Frameworks for Cooperation in Distributed Problem Solving

REID G. SMITH, MEMBER, IEEE, AND RANDALL DAVIS

Abstract — **Two forms of cooperation in distributed problem solving are considered: *task-sharing* and *result-sharing.* In the former, nodes assist each other by sharing the computational load for the execution of subtasks of the overall problem. In the latter, nodes assist each other by sharing partial results which are based on somewhat different perspectives on the overall problem. Different perspectives arise because the nodes use different knowledge sources (KS's) (e.g., syntax versus acoustics in the case of a speech-understanding system) or different data (e.g., data that is sensed at different locations in the case of a distributed sensing system). Particular attention is given to control and to internode communication for the two forms of cooperation. For each, the basic methodology is presented and systems in which it has been used are described. The two forms are then compared and the types of applications for which they are suitable are considered.**

I. DISTRIBUTED PROBLEM SOLVING

DISTRIBUTED problem solving is the cooperative solution of problems by a decentralized and loosely coupled collection of knowledge sources (KS's) (procedures, sets of rules, etc.), located in a number of distinct processor nodes. The KS's *cooperate* in the sense that no one of them has sufficient information to solve the entire problem; mutual sharing of information is necessary to allow the group as a whole to produce an answer. By *decentralized* we mean that both control and data are logically and often geographically distributed; there is neither global control nor global data storage. Loosely coupled means that individual KS's spend the great percentage of their time in computation rather than communication.

Distributed problem solvers offer advantages of speed, reliability, extensibility, the ability to handle applications with a natural spatial distribution, and the ability to tolerate uncertain data and knowledge. Because such systems are highly modular they also offer conceptual clarity and simplicity of design.

Although much work has been done in distributed processing, most of the applications have not addressed issues that are important for the design of artificial intelligence (AI) problem solvers. For example, the bulk of the processing is usually done at a central site with remote processors limited to basic data collection (e.g., credit card verification). While it is common to distribute data and processing, it is not common to distribute control, and the processors do not cooperate in a substantive manner.

Researchers in the area of distributed processing have not taken problem solving as their primary focus. It has generally been assumed, for example, that a well-defined and *a priori* partitioned problem exists and that the major concerns lie in an optimal static distribution of tasks, methods for interconnecting processor nodes, resource allocation, and prevention of deadlock. Complete knowledge of timing and precedence relations between tasks has generally been assumed, and the major reason for distribution has been taken to be load balancing (see for example [1], [3]). Distributed problem solving, on the other hand, includes as part of its basic task the partitioning of a problem.

Perhaps the most important distinction between distributed problem solving and distributed processing systems can be found by examining the origin of the systems and the motivations for interconnecting machines. Distributed processing systems often have their origin in an attempt to synthesize a network of machines capable of carrying out a number of widely disparