a compatible location hypothesis to extend the track. Merge (m:) KSs combine two shorter track hypotheses that are compatible into a single longer track. The sensors KS creates signal location hypotheses out of sensed data. Communication KSs are not shown.

Figure 2: Levels of Abstraction and Knowledge Sources.

goals it is expected to satisfy. Appropriate goal processing can therefore alter KSI rankings to improve local control decisions.

A goal's rating may also be modified based on its source. Nodes have communication KSs and (simulated) communication hardware so that they can exchange hypotheses and goals (Figure 3). Received hypotheses and goals trigger the same activities that locally generated hypotheses and goals do, so that a node is responsive both to internally generated information and to externally-directed suggestions or requests for work received from other nodes. However, a node may modify the ratings of received goals or goals stimulated by received hypotheses: if the ratings of these goals are increased, the node is said to be ***externally-biased*** and is essentially subservient to others; if the ratings of these goals are decreased, the node is ***locally-biased*** and prefers its own activities over the suggestions of others; and if the ratings of these goals are not modified at all, the node is ***unbiased***.

3.2 DVMT Problem Solving Activity

A node that makes better local control decisions searches the solution space more effectively by invoking fewer inappropriate KSIs as it works towards generating important hypotheses. To evaluate an approach for improving local control decisions, therefore, we use the DVMT to simulate problem solving environments and measure the number of local control decisions required to generate the solution. We explore the strengths and limitations of an approach by developing a set of environments that cover a range of problem solving situations. For example, we might make problem solving more difficult by altering the distribution of sensed data among the nodes or by giving a node "noisy" data along with the actual vehicle data. These modifications can complicate problem solving so that issues in cooperation among nodes become more pronounced.

Given a particular problem solving environment, the DVMT simulates the nodes' activities as follows.[1] Each node begins by transforming its sensed data into a set of signal location hypotheses (using the sensors KS).[2] With each new hypothesis, the node generates goals to improve upon it and forms KSIs to achieve these goals. After it has created all of its signal hypotheses, a node then chooses a KSI to invoke. The KSI invocation may cause the creation of new hypotheses, which stimulate the generation of more goals, which in turn may cause more KSIs to be formed. The node then chooses another KSI and the cycle repeats. In our environments, each cycle requires one time unit.

By limiting the KSs available to nodes in our environments, we allow them to form tracks from locations only at the vehicle level. Nodes thus perform both low-level problem solving activities (creating group location and vehicle location hypotheses), and high-level activities (forming, extending, and merging vehicle track and pattern track hypotheses). Initially, the node will drive up its most promising signal data to form tracks. Through goal processing, the goals to extend these tracks boost the importance of low-level processing

[1]More detailed descriptions can be found elsewhere [5,20].

[2]Although we have been exploring scenarios where sensor data is incorporated over time, the less complicated environments where all sensor data is incorporated at once illustrate the issues we address in this article.

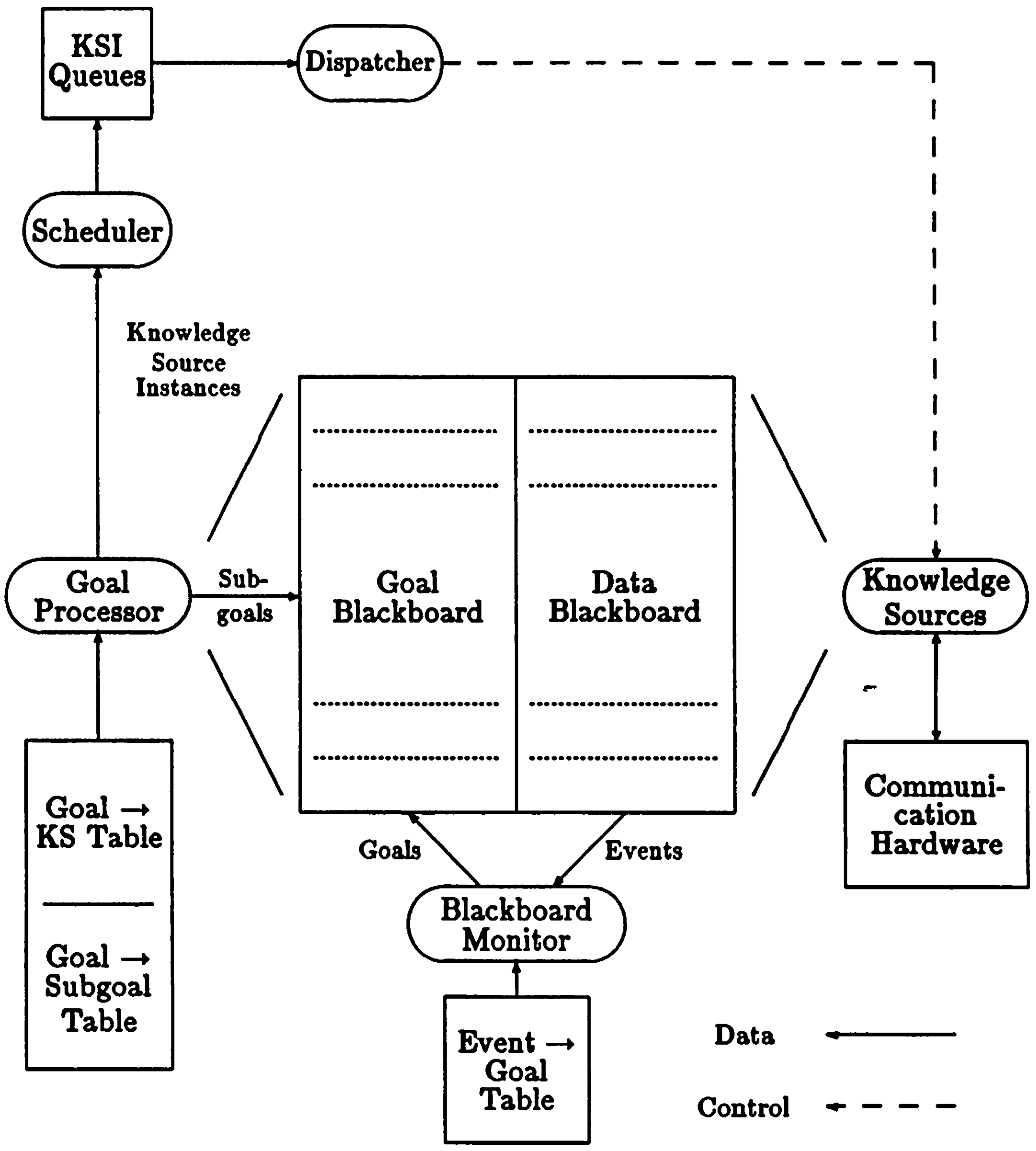


Figure 3: DVMT Node Architecture

that may eventually allow the tracks to be extended. This form of problem solving is called *island-driven* because it uses islands of high belief to guide further low-level processing. This problem solving is also *opportunistic* because nodes can react to highly-rated new data (perhaps received from another node) to form alternative islands of high belief.

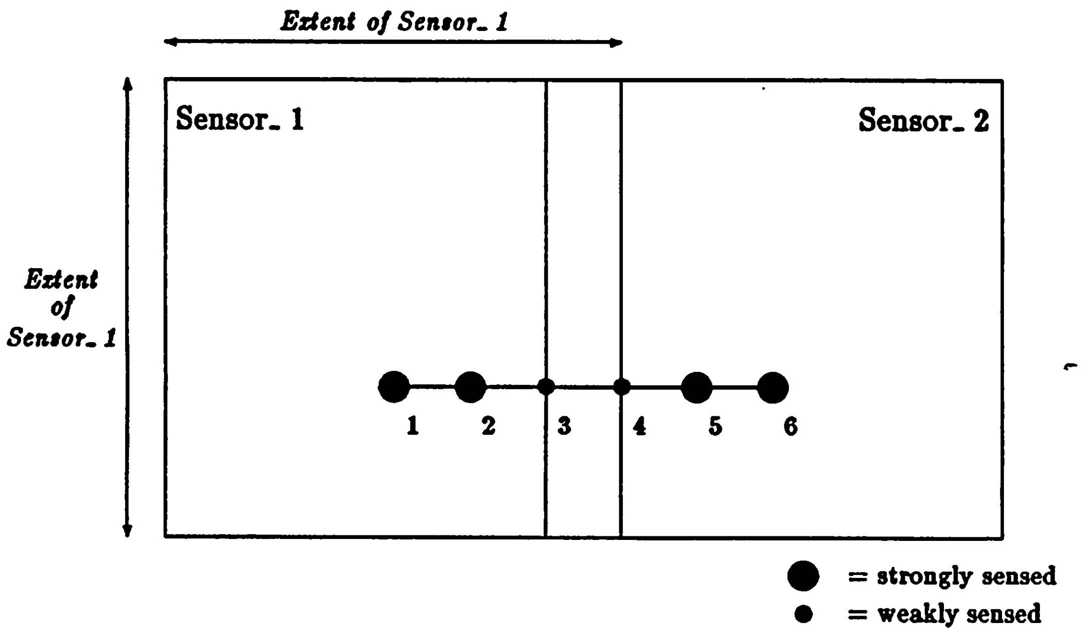
When *any* node generates, through its own uncertain decisions, a hypothesis that "matches" the correct (predefined) solution, the network has solved the problem.[3] As soon as a node finds this solution, it broadcasts its success to the other nodes and then ceases its activity. In turn, the other nodes cease their activity when either they receive this message (which is subject to communication delays) or they generate the solution themselves, whichever is earlier. Network activity thus stops within a finite amount of time (usually a few time units) after any node first finds the solution.

In our environments, low-level processing requires a minimum of three time units to generate a fully supported vehicle location hypothesis (two KSIs to generate two group hypotheses and one KSI to combine these at the vehicle level). With n sequential vehicle locations, high-level processing requires a minimum of n time units to generate a pattern track hypothesis (n-1 to generate the full vehicle track and one to synthesize it to the pattern track level). This analysis provides the framework for an algorithm that has been implemented to calculate a benchmark value for optimal performance. *Optimal* performance corresponds to the minimum amount of time that the network would require if each node had no control uncertainty, and so, made only correct control decisions. But because nodes cannot always be expected to make only correct decisions, we have developed a more subjective benchmark which we call *reasonable* performance. When highly rated noise is present in an environment, a node should temporarily prefer to process it rather than less highly rated but true data. For environments with such noise, reasonable time is computed by adding to the calculation of optimal time an estimate of the amount of time needed to process and dismiss the noise. We shall refer to the optimal and reasonable performance benchmarks throughout this article.

[3]The solution is thus specified before problem solving begins, but a node cannot use this information to guide its processing. Without the availability of such an "oracle", termination of problem solving is much more difficult, requiring nodes to determine whether further problem solving is likely to improve upon a possible solution which it has already generated or will cause it to generate another potentially better solution. In effect, the termination decision depends on the criteria for deciding whether network goals have been satisfied [5]. An effort is underway to address these issues.

4. Organizational Structuring To Increase Coherent Cooperation

Consider the simple two-sensor and data configuration in Figure 4. We might connect both sensors to a single problem solving node and allow it to develop the entire solution. The performance of this environment, as simulated by the DVMT, is shown in Table 1, experiment E1.1. The node required 29 time units (hence 29 KSIs) to generate the solution, 5 more than optimal (E1.2). Despite having had all of the data, the node still had control uncertainty and could not determine an optimal sequence of KSIs. It made 5 incorrect local control decisions to form pattern tracks out of short vehicle tracks; an optimal sequence would have formed the complete vehicle track first.



The data points and their associated times are given. Data at times 3 and 4 fall in the overlapping area. The data points are connected to indicate the vehicle track, the vehicle moving from left to right.

Figure 4: Simple Two-Sensor Configuration and Data.

Next, we connect each sensor to a separate node, each just like the single node in the first case except that they also have the ability to exchange *all* partial results (with a communication delay of one time unit).[4] The resulting performance (E1.3) is actually

[4]All of the two-node environments in this article use a communication delay of one time unit. A node can perform one KSI in the time it takes to transmit a message.

Experiment	Nodes	Organization	Time
E1.1	1	–	29
E1.2	*optimal*(1)	–	24
E1.3	2	none	32
E1.4	2	unbiased	22
E1.5	2	loc-bias	17
E1.6	*optimal*(2)	–	14

Legend

Nodes:	Number of nodes in network (*optimal(n)* = optimal solution time for *n* nodes)
Organization:	Bias of nodes
Time:	Earliest time at which a solution was found

Table 1: Initial Multi-node Experiments.

worse than the single node case! This is because the two nodes, although they exchanged information, were not *organized* to work together coherently. Since they exchanged all partial results, they completely duplicated each other's work, so that each derived the entire solution the same way the single node did. In fact, because of communication delays, each node occasionally performed additional unnecessary KSIs because the partial results needed to continue developing the correct solution had not yet been received from the other node. The unnecessary KSIs constructed less credible partial results along unlikely alternative solution paths. This resulted in performance that was actually worse than in the single node case. What the network needs to improve performance is organization.

Organization can improve performance by reducing the responsibilities of each node. In our simple case, we might make each node responsible for low-level processing only on data it gets from its own sensor[5](for example, node-1 processes only low-level data from sensor-1). We might also make better use of the communication resources by insisting that only high-level partial results, such as vehicle tracks, are exchanged. Making these modifications, network performance is significantly improved (E1.4) although still not optimal (E1.6)—the nodes still make incorrect local decisions. This improved performance also requires much less communication, reducing the number of communicated hypotheses from 172 in E1.3 to only 4 in E1.4.

Organization can also improve performance by forming authority relationships between nodes—by making nodes externally-biased, locally-biased, or unbiased. Authority relationships determine how one node's local view of problem solving will affect the view of another node. For example, if one node has a more accurate or complete view of the problem than another (as in the hierarchical network organizations discussed later), it will be to the network's advantage to let that node's view of the problem (as communicated through

[5]We assume in our environments that overlapping sensors sense the same data in the area of overlap: confidence in data is independent of the number of sensors which observe it. If two nodes independently derive the same partial solution based on independent observations, one of these derivations is redundant (unnecessary). Situations where independent derivation increases confidence are not addressed in this article.

hypotheses and goals) guide the activities of a less informed node. Examination of the simulation results for E1.4 indicates that nodes incorrectly decide to perform significant amounts of redundant work in the overlapping area. Due to the organization, each node first builds a short track based on its uniquely received highly-sensed data, and these tracks are then exchanged. Because each track stimulates work on a different weakly-sensed data point, *each* node works on *both* of these points *in parallel*. Hence, in their overlapping area, each node does low-level processing on both data points, resulting in redundant work and less than completely coherent cooperation. If we modify the organization further so that the nodes are more locally-biased, then each will prefer extending its local track first. Since the extended tracks are also exchanged, each node can avoid the incorrect decisions to perform redundant low-level processing on the other weak data and performance is improved still further (E1.5).

Is it possible to improve the performance even more? As we shall see later, the answer is yes, but improvement is limited. Even with optimal coherent cooperation between nodes—if the nodes always did the best possible activity—two nodes cannot solve the problem in half the time it takes one node. In this environment, optimal solution time for one node is 24 (E1.2), for two nodes is 14 (E1.6), and for three nodes is 11 time units. Cooperating nodes incur processing overhead when they integrate their partial solutions. For example, when one node is generating the overall network solution, either the other nodes are simultaneously generating the same solution (unnecessarily), or they are generating tentative partial solutions that cannot possibly be of use. In either case, we postulate that, *for that interval of time*, we are only using *one-nth* of our *n-node* network. Hence, as the size of the network increases, the effective use of the network over this interval decreases.[6] The processing resources unused during integration are attributed to cooperation overhead.

An effective problem solving network balances cooperation overhead and problem solving parallelism to achieve acceptable network performance. In a small network, cooperation overhead is low (there are fewer partial solutions to exchange and integrate) but network problem solving may be unacceptably slow because each node has too many problem solving tasks. In a large network, each node has fewer tasks, but more partial solutions must be exchanged and integrated; network performance may be degraded by excessive cooperation overhead. An effective problem solving network therefore must have an appropriate number of nodes, and acceptable network performance depends on coherent cooperation among these nodes. In our simple example above, we saw that coherence could be increased by statically allocating task responsibilities, communication paths, and authority relationships. Expanding on these ideas has led to the concept of organizational structuring in distributed problem solving networks.

[6]This analysis holds for these distributed problem solving experiments because the network only develops a single solution and then stops. In *continuous* systems, this may not be the case.

4.1 Organizational Structuring

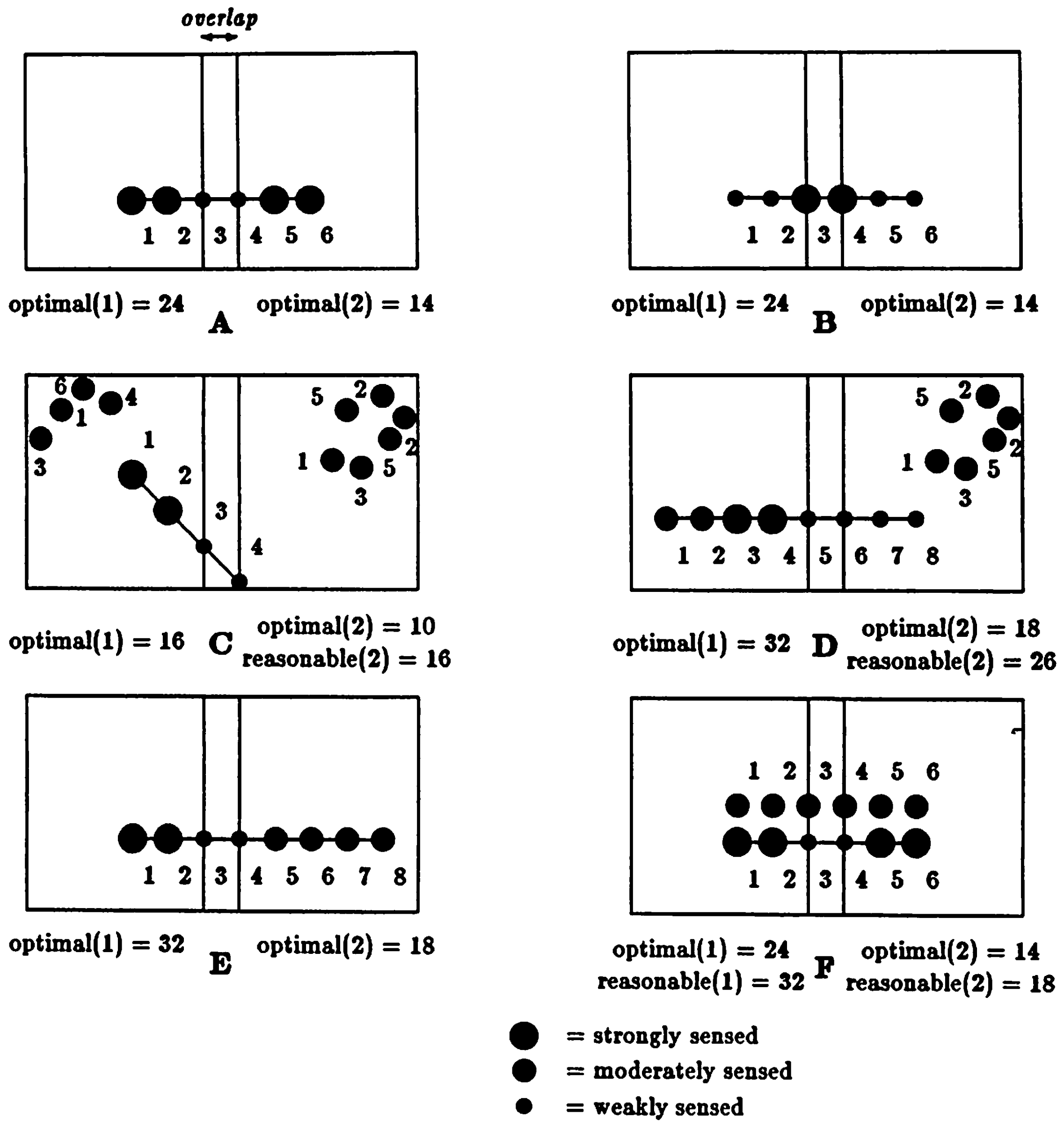
An **organizational structure** specifies a set of long-term responsibilities and interaction patterns for the nodes. This information guides the local control decisions of each node and increases the likelihood that the nodes will behave coherently by providing a global strategy for network problem solving. The construction and maintenance of the organizational structure is a network control problem, and we postpone its discussion until later in this article. For the present, we assume that the organizational structure is established at network creation and is never altered.

The organizational structure is implemented in the DVMT as a set of data structures called **interest areas**. As implied by its name, an interest area specifies the node's interest (represented as a set of parameters) in a particular area of the partial solution space (characterized by blackboard levels, times, locations, and event-classes). The scheduler uses the interest area to modify the ratings of goals; goals to generate hypotheses in desirable areas of the blackboard would have their ratings increased. Goals to transmit or receive information similarly have their ratings modified based on the interest areas of both the sending and receiving nodes. Since the goal rating is a factor in ranking KSIs, the interest areas can influence node activity; but because there are other factors in ranking KSIs (such as the expected beliefs of the output hypotheses), a node still preserves a certain level of flexibility in its local control decisions. The organizational structure thus provides guidance without dictating local decisions, and can be used to control the amount of overlap and problem solving redundancy among nodes, the problem solving roles of the nodes (such as "integrator", "specialist", and "middle manager"), the authority relations between nodes, and the potential problem solving paths in the network [6].

4.2 Further Experiments with Organizational Structuring

In this section, we briefly explore the effects of organizational structuring on some simple two-node environments (Figure 5). Our intent is to show that overly specialized organizational structures allow effective network performance in particular problem solving situations, but that no such organization is appropriate in all situations. We continue using two-node environments to keep the explanations uncomplicated. Experiments on environments composed of larger networks are discussed later in this paper and elsewhere [5,6].

Each of the six two-node environments (A–F) was devised to illustrate a specific problem solving situation. Environment A (which we have previously studied) illustrates the situation where each node develops its own local hypotheses that guide activity in the overlapping area. Environment B illustrates the situation where nodes will process data in the overlapping area first. Environment C illustrates the situation where one of the nodes (node 1, the left node) could potentially derive the solution alone. When the reception of information causes a node to shift its problem solving focus, we say that the node is *distracted* [17]. In Environment D, we illustrate a situation with *positive* (beneficial) distraction: when node 1 sends a highly rated track hypothesis to node 2, node 2 will stop working on its moderately sensed noise and attempt to extend the track with its


overlap
1 2 3 4 5 6
optimal(1) = 24 A optimal(2) = 14
1 2 3 4 5 6
optimal(1) = 24 B optimal(2) = 14
6 4 1 3 1 2 3 4 5 2 2 1 5 3
optimal(1) = 16 C optimal(2) = 10 reasonable(2) = 16
5 2 2 1 5 3
1 2 3 4 5 6 7 8
optimal(1) = 32 D optimal(2) = 18 reasonable(2) = 26
1 2 3 4 5 6 7 8
optimal(1) = 32 E optimal(2) = 18
1 2 3 4 5 6
1 2 3 4 5 6
optimal(1) = 24 reasonable(1) = 32 F optimal(2) = 14 reasonable(2) = 18
● = strongly sensed
● = moderately sensed
● = weakly sensed

For each configuration, the optimal solution time for a one-node network (optimal(1)) and for a two-node network (optimal(2)) is given. For cases where reasonable performance differs from optimal performance, reasonable solution time is also given. The sensed time is given near each data point. Note that noise in C and D cannot be correlated into tracks—data at sequential times are too far apart to be combined into a track.

Figure 5: Six Simple Two-Sensor Configurations.

weakly sensed vehicle data. Environment E illustrates *negative* (detrimental) distraction: when node 2 receives the strongly believed partial track from node 1, it may be tempted to work on the weakly sensed data in the overlapping area (work that node 1 is already doing), delaying important work on the moderate data. Finally, Environment F illustrates a complex problem solving situation where a moderately sensed "ghost" track[7] now runs parallel to the true track. Intelligent local control decisions are crucial in this situation to avoid generating many unnecessary tracks that combine ghost data with true vehicle data.

The experimental results are summarized in Table 2. The results from Environment A are included for completeness (E2.1 – E2.4). Bias will not affect redundancy in Environment B since nodes process data in the overlapping area first. Splitting the overlapping area down the middle and assigning half to each node, however, does improve performance (E2.5, E2.6). In Environment C, such splitting will degrade performance since node 1 cannot complete the solution track by itself and must try to distract node 2 into completing it instead (E2.7, E2.8). Hence, the static division of responsibility in an overlapping area might backfire.

Similarly, a static authority relationship between nodes, determined by their bias, will be more or less useful depending on the problem solving situation. In situations with positive distraction (Environment D), making nodes locally-biased will decrease their responsiveness to received hypotheses and will thus degrade network performance (E2.9, E2.10). However, locally-biased nodes will perform better in situations with negative distraction (Environment E, experiments E2.11, E2.12) because of their "skepticism" [5,23]. Confusion caused by ghost data (Environment F) results in poor network performance regardless of organization (E2.13, E2.14). Mechanisms to help rectify this situation are presented in the next section.

Finally, one-node or *centralized* experiments for Environments B through F are provided (E2.15–E2.19). These help to further illustrate how important the proper organization is in decentralized experiments, as again we see evidence that improper organizations can reduce (or completely negate) the performance improvements achievable through parallelism (for example, compare E2.17 with E2.9).

In summary, we have recognized the strengths and limitations of organizational structuring. By providing even a simple network with some organization, better use of communication and computation resources can be achieved. An organizational structure that appropriately balances the amount of interest that nodes have in various problem solving activities and the degree to which they may be externally biased (based on the relative completeness of their views) can result in consistently acceptable network problem solving behavior over the long term. However, an organization that is specialized for one short-term situation may be inappropriate for another. Because network reorganization may be costly and time consuming (see Section 8), and since specific problem characteristics cannot be predicted beforehand, an organizational structure should thus be chosen which can achieve acceptable and consistent performance in the long-term rather than being very good in a limited range of situations and very bad in others.

[7]Ghost tracks are sequences of noisy data points which mimic (and often parallel) an actual vehicle track.

Experiment	Environment	Nodes	Organization	Overlap	Time
E2.1	A	1	–	–	29
E2.2	A	2	none	all	32
E2.3	A	2	unbiased	all	22
E2.4	A	2	loc-bias	all	17
E2.5	B	2	unbiased	all	26
E2.6	B	2	unbiased	split	20
E2.7	C	2	unbiased	all	21
E2.8	C	2	unbiased	split	25
E2.9	D	2	loc-bias	all	45
E2.10	D	2	unbiased	all	38
E2.11	E	2	unbiased	all	31
E2.12	E	2	loc-bias	all	28
E2.13	F	2	unbiased	all	75
E2.14	F	2	unbiased	split	71
E2.15	B	1	–	–	28
E2.16	C	1	–	–	33
E2.17	D	1	–	–	41
E2.18	E	1	–	–	34
E2.19	F	1	–	–	92

Legend

Environment:	Configuration from Figure 4
Nodes:	Number of nodes in network
Organization:	Bias of nodes
Overlap:	Whether each node is responsible for *all* of overlap or if overlap is *split* between them
Time:	Earliest time at which a solution was found

Table 2: Experiments with Organizational Structuring

5. Planning and Dynamic Refinement to Organizational Structure

To dynamically tailor an "all-purpose" organizational structure for specific network needs, we introduce the second mechanism to increase coherence: a planner. Our planner allows nodes to represent and reason about sequences of related actions. For example, given a highly rated hypothesis, a node typically executes a sequence of KSIs that drive up low level data to extend the hypothesis. However, the entire sequence of KSIs is never on the queue at once because a KSI is only created when the hypotheses it will use have been created, which in turn require the execution of the previous KSI in the sequence. We have therefore developed a structure, called a *plan*, to explicitly represent a KSI sequence.

A plan, as a representation of some sequence of related (and sequential) activities, indicates the specific role that a node will be playing in the organization over a certain time interval. A node can use this additional information to improve its coordination knowledge by dynamically refining its interest areas. For example, consider a node that is responsible both for low-level synthesis activity in a small region and for integrating partial tracks over a much larger region. At a given time, the node might develop and begin to execute a plan to perform synthesis. Given this plan, the node could recognize that it is not integrating partial tracks, and therefore knows that any locally generated or received tracks should be forwarded to other nodes for integration. On the other hand, if the node were executing a plan to integrate partial tracks, it might decide that it should not transmit the larger tracks that it forms because this might occupy bandwidth that could better be used by other nodes to send additional partial tracks to it for integration.

5.1 Implementation of the Planner

Each *plan* represents a desire to achieve a high-level goal by performing a sequence of activities. To identify plans, the node needs to recognize these high-level goals. Inferring high-level goals based on pending KSIs is an inappropriate approach; it is like attempting to guess a chess opponents strategy after seeing a single move. Furthermore, the hypothesis and goal blackboards provide information at too detailed a level to efficiently infer these high-level goals. What is required is a structure similar to the blackboards where related hypotheses and goals are grouped together. We have developed a preliminary version of this structure which we call the *abstracted blackboard*, a structure reminiscent of the focus-of-control database first used in the Hearsay-II speech understanding system [10,13]. Our implementation of the abstracted blackboard is incomplete because it does not adequately incorporate the information from the goal blackboard. However, for the type of processing performed in these environments, hypothesis abstraction is usually effective.

Hypotheses with similar level, time, and region characteristics are grouped together on the abstracted blackboard. This grouping acts as a smoothing operator, obscuring details about individual hypothesis interactions so that broader, long-term interactions between areas of the solution space can be discerned. By transforming the data blackboard into the abstracted blackboard, we explicitly generate a state representation that is uniquely appropriate for planning at a higher level of abstraction.

The implementation, which is more completely outlined elsewhere [9], clusters hypotheses and summarizes their attributes into a set of values that are stored on the abstracted blackboard. A *situation recognizer* scans the abstracted blackboard to develop a higher level view of the problem solving occurring at the node. Based on this view, it generates higher level goals and passes them to the planner. A goal might be to extend a highly believed portion of the blackboard into a new area or to improve credibility in another area. The *planner* in turn creates a plan to satisfy the higher level goal, and associates with the plan a set of KSIs that represent potential steps in the plan. This set is updated as the plan is carried out and appropriate new KSIs are formed.

At any given time, a node will work on its most highly rated plan; problem solving remains opportunistic because plans are interruptable. The plan rating reflects the importance of achieving the higher level goal, the expected performance of the plan's KSIs (reflected by their ratings), and the interest area specifications. We have therefore made important modifications to the control structure of a node (Figure 6). As the figure indicates, the creation and ranking of plans requires the planner to integrate the influences of the long-term strategy of the organizational structure (the interest areas), the medium-term higher-level view of the current situation (the abstracted blackboard), and the short-term KSI input indicating actions that can be achieved immediately (the KSI queue). Hence, decisions in a plan-based node are more informed than those in a KSI-based node (a node without plans).

5.2 Experiments with Plan-based Nodes

A planner increases a node's self-awareness, and the problem solving of a node is improved. In this section, we illustrate this improvement using some simple one and two node environments, all based on the configurations from Figure 4. The results for these experiments are summarized in Table 3. Further experiments on larger networks will be presented later in this article.

We begin by considering centralized environments. Comparing the performance when nodes are plan-based (E3.1–E3.6) to when they are KSI-based (Table 2, E2.1 and E2.15–E2.19), we see that performance is improved. Indeed, in Experiments E3.1 through E3.5, the centralized performance is optimal. When there is significant distraction due to the ghost track (Environment F), distraction affects both types of nodes (neither finds the solution in optimal time), but the high-level view of a plan-based node helps it quickly recognize that the distracting data will not satisfy the high-level goal which drives the plan. For example, the high-level goal may be to create a hypothesis which extends a hypothesis with two time-locations to three time-locations. Distracting data that initially looks like it might satisfy this goal is developed to the point where the system can determine that, in fact, further work along this line will not contribute to achieving the goal. At this point, a plan to develop other data which could satisfy the high-level goal becomes the highest rated plan. Hence, the time spent deviating from the correct solution path is reduced, and the plan-based node can generate the solution faster.

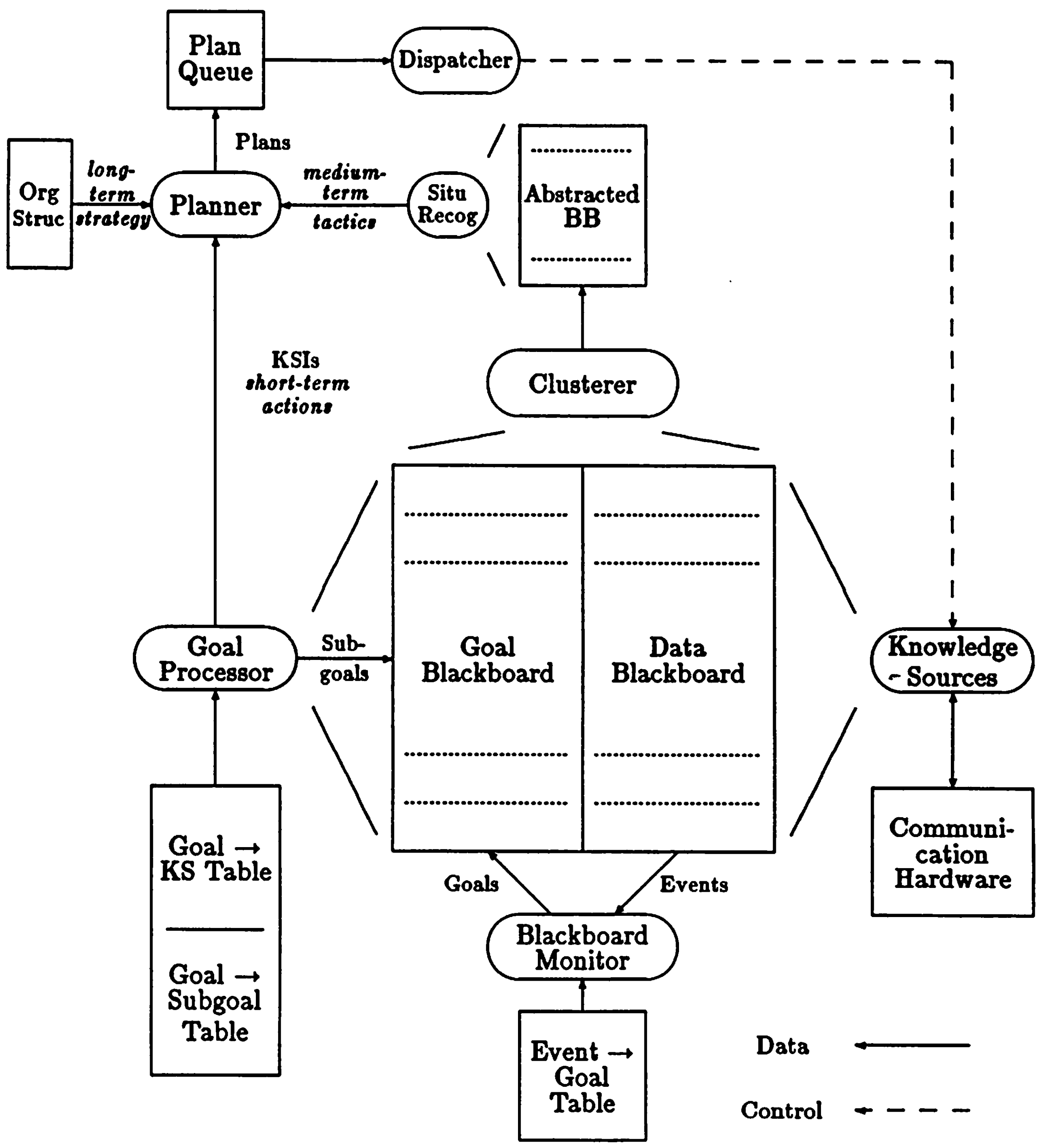


Figure 6: The Modified Problem Solving Architecture of a Node.

Experiment	Environment	Nodes	Organisation	Overlap	Time
E3.1	A	1	–	–	24
E3.2	B	1	–	–	24
E3.3	C	1	–	–	16
E3.4	D	1	–	–	32
E3.5	E	1	–	–	32
E3.6	F	1	–	–	32
E3.7	A	2	unbiased	all	14
E3.8	A	2	loc-bias	all	14
E3.9	B	2	unbiased	all	19
E3.10	B	2	unbiased	split	16
E3.11	C	2	unbiased	all	16
E3.12	C	2	unbiased	split	16
E3.13	D	2	loc-bias	all	35
E3.14	D	2	unbiased	all	27
E3.15	E	2	loc-bias	all	18
E3.16	E	2	unbiased	all	25
E3.17	F	2	unbiased	all	18
E3.18	F	2	unbiased	split	18

Legend

Environment:	Configuration from Figure 4
Nodes:	Number of nodes in network
Organisation:	Bias of nodes
Overlap:	Whether each node is responsible for *all* of overlap or if overlap is *split* between them
Time:	Earliest time at which a solution was found

Table 3: Experiments with the Planner

We now examine the two-node cases, where each node has a limited local view of network problem solving. Having established that the problem solving behavior of a plan-based node is more effective than that of a KSI-based node, we expect that a network of such nodes might have improved performance, not because they display better "teamwork" (their global knowledge does not increase), but rather because each is a better "player".

These expectations were empirically verified on the environments. In Environment A, the solution is found at optimal time whether or not the plan-based nodes are locally-biased (E3.7, E3.8). Each node will complete its plan to extend its locally created track before it begins extending the received track since the plan to extend the local hypothesis is formed first. In Environment B, dividing the overlapping area into separate regions of preference still has a positive effect, as would be expected (E3.9, E3.10). However, in Environment C, the division no longer has deleterious effects (E3.11, E3.12), because the new planner places greater emphasis on extending existing tracks rather than developing new tracks from signal data. Note that, since node 1 must generate the entire solution, performance in the two-node networks is the same as in the centralized experiment (E3.3).

The degree of local-bias in a node, and hence its susceptibility to distraction, still has important consequences in plan-based nodes. In Environment D, locally-biased plan-based nodes, while performing better than their KSI-based counterparts, still have performance far from optimal (E3.13). Unbiased plan-based nodes, on the other hand, have performance which is very nearly optimal (E3.14). Alternatively, in the environment featuring negative distraction (Environment E), the locally-biased plan-based nodes give optimal performance (E3.15), while the unbiased nodes once again display the negative distraction (E3.16). Hence, these environments serve to illustrate that improving the planner of each node is not enough to insure coherent cooperation—even though the nodes are better "players", dealing with distraction often requires nodes to have more knowledge about each other's activities.

The ghost track in Environment F can be more effectively handled by plan-based nodes, as we saw in the centralized case. Network performance is still not optimal—the node still requires time to examine and rule out the noisy data—but performance is reasonable (E3.17). Furthermore, note that division of the overlapping area does not make a difference (E3.18), for the same reasons that it did not affect Environment C with plan-based nodes.

In conclusion, the increased self-awareness afforded by the new planning mechanisms significantly improves problem solving performance. Similar results are found in experiments with the larger networks described later in the article. The experiments thus emphasize the importance of sophisticated local control that recognizes and reacts appropriately to various problem solving situations. Our future work will include expanding the repertoire of situations that can be dealt with so that plans can be developed in more complex environments.

6. Communication and Increasing Global Awareness

In the previous two sections, we outlined two mechanisms for increasing coherence in distributed problem solving networks by increasing the knowledge available to the node, both in terms of its general network responsibilities (organizational structure) and understanding how its current state can refine this view (planning). Unfortunately, even with this added sophistication, the nodes still may not always be completely coherent. Although the control uncertainty based on its own state has been reduced, a node will still be uncertain about the organizational role being played by each of the other nodes. The organizational structure limits the possibilities, but to retain network flexibility, each node will not be overly limited. To dynamically refine their views of the roles being played by the other nodes, nodes must exchange information.

There are three major characteristics of the information communicated among nodes that affects coherence: relevance, timeliness and completeness. *Relevance* measures, for a given message, the amount of information that is consistent with the solution derived by the network. Irrelevant messages may distract the receiving node into wasting its processing resources on attempts to integrate inconsistent information, so higher relevance of communicated information can result in more global coherence (since this information stimulates work along the solution path). *Timeliness* measures the extent to which a transmitted message will influence the current activity of the receiving node. Since timeliness depends not only on the content of the message but also on the state of the nodes, a message's timeliness can vary as node activity progresses. If the transmitted information will have no effect on the node's current activity, there is no point in sending it; however, if the transmitted information will distract the receiving node to work in a more promising area, or if the node needs the information to continue developing a promising partial solution, then it is important that the information be sent promptly. Finally, *completeness* of a message measures the fraction of a complete solution that the message represents. Completeness affects coherence by reducing the number of partially or fully redundant messages communicated between nodes—messages which negatively distract nodes into performing redundant activity. Furthermore, as the completeness of received messages increases, the number of ways that the messages can be combined with local partial results decreases due to their larger context. Finally, achieving completeness is important to minimize communication requirements in our loosely-coupled distributed system.

It is important to note that these three characteristics of communicated information are not independent. For example, higher completeness leads to higher relevance but also a potential decrease in timeliness. Communication policies that guide decisions about what information should be sent, to what nodes, and when, often involve tradeoffs among the three characteristics. With increased self-awareness, a node can be more informed about the relevance and completeness of its local hypotheses and can make more intelligent predictions both about how a hypothesis will affect its local decisions and about whether the timely transmission of the hypothesis is therefore likely to lead to positive or negative distraction. These decisions will be based on the node's communication policy.

6.1 Communication Policies and Coherence

When a node receives a partial solution, it may be distracted into performing actions intended to increase the completeness of this partial solution. But the node that sent the partial solution might also be attempting to increase its completeness. Sending a partial solution is therefore risky: it might positively distract the recipient node into extending the partial solution into new areas, or it might negatively distract the recipient node into duplicating the actions of the sending node. To avoid negative distraction, the sending node might only transmit its most complete partial solutions, but this can reduce the timeliness of sending positively distracting messages. Increasing coherent cooperation by selective communication of partial results thus requires communication policies that consider the tradeoffs between timeliness and completeness. The policies we have formed are extensions to the ideas first described by Lesser and Erman [17].

In the most simple communication policy, any partial solution that could be of interest to another node (according to the organizational structure) is sent as soon as possible. This *send-all* policy allows a node to transmit small or irrelevant hypotheses even when the node's subsequent processing will immediately improve this data or recognize its irrelevance. These messages unnecessarily burden the communication channels and can negatively distract the recipient node. For example, the recipient node may combine these smaller hypotheses together into a larger hypothesis—a redundant activity, since the sending node both generates and transmits this larger hypothesis as well.

With the *locally-complete* policy, a node transmits only its most complete hypotheses—those hypotheses it cannot improve upon locally. This policy diminishes negative distraction at the potential cost of reducing timely positive distraction, and minimizes the number of transmissions while maximizing the completeness and relevance of each message. For example, in Environments D and E, node 1 quickly generates a highly believed, short, locally-incomplete hypothesis. With the locally-complete policy, this hypothesis would not be transmitted. In Environment D, we would expect this to degrade performance, but in Environment E, we would expect performance to improve.

A third communication policy, called *first-and-last*, selectively transmits some locally-incomplete hypotheses to avoid the degradation of positive distraction. It is essentially the locally-complete policy with the added stipulation that hypotheses which do not incorporate any previously communicated information can be transmitted even if they are locally-incomplete. Hence, the first partial hypothesis will be transmitted for predictive information, and the last (locally-complete) version will be sent for integration information, but any intervening (locally-incomplete) versions will not be transmitted.

Because plans make predictions about activity in the near future, a node can determine a hypothesis' local completeness by comparing it to the current plan. If that plan does not represent an intention to improve upon the hypothesis, then the hypothesis is considered locally-complete. Thus, a node working on a low-level synthesis plan to extend a track will transmit other tracks elsewhere for integration while a node planning to integrate partial tracks will refrain from sending them elsewhere. As our understanding of problem solving activities increases and our representations of plans improves, nodes will be better able to determine local completeness for hypotheses that may be improved by plans "near" the top of the plan queue, and communication decisions will be further improved. Meta-level communication (Section 6.3) can improve these decisions still further by allowing a node to better predict the effects of a message on the recipient node(s).

6.2 Evaluating Communication Policies

We now investigate how the communication policies affect performance in some of our simple two-node environments (Figure 4). The results are given in Table 4. Although our discussion below focuses on the effects of the policies on coherent cooperation, it should be noted that they also affect the communication resource use. Therefore, we indicate the number of hypotheses exchanged in each case, and here note that the more selective policies of locally-complete and first-and-last generally result in fewer transmissions (except when these strategies degrade network performance, such as E4.2). In these small environments, the reduction in communication due to these policies is not spectacular; in larger environments, however, the results can be much more dramatic, as we describe in Section 7.1.

The locally-complete and first-and-last policies degrade performance in Environment A because nodes withhold the most useful hypotheses (consisting of three locations) causing redundant processing in the overlapping area (E4.1–E4.3). In environment B, communication policy has no effect if the organization does not split the overlapping area (E4.4–E4.6); since the nodes begin by working in the center of the solution and progress outward, the tracks involving the end points are locally-complete and will be transmitted in all cases. However, by splitting the overlapping area, locally-complete is once again the worst, with first-and-last between it and send-all in terms of performance, because once again waiting for completeness causes redundant processing in the overlapping area (E4.7–E4.9).

In both environments with distraction (Environment D and Environment E), the communication policy does not make a significant difference if the nodes are locally-biased because the reception of hypotheses (or lack thereof) does not affect local problem solving decisions (E4.13–E4.15, E4.19–E4.21). With unbiased nodes, the locally-complete policy degrades performance in situations with positive distraction (Environment D); as expected, the first-and-last policy rectifies this (E4.10–E4.12). By withholding negatively distracting hypotheses, the locally-complete policy improves performance in Environment E (E4.16–E4.18). Finally, in Environment F, the locally-complete policy performs more poorly (E4.22–E4.24) for the same reasons as in Environment A.

In summary, then, communication policies only substantially affect network performance if nodes are unbiased because the policies depend on distraction. As expected, a locally-complete policy handles negative distraction by holding back the distracting hypothesis, while first-and-last is more appropriate for positive distraction because it sends the predictive information earlier. The experimental results indicate that the judicious exchange of partial results can increase coherent cooperation among nodes and decrease communication resource requirements, but that no single static communication policy appears appropriate for all problem solving situations. Although it decreases communication costs, therefore, selective communication of partial results does not represent a flexible

Experiment	Environment	Policy	Organisation	Overlap	Time	Hyps
E4.1	A	*S*	unbiased	all	14	4
E4.2	A	*L*	unbiased	all	18	6
E4.3	A	*F*	unbiased	all	17	2
E4.4	B	*S*	unbiased	all	19	10
E4.5	B	*L*	unbiased	all	19	6
E4.6	B	*F*	unbiased	all	19	6
E4.7	B	*S*	unbiased	split	16	8
E4.8	B	*L*	unbiased	split	20	4
E4.9	B	*F*	unbiased	split	18	8
E4.10	D	*S*	unbiased	all	27	15
E4.11	D	*L*	unbiased	all	33	8
E4.12	D	*F*	unbiased	all	26	7
E4.13	D	*S*	loc-bias	all	35	17
E4.14	D	*L*	loc-bias	all	35	9
E4.15	D	*F*	loc-bias	all	35	9
E4.16	E	*S*	unbiased	all	25	12
E4.17	E	*L*	unbiased	all	18	5
E4.18	E	*F*	unbiased	all	25	8
E4.19	E	*S*	loc-bias	all	18	11
E4.20	E	*L*	loc-bias	all	18	5
E4.21	E	*F*	loc-bias	all	19	6
E4.22	F	*S*	unbiased	all	18	8
E4.23	F	*L*	unbiased	all	26	8
E4.24	F	*F*	unbiased	all	18	4

Legend

Environment:	Configuration from Figure 4
Policy:	S is send-all, L is locally-complete, F is first-and-last
Organisation:	Bias of nodes
Overlap:	Whether each node is responsible for *all* of overlap or if overlap is *split* between them
Time:	Earliest time at which a solution was found
Hyps:	Number of exchanged hypotheses

Table 4: Two-node Experiments with Communication Policies

approach to increasing coherent cooperation.

6.3 Meta-level Communication

An exchanged hypothesis implicitly provides details about the local problem solving at its creating node. Given the additional capabilities of the modified, plan-based nodes to make predictions about near-future activity, we might expect nodes to be better at extracting useful coordination information from messages. However, the information contained in these messages is targeted toward improving the problem solving activity of the nodes, not toward improving the coordination between them. To improve coordination, messages should contain less detailed information about specific actions and more general information about the current and planned problem solving activities of the node. Hence, we argue that communication between nodes should not be limited to *domain-level* information (partial solutions) but should also include the exchange of meta-level information—information specifically intended to increase coherent cooperation in the network. Indeed, it is likely that we will want to exchange information on any number of levels depending on the intended use of that information.

Abstracted blackboard entries summarize a node's hypotheses, and thus provide a high-level view of where a node has searched. By exchanging portions of the abstracted blackboards, nodes can reason about the *past* activities of their neighbors. Furthermore, a node that knows the current plan of its neighbor can reason about its *present* actions. Reasoning about the *future* actions of a node, however, is a complex problem. Not only must estimations be made about the duration and effects of a node's plans, but also about what further information the node may receive (from another node or from its sensors) that could affect these estimations. We are currently developing mechanisms to generate such estimations and expectations.

Our current implementation assumes that a node can make completely accurate short-term predictions about future activity based solely on the plan queue. We simulate this best-case scenario by letting nodes access the abstracted blackboards and plan queues of other nodes. In Section 7, we briefly explore what happens when a node must work with out-of-date meta-level information because of channel delays. Eventually, nodes will have to perform more complex reasoning about when to send meta-level information to each other, and in doing so will have to consider relevance, timeliness and completeness issues.

When developing a plan, a node can use meta-level information to determine if the plan will redundantly derive information that another node has either generated (present in the abstracted blackboard) or is in the process of generating (the top plan). By avoiding redundant activity, solution generation rate can improve because less highly rated but potentially useful activities will be invoked earlier (rather than redundant invocation of highly rated activities). Meta-level information also allows nodes to better predict the effects a message will have on other nodes, thereby improving communication decisions.

6.4 Evaluating Meta-level Communication Utility

To evaluate the benefits and costs of exchanging meta-level information, the improvement in network computational performance (using the computation resources more effectively) must be weighed against the additional communication resource usage. In the experiments in this section, we assume that there are sufficient communication resources to accommodate both the exchange of hypotheses (nodes use the send-all policy) and the additional meta-level messages. In the next section, we perform an empirical study of the costs and benefits of the planning mechanisms and of meta-level communication.

The experimental results (Table 5) indicate that the distributed problem solving networks with organizational structuring, plan-based nodes, and meta-level communication can often be completely coherent in their cooperation—they could not have acted as a better team. In Environment A, the solution is found in optimal time regardless of bias (E5.1, E5.2), as was the case without meta-level communication (Table 3, E3.7, E3.8). Meta-level communication is advantageous in Environment B (E5.3, E5.4). Splitting the overlapping area among the two nodes no longer has an effect because now nodes can avoid redundancy dynamically. In essence, by communicating meta-level information, the nodes *are* splitting the overlapping area, but they are doing so dynamically rather than due to a static organizational decision.[8] The performance is slightly suboptimal because nodes unnecessarily exchange and integrate short (locally-incomplete) hypotheses.

In Environment D, locally-biased nodes still perform more poorly (E5.5, E5.6). In Environment E, however, it is no longer necessary to make nodes locally-biased to prevent negative distraction because the exchange of meta-level information allows unbiased nodes to dynamically assign responsibility in the overlapping area (E5.7, E5.8). Meta-level communication does not improve performance in Environment F (compare E5.9 and E5.10 with Table 3, E3.17 and E3.18) because the performance in these earlier experiments was already equal to what one could reasonably expect in a completely coherent network.

Based on these results, we can make several important conclusions. First, meta-level communication will sometimes not improve performance, particularly if that performance is already about as good as the network could reasonably achieve. The exchange of meta-level information can thus increase communication without improving performance. There are important cost and benefit tradeoffs to be considered in the use of meta-level communication, and we will explore this issue more fully in the next section.

A second conclusion concerns the use of bias to affect coherence. In the two-node environments, neither node inherently has a better view of network problem solving than the other—each is responsible for as much of the sensed area as the other. By exchanging meta-level information, the nodes will have increased but still roughly equivalent views of network activity. Since the nodes can use this meta-level information to make informed decisions about how to react (if at all) to a received hypothesis, they may remain unbiased and still avoid negative distraction (e.g. E5.8). Statically biasing these nodes, on the other

[8]The first node to work on some part of the overlapping area essentially claims that part: the meta-level information reflects this work and causes other nodes to avoid redundantly working there. The overlapping area is thus split as each node claims different portions of it.

Experiment	Environment	Nodes	Organization	Overlap	Time
E5.1	A	2	unbiased	all	14
E5.2	A	2	loc-bias	all	14
E5.3	B	2	unbiased	all	15
E5.4	B	2	unbiased	split	15
E5.5	D	2	loc-bias	all	36
E5.6	D	2	unbiased	all	26
E5.7	E	2	loc-bias	all	18
E5.8	E	2	unbiased	all	18
E5.9	F	2	unbiased	all	18
E5.10	F	2	unbiased	split	18

Legend

Environment:	Configuration from Figure 4
Nodes:	Number of nodes in network
Organization:	Bias of nodes
Overlap:	Whether each node is responsible for *all* of overlap or if overlap is *split* between them
Time:	Earliest time at which a solution was found

Table 5: Experiments with Meta-level Communication

hand, can result in poorer performance (e.g. E5.5). We therefore conclude that, when nodes have equally complete views of the problem, limiting the choices of nodes by imposing a static bias may be inappropriate. However, bias can still be useful if some nodes have more complete views than others due to a hierarchical organization (see next section).

The final conclusion is that meta-level communication can significantly improve coherent cooperation among overlapping nodes. By making small refinements to their organizational structures based on exchanged meta-level information, nodes can dynamically divide responsibility in their overlapping areas as problem solving progresses. A specific, static division of these areas may sometimes degrade network behavior since one node cannot dynamically take on more responsibility in the area if its neighbor is working elsewhere. However, to completely organize the nodes dynamically using meta-level information would be infeasible due to the limited communication bandwidth, particularly in large, complex networks where each node must coordinate with many others. Our approach to coherent cooperation therefore stresses the use of both an organizational structure to act as a general, long-term framework for coordination and of meta-level communication to make small, short-term refinements to this framework based on the current problem solving activities.

7. Further Experiments in Coherent Cooperation

In this section, we attempt to achieve two goals. The first goal is to show that the results obtained in the two-node environments are indicative of larger environments and the conclusions we have drawn therefore carry over to more complex distributed problem solving networks. The second goal is to to determine when the improvements in network performance justify the additional computation and communication costs of our new mechanisms.

7.1 Experiments with Larger Problem Solving Networks

As the number of nodes in distributed problem solving networks increases, it becomes significantly more difficult to describe (much less explain) network behavior. Therefore, all of our earlier discussions on experimental results were based on very simple one- or two-node environments. However, most of our research has been devoted to larger networks that display more interesting and complex behavior. Below, we briefly summarize these experiments on larger networks, indicating how increased node sophistication and the use of meta-level communication can improve performance.

The larger networks are based on two different sensor and data configurations. The first of these is has 4 sensors and 16 data points (Figure 7), eight as part of the vehicle track and the other eight running parallel to it as a ghost track (recall Environment F, Figure 4). Note that the vehicle data is strongly sensed at each end and is weakly sensed in the middle, while the ghost data is moderately sensed throughout.

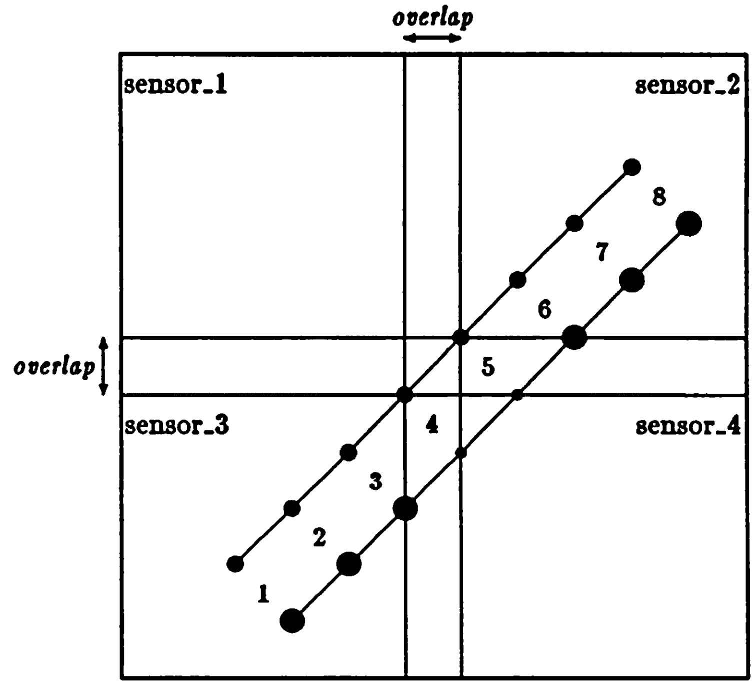


Figure 7: Four-Sensor Configuration with Sensed Data.

Upon this configuration we base three networks. The first is the single-node, centralized case, where one node receives data from all four sensors. The second is a four-node network where each node is attached to a different sensor. As in the two-node environments, the nodes are organized *laterally*, meaning that they exchange information among themselves and each attempts to form the network solution. The third network has five nodes. Once again, four of the nodes are connected to their own sensors, but now they all communicate only with the fifth node (which receives no sensor data). In this *hierarchical* organization, the four nodes attached to the sensors perform low-level processing tasks on their received sensor data, while the fifth node performs high-level integration tasks to form an overall solution from the hypotheses it gets from the other nodes.

In hierarchical organizations, achieving authority relationships through the use of bias becomes more important. For example, in the five node network the fifth node can have a much more complete view of network problem solving than any of the four "low-level" nodes, and thus should have more control over network problem solving. We achieve this by making the low-level nodes externally-biased while the fifth node is locally-biased. Problem solving in such hierarchical organizations will often seem more focused and, if the top-level node has an appropriate view, network performance can improve [6].

The second sensor configuration has 10 sensors and 36 data points (Figure 8). Half of the data corresponds to the vehicle track and the other half to a parallel ghost track. The vehicle track is composed of four sections of strongly sensed data surrounding three sections of weakly sensed data; the ghost track is moderately sensed throughout.

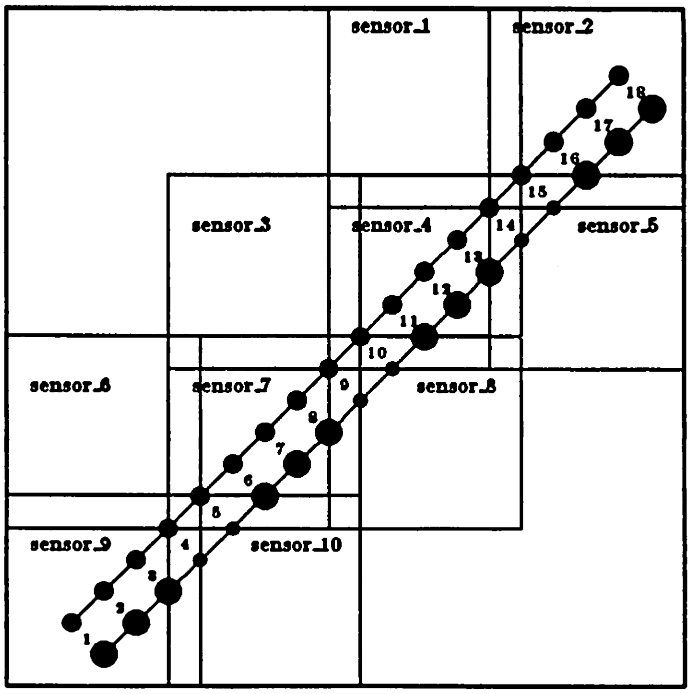


Figure 8: Ten-Sensor Configuration with Sensed Data.

Once again, we have developed three networks based on this sensor configuration. The first is the centralized case where all sensor data is processed by one node. The second is a lateral ten-node network, where nodes with overlapping sensed areas may exchange information and each node tries to generate the network solution. The third network has thirteen nodes in a three level hierarchy. At the "base-level" are ten nodes, each connected to a different sensor, that perform the low-level processing. Each of these nodes communicates with one of the two "intermediate-level" nodes, each of which integrates the data from its five subordinate nodes into larger, more complete hypotheses. These hypotheses are then sent to the single "top-level" node, which combines them into an overall network solution. Network problem solving can once again be more focused if this top-level node can exert authority over the intermediate-level nodes, which in turn can exert authority over the base-level nodes.

Ghost data can complicate network problem solving. In the centralized cases, the ghost data can degrade performance because the node must examine it and discard it before considering the weakly sensed vehicle data. In the lateral organizations, the exchange of short tracks formed from ghost data can distract nodes from important low-level tasks. In hierarchical organizations, the higher level nodes must examine and discard ghost data, and the propagation of information around the hierarchy can slow down network problem solving even though that problem solving can be more focused.

In Table 6, we summarize the effects of the three mechanisms to improve local control. In the centralized cases (E6.1 – E6.8), the planner enables nodes to achieve reasonable performance. Similarly, in both laterally and hierarchically organized networks, nodes also benefit from the planner, and meta-level communication allows performance that is generally as effective as is reasonable in a completely coherent network. This is true in the four-node (E6.9 – E6.13), five-node (E6.14 – E6.18), ten-node (E6.19 – E6.23), and thirteen-node (E6.24 – E6.28) networks. Finally, the experiments indicate that laterally organized nodes can be as focused as hierarchically organized nodes when they have increased self- and network-awareness (from the planner and meta-level communication).[9]

The intelligent use of the communication channels becomes increasingly important in the larger environments for two reasons. First, because the number of partial tracks increases quadratically with the number of data points, the blind exchange of all partial tracks quickly becomes infeasible. Second, as partial tracks get longer in laterally organized environments, they will be of interest to more and more nodes, so a single partial track might have to be propagated to many nodes in the network. Combining these reasons, it becomes clear that a send-all policy will not be feasible in many large environments. Unlike the two-node networks where the policy had little effect on the number of hypotheses exchanged (Table 4), the policy becomes crucial in these larger networks. Indeed, the ten-node and thirteen-node environments outlined above used the locally-complete communication policy to reduce the combinatorial explosion of transmissions—simulating

[9]This requires *all* of the lateral nodes to be well informed, meaning that large amounts of meta-level communication must be exchanged compared to the hierarchical case where few nodes have the better view of network activity. Achieving more focused network activity may thus be more expensive in lateral organizations (although such lateral organizations may be more tolerant of node failures).

Experiment	Nodes	Sensors	OS	Planner	MLC	Other	Time
E6.1	1	4	–	no	–	–	102
E6.2	1	4	–	yes	–	–	40
E6.3	1	4	–	–	–	reasonable	40
E6.4	1	4	–	–	–	optimal	32
E6.5	1	10	–	no	–	–	262
E6.6	1	10	–	yes	–	–	96
E6.7	1	10	–	–	–	reasonable	96
E6.8	1	10	–	–	–	optimal	72
E6.9	4	4	yes	no	no	–	33
E6.10	4	4	yes	yes	no	–	26
E6.11	4	4	yes	yes	yes	–	15
E6.12	4	4	–	–	–	reasonable	15
E6.13	4	4	–	–	–	optimal	15
E6.14	5	4	yes	no	no	–	31
E6.15	5	4	yes	yes	no	–	28
E6.16	5	4	yes	yes	yes	–	17
E6.17	5	4	–	–	–	reasonable	16
E6.18	5	4	–	–	–	optimal	16
E6.19	10	10	yes	no	no	–	46
E6.20	10	10	yes	yes	no	–	28
E6.21	10	10	yes	yes	yes	–	18
E6.22	10	10	–	–	–	reasonable	18
E6.23	10	10	–	–	–	optimal	18
E6.24	13	10	yes	no	no	–	44
E6.25	13	10	yes	yes	no	–	36
E6.26	13	10	yes	yes	yes	–	23
E6.27	13	10	–	–	–	reasonable	20
E6.28	13	10	–	–	–	optimal	20

Legend

Nodes:	Number of nodes in network
Sensors:	Number of sensors in configuration
OS:	Use of organizational structuring
Planner:	Use of the planner
MLC:	Nodes exchange meta-level information
Other:	Indicates calculated time for reasonable or optimal performance
Time:	Earliest time at which a solution was found

Table 6: Experiments with Larger Configurations

these environments with the send-all policy has been infeasible because each node needs excessive amounts of memory to store all of the exchanged hypotheses.

In conclusion, our mechanisms improve network performance in large as well as small environments. Indeed, some of these mechanisms are vital in larger node networks, especially the mechanisms that allow more intelligent use of the communication resources.

7.2 The Costs of Increasing Coherent Cooperation

Throughout this article, our discussions have emphasized the issues in increasing coherent cooperation among overlapping nodes. By statically allocating different portions of the overlapping area to the nodes, we reduced redundant processing in the area, but risked delaying the timely coverage of the entire area. Through meta-level communication, overlapping nodes can dynamically assign responsibility in the overlapping area, and thus can cooperate more coherently. However, there is a cost associated with the increased capabilities of nodes to make more informed local control decisions.

We investigate the costs of our mechanisms after first distinguishing between two reasons for exchanging meta-level information. The first is to avoid immediate redundancy—the local planner will use meta-level information to decide whether a potential plan represents redundant work. The second is to avoid future redundancy—the local planner will use meta-level information to focus the node on areas that are least likely to *eventually* lead to performing redundant work. For example, consider what might happen if two nodes share four sequential data points and only exchange track hypotheses. If node-A begins working on the first of these points, where should node-B work? To avoid immediate redundancy, node-B will not work on the same data as node-A. Furthermore, if node-B works on the second data point, it will force node-A into performing redundant work later on since node-A must combine the first data point with the second to form an exchangeable track hypothesis. Node-B should therefore avoid future redundancy, perhaps by working as far away from where node-A is working as possible. Avoiding future redundancy will require more meta-level messages to be exchanged because nodes will need to know about their neighbors activities near the overlapping area as well as in it.

We study the costs and benefits of our mechanisms on a set of environments with different amounts of overlap between nodes (Figure 9). The environments are variations on the four-node laterally-organized networks described previously. Again, each node receives data from one sensor, but we modify the amount of overlapping data nodes have by altering the sensitivity of each sensor so that it picks up more or less data. In all cases, each data point is received by at least one sensor; in Environment OV22, all data is received by all four sensors so that each node has a complete view of the problem.

For each of these five environments, we ran four experiments. In the first, the nodes were given a lateral organization and were unbiased. In the second, the nodes were also given the planner. In the third, nodes could furthermore exchange meta-level information to avoid immediate redundancy, and in the fourth, nodes could exchange meta-level information to avoid future redundancy as well. Measurements for each experiment indicate network performance (as solution time) and the computation and communication

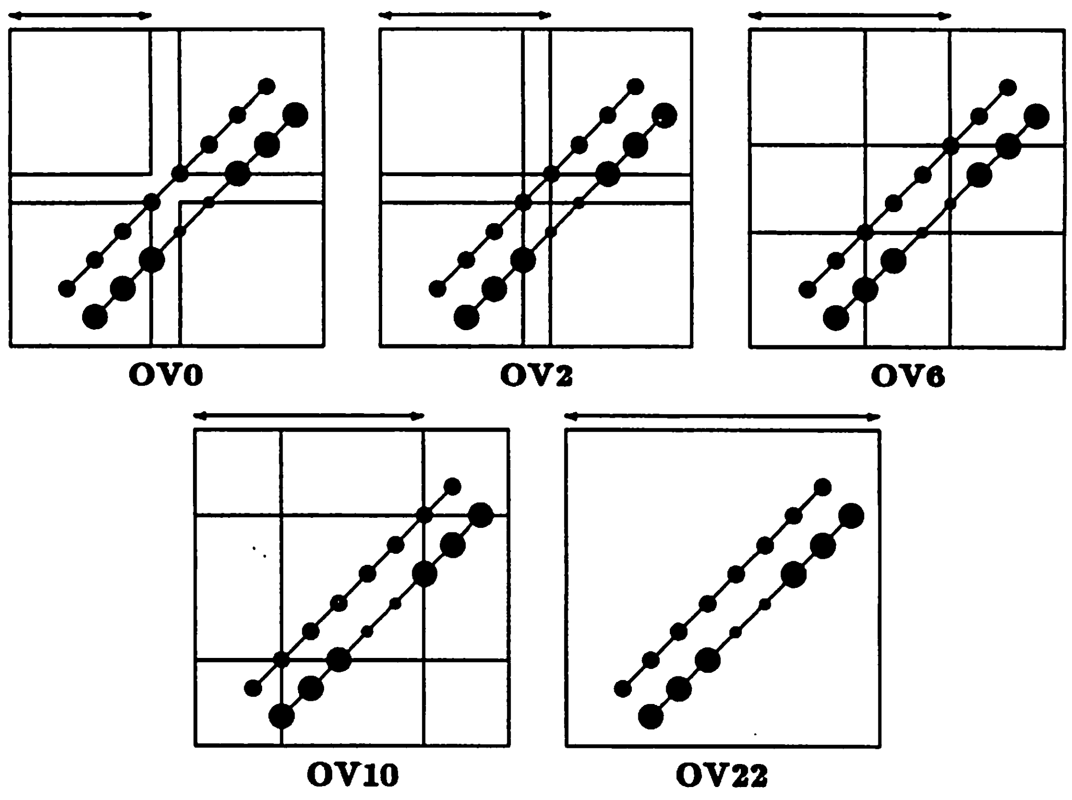


Figure 9: Overlapping Configurations

requirements of the nodes. Without the planner, the computational cost is the network runtime, while with the planner, we increase the runtime by fifty-percent—sixty-percent if meta-level communication is included as well. These increments are based on an estimate of the additional computation costs reflected by the DVMT. The communication requirements are indicated by the number of hypotheses and meta-level messages (where appropriate) exchanged, which are summed to indicate the total amount of communication resource usage. Finally, statistics were accumulated to measure how many hypotheses were abstracted in the meta-level messages exchanged.

The experimental results are summarized in Table 7. In the environments with little or no overlap, the planner did not significantly alter the computation costs—although the nodes with the planner ran fewer tasks, their increased overhead for planning tended to balance this out. However, as overlap became much larger, having a planner became more advantageous. We can conclude that, as a node becomes more uncertain about where to apply its computation resources, the node's *computational overhead* for planning (and meta-level communication) becomes increasingly acceptable. It pays to have a planner in complex problem solving situations, particularly when planning is relatively cheap compared to problem solving actions.

Deciding when the extra *communication overhead* of meta-level communication is acceptable is more difficult because meta-level communication always increases the commu-

Experiment	Environment	Planner	MLC	Time	Comp.	Hyps	MLM	Comm.	Equiv
E7.1	OV0	no	none	26	26	8	0	8	8
E7.2	OV0	yes	none	15	23	10	0	10	10
E7.3	OV0	yes	part	15	24	10	14	24	28
E7.4	OV0	yes	full	15	24	10	56	66	133
E7.5	OV2	no	none	33	33	18	0	18	18
E7.6	OV2	yes	none	26	39	25	0	25	25
E7.7	OV2	yes	part	15	24	24	35	59	101
E7.8	OV2	yes	full	15	24	24	118	142	271
E7.9	OV6	no	none	51	51	27	0	27	27
E7.10	OV6	yes	none	25	38	30	0	30	30
E7.11	OV6	yes	part	18	29	40	61	101	177
E7.12	OV6	yes	full	19	30	48	169	217	421
E7.13	OV10	no	none	116	116	51	0	51	51
E7.14	OV10	yes	none	34	51	40	0	40	40
E7.15	OV10	yes	part	19	30	42	78	120	266
E7.16	OV10	yes	full	18	29	42	180	222	436
E7.17	OV22	no	none	212	212	336	0	336	336
E7.18	OV22	yes	none	42	63	48	0	48	48
E7.19	OV22	yes	part	25	40	57	135	192	396
E7.20	OV22	yes	full	20	32	59	209	268	463

Legend

Environment:	Overlap environment tested
Planner:	Use of the planner
MLC:	Nodes exchange meta-level information
Time:	Earliest time at which a solution was found
Comp.:	Computational resources used for problem solving and planning
Hyps:	Number of hypotheses exchanged
MLM:	Number of meta-level messages
Comm.:	Communication resources used
Equiv:	Number of hypotheses which would need to be exchanged to equal amount of hypothesis and meta-level information

Table 7: Experiments with Overlapping Configurations

nication resource usage. If bandwidth is extremely limited, then meta-level communication may be a luxury that cannot be afforded: even though it can increase network performance significantly in highly overlapping environments, performance without it might be satisfactory. Also, in environments with little or no overlap, meta-level communication has less effect, and might be done without whether or not bandwidth was tight. However, if the communication channels can handle the increased usage, meta-level communication can nearly halve the time necessary to generate the solution in highly overlapping environments (compare E7.14 to E7.16, E7.18 to E7.20). In applications where network performance is at a premium, the increased overhead of meta-level communication might be acceptable.

Although these environments did not emphasize its positive aspects, overlap is extremely desirable. It insures reliable performance in distributed problem solving networks by allowing the network to derive the solution despite errorful data and node failures. Based on the empirical evidence, we conclude that our mechanisms are particularly efficacious in highly overlapping, hence reliable, distributed problem solving networks.

The experiments also indicate that, with no overlap, any meta-level exchange is unnecessary; with moderate overlap, it seems sufficient to exchange enough information to avoid immediate redundancy; and with a high degree of overlap, using meta-level information to avoid future redundancy becomes important to network performance. By exchanging meta-level information, nodes use about half as much communication resources than if they had exchanged enough hypotheses to convey the same information. Furthermore, the computational requirements would be significantly increased if it were hypotheses that were exchanged—by exchanging plans, nodes tell each other what they are going to be doing; by exchanging hypotheses, nodes would have to duplicate each others planning-computations to generate the same predictive information. In addition, nodes can reduce the amount of meta-level communication by making intelligent predictions about the future activities of other nodes based on information they already have, instead of exchanging more information. We hope to provide nodes with this capability; currently, our nodes are unable to make such predictions. We have run experiments where nodes receive obsolete meta-level information. If overlap is small, this obsolescence does not significantly affect the utility of the information because the nodes are unlikely to perform redundant activities anyway. As overlap becomes larger, however, the effects become more dramatic. When nodes completely overlap (OV22), the exchange of obsolete meta-level information caused the network to find the solution in 54 time units—the network would have performed better if there had been no meta-level communication at all (E7.18)!

Our research therefore indicates that, by organizing nodes, by providing them with planners, and by allowing them to exchange meta-level information, distributed problem solving networks can achieve enhanced performance. In fact, in the types of environments we have outlined and with the assumptions we have made, these mechanisms can often make network problem solving completely coherent. However, these mechanisms need more work before they can become applicable in a wider range of environments with different problem solving situations and network characteristics. In particular, our future work will be directed toward improving the ability of nodes to reason about and make predictions based on both locally-generated and received meta-level information.

8. Coherent Cooperation in Other Domains

As the number of tasks a node may perform increases, choosing among them requires increasingly sophisticated local control, particularly if the tasks are highly interdependent. However, local control is not the whole story; any complete distributed computing system requires network control as well. In this section, we briefly outline how network control can be combined with the local control mechanisms presented. In our distributed problem solving network, network control allows the nodes to organize (and reorganize) themselves if network responsibilities are inappropriately assigned and to exchange tasks if the processing load is poorly distributed. With network and local control united in a single framework, we then look beyond distributed problem solving to explore how our approach might be implemented in other distributed computing applications.

8.1 Network Control

In the preceding sections, we assumed that the organizational structure is established during network initialization and that the inherent distribution of data sufficiently balances the processing load among nodes. In situations where these assumptions are invalid, network control must reallocate organizational responsibilities and problem solving tasks to ensure both that each node has a set of important tasks to perform and that all important tasks are assigned to one or more nodes.

One form of network control is task passing. A bidding algorithm allows an overloaded node (a node with numerous highly rated tasks) to make informed decisions about where it can send some of these tasks [7,22]. Task passing using bidding protocols is an elegant mechanism for temporarily alleviating a poorly balanced situation. Unfortunately, its efficacy is highly dependent on the delay, bandwidth, and error rate of the communication channels:

- Bidding requires a sequence of message exchanges before a task is passed, and messages are subject to delays. If delays are substantially longer than task execution times, an overloaded node may execute its tasks faster locally than if it had passed them to idle nodes and waited for the results.[10]
- A task message specifies not only actions to execute but also a context (hypotheses, goals) for their execution. Channel bandwidth may limit the number of nodes that can obtain a task's context and might thus influence where a task is relocated [2].
- Since bidding requires mutual agreement among nodes, it may be inappropriate for environments with errorful channels [12].

Task passing overhead and delays may be reduced if nodes have sufficient network awareness to send tasks directly to one or more likely nodes using focused addressing

[10]In our larger environments, for example, a node may perform two local KSIs in the time it takes a message to be transmitted, so passing single KSIs would most likely be ineffective. Passing plans which represent extended sequences of KSIs might prove more useful, but by accepting longer tasks a node may significantly reduce its own responsiveness to local processing responsibilities.

techniques [26]. Organizational knowledge and meta-level information can improve focused addressing decisions and lets nodes monitor network activity to detect whether a passed task is actually being performed. However, even with these improvements, excessive amounts of task passing can burden communication resources and degrade network performance; task passing is generally effective only for remedying temporary, minor load imbalances.

The other form of network control makes long-term changes to node activities by modifying the organizational structure. Rather than passing an idle node a series of (possibly unrelated) tasks, it may be advantageous to change the role the node plays in the network. For example, the node may become responsible for integrating hypotheses from a number of other nodes. This change to the organization provides the node with a role that will keep it busy doing important tasks for an extended period of time—tasks that are related and likely to share context. Network control must also modify the organization to reallocate network responsibilities when nodes fail.

The complex problem of network reorganization is the topic of a parallel research effort. We have been developing a knowledge-based fault-diagnosis component to detect and locate inappropriate system behavior [14] and an organizational self-design component [5,21]. The planner and meta-level communication, which increase a node's self- and network-awareness, are expected to prove useful in this research by helping to both detect when an organization is inappropriate and to determine how to improve it. We speculate that network reorganization will require some form of negotiation among nodes [7]. However, in domains where communication is slow and uncertain, fully negotiated changes may be impractical, so that nodes may have different views of the organisational structure. Although this should be minimized to improve coherence, the FA/C approach allows network problem solving despite such inconsistencies.

In summary, network control can take the form of short-term task passing decisions or long-term organizational structuring decisions. Task passing is useful for rectifying temporary, specific imbalances while organization modification reallocates network responsibility to better balance the load over the long run. Network control must be integrated with local control: local control decisions affect network activity, and network control decisions influence what tasks a node has and how it ranks them. Coherent cooperation requires that both types of control be sophisticated to insure that node activities and interactions contribute to overall network performance.

8.2 Other Domains

Making intelligent local and network control decisions is the key to achieving coherent cooperation. In this article, we have detailed mechanisms to improve local control decisions in the DVMT, and have considered approaches for network control. We believe, however, that our ideas are applicable in many other distributed computing systems.

Improving control decisions requires knowledge. Local control must know how a particular decision it makes will affect future activity both locally and at other nodes, and how decisions at other nodes may affect it; network control must know about the current and expected future activities of the nodes and how decisions to move tasks or responsibilities in the network will affect network performance. If tasks are nontrivial and interdependent, all this knowledge is difficult and time consuming to generate and, since it continuously changes, expensive to exchange. Our approach accepts that control decisions in realistic systems will not be made with complete knowledge. Instead, it concentrates on generating and exchanging high-level views of node and network activity that convey important knowledge without details.

The knowledge needed to make informed control decisions depends on the tasks and interactions in the network. In the DVMT, the important knowledge is about a node's search activities: organizational structuring constrains the possibilities, the planner predicts more accurately where a node will search in the near future, and meta-level communication exchanges these predictions. Details about specific KSs to run and goals to satisfy would not significantly improve control decisions. If a distributed system must provide real-time response, details about what results a task will generate may be less important than task deadlines and estimated runtimes. In distributed systems with severe resource limitations, the expected resource needs of tasks is the important knowledge, while in systems where tasks on different nodes must sometimes synchronize, knowledge about the synchronization is crucial for smooth cooperation. In short, tasks may interact in any number of ways (cooperate to solve a problem, compete for limited resources, synchronize); the mechanisms that make control decisions and the knowledge used must be geared to the form that interaction takes in a particular application.

Once task interactions for an application are characterized, mechanisms like those in the DVMT can increase the coherence of cooperation. An organizational structure can broadly define network responsibilities—types of tasks a node may perform, resources assigned to a node, other nodes with which a node may need to synchronize. By planning task sequences, a node can make predictions about results it may generate, about when it will be idle, about its expected resource requirements, or about exactly when it will be executing specific tasks. Meta-level communication enables these predictions to be exchanged, and cooperation can be more coherent as nodes work to generate compatible results, maximize the number of tasks which meet their deadlines, avoid resource conflicts, or synchronize task executions.

We therefore believe that the approach outlined in this article can be extended to many other distributed computing applications where tasks interact. Indeed, we have used the same approach in our distributed version of the DVMT [8]. To enable processes on separate machines to cooperate coherently, we provide each with an organizational structure that allocates tasks (in this case, simulated nodes) among them and establishes communication links. Each machine may have several nodes to simulate and local control interleaves node activities. Because nodes can exchange information, their activities must be sufficiently synchronized to insure deterministic simulation results. Local control decisions about when to allow a simulated node to continue its activities are therefore guided, in part, by meta-level state information from other machines. Hence, even in this relatively simple distributed application with interacting tasks, the approach we have outlined proves useful.

9. Summary and Perspective

Coherent cooperation is the holy grail of distributed problem solving network research. We have described an approach for achieving coherent cooperation that is responsive to changes in network activity as problem solving progresses. Coherence is accomplished without excessive communication because this approach combines the use of a static organizational structure with mechanisms that enable nodes to dynamically generate and exchange minor refinements to this structure. This approach thus relies on the ability of network problem solving nodes to integrate incomplete and inconsistent information about the state of network problem solving, the availability of network resources (processors, communication paths, sensors, and effectors), the organizational roles and responsibilities of each node, and the potential future activities that each node can perform.

This view of distributed problem solving stresses the importance of sophisticated local control that integrates knowledge of the problem domain with (meta-level) knowledge about network coordination. Such control allows nodes to make rapid, intelligent local decisions based on changing problem characteristics without the overhead of conferring with each other to coordinate these decisions. Instead, coordination is based on an organizational view of individual node activity, so nodes need not have detailed models of the problem solving activity of their compatriots. Dynamic improvements to this organizational view may be achieved with the exchange of meta-level messages which briefly convey high-level coordination information. In short, the nodes initiate their own activities and take advantage of all available local and network knowledge to form the best "team" possible within the constraints of their environment.

This approach to distributed problem solving is based on the characteristics and performance criteria of a particular class of distributed applications. In contrast with others [16], we assume that the network contains only a limited number (tens to hundreds) of highly sophisticated, and loosely-coupled, problem solving nodes rather than thousands of relatively simple processing elements. Therefore, we must coordinate our limited number of nodes to make the most effective team possible. Since we assume that communication between agents is potentially slow and unreliable, we regard coordination that requires mutual agreement on contracts before action [1,7] to be insufficiently responsive to changing problem circumstances (indeed, mutual agreement might not even be possible [12]). To insure reliability, we cannot accept centralized coordination [1]. The unpredictable nature of the problem solving environment makes simple game theoretic models of agents unrealizable [24], while more complete models of agent beliefs [15] might require nodes to essentially duplicate each others reasoning.

In this article we have focused on one particular aspect of our overall network coordination framework: modifications to the blackboard architecture of the individual nodes that enhance their ability to make predictions about their future activities. Network coherence is increased by allowing a node to refine its organizational role based on these predictions. Exchanging the predictions permits a node to refine its view of the organizational roles of the other nodes. We empirically established that each of these improvements enhanced the ability of nodes to cooperate coherently in a variety of problem solving situations. Furthermore, we studied the costs and benefits of using these improvements, and established that they were particularly cost-effective in complex problem solving situations. Finally, we outlined how our approach for improving local control decisions can be combined with network control, and discussed how our approach might be used in other distributed computing applications.

Our future plans to improve the organizational refinement process include augmenting the abstracted blackboard to more appropriately represent both the current state of a node (based on its data) and its potential future states (based on its goals), enhancing the mechanisms to recognize problem solving situations, extending the plan structures to incorporate more information, and improving the abilities of nodes to make predictions about other nodes' activities based on received meta-level information. We also plan to implement and evaluate network control mechanisms, and we intend to more deeply study the complex issue of achieving coherent cooperation in scenarios involving real-time constraints, where the dynamic allocation of network responsibilities might include the exchange of tasks between nodes. Our preliminary experiments indicate that these developments should significantly improve the performance of distributed problem solving networks.

Acknowledgments

We would like to thank Krithi Ramamritham, Jack Stankovic, and Lee Erman for their comments on earlier drafts of this paper.

REFERENCES

[1] Stephanie Cammarata, David McArthur, and Randall Steeb. "Strategies of cooperation in distributed problem solving." *Proceedings of the Eighth International Conference on Artificial Intelligence*, pages 767–770, August 1983.

[2] L. Casey and N. Shelness. "A Domain Structure for Distributed Computer Systems." *Proceedings of the Sixth ACM Symposium on Operating System Principles*, pages 101–108, November 1977.

[3] Daniel D. Corkill and Victor R. Lesser. "A goal-directed Hearsay-II architecture: Unifying data and goal directed control." Technical Report 81-15, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, June 1981.

[4] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlicka. "Unifying data-directed and goal-directed control: An example and experiments." *Proceedings of the Second National Conference on Artificial Intelligence*, pages 143–147, August 1982.

[5] Daniel David Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks.* Ph.D. Dissertation, University of Massachusetts, February 1983. (Available as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1982.)

[6] Daniel D. Corkill and Victor R. Lesser. "The use of meta-level control for coordination in a distributed problem solving network." *Proceedings of the Eighth International Conference on Artificial Intelligence*, pages 748–756, August 1983.

[7] Randall Davis and Reid G. Smith. "Negotiation as a metaphor for distributed problem solving." *Artificial Intelligence*, 20(1983):63—109.

[8] Edmund H. Durfee, Daniel D. Corkill, and Victor R. Lesser. "Distributing a Distributed Problem Solving Network Simulator." *Proceedings of the Fifth Real-time Systems Symposium*, pages 237—246, December 1984.

[9] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. "Increasing coherence in distributed problem solving networks." *Proceedings of the Ninth International Conference on Artificial Intelligence*, pages 1025—1030, August 1985.

[10] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, D. Raj Reddy. "The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty." *Computing Surveys*, 12(2):213—253, June 1980.

[11] Michael Fehling and Lee Erman. "Report on the Third Annual Workshop on Distributed Artificial Intelligence." *SIGART Newsletter*, 84:3–12, April 1983.

[12] Joseph Y. Halpern and Yoram Moses. "Knowledge and common knowledge in a distributed environment." *Proceedings of the Third ACM Conference on Principles of Distributed Computing*

[13] Frederick Hayes-Roth and Victor R. Lesser. "Focus of attention in the Hearsay-II speech understanding system." *Proceedings of the Fifth International Conference on Artificial Intelligence*, pages 27–35, August 1977.

[14] Eva Hudlicka and Victor Lesser. "Design of a knowledge-based fault detection and diagnosis system." *Proceedings of the Seventeenth Hawaii International Conference on System Sciences*, pages 224-230, January 1984.

[15] Kurt Konolige. "A deductive model of belief." *Proceedings of the Eighth International Conference on Artificial Intelligence*, pages 377-381, August 1983.

[16] William A. Kornfeld and Carl E. Hewitt. "The scientific community metaphor." *IEEE Transactions on Man, Systems, and Cybernetics*, SMC-11(1):24-33, January 1981.

[17] Victor R. Lesser and Lee D. Erman. "An experiment in distributed interpretation." *IEEE Transactions on Computers*, C-29(12):1144-1163, December 1980.

[18] Victor R. Lesser and Daniel D. Corkill. "Functionally accurate, cooperative distributed systems." *IEEE Transactions on Man, Systems, and Cybernetics*, SMC-11(1):81-96, January 1981.

[19] Victor Lesser, Daniel Corkill, Jasmina Pavlin, Larry Lefkowitz, Eva Hudlicka, Richard Brooks, and Scott Reed. "A high-level simulation testbed for cooperative distributed problem solving." *Proceedings of the Third International Conference on Distributed Computer Systems*, pages 341—349, October 1982.

[20] Victor R. Lesser and Daniel D. Corkill. "The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks." *AI Magazine*, 4(3):15-33, Fall 1983.

[21] H. Edward Pattison, Daniel D. Corkill, and Victor R. Lesser. "Instantiating descriptions of organizational structures." Technical Report, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, in preparation.

[22] Krithivasan Ramamritham and John A. Stankovic. "Dynamic task scheduling in hard real-time distributed systems." *IEEE Software*, pages 65—75, July 1984.

[23] 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 01003, September 1981.

[24] Jeffrey S. Rosenschein and Michael R. Genesereth. "Deals among rational agents." *Stanford Heuristic Programming Project Report No. HPP-84-44*, December 1984.

[25] 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.

[26] John A. Stankovic, Krithi Ramamritham and Sheng Chang Cheng. "Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems." To appear in *IEEE Transactions on Computers*, December 1985.

[27] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley 1977.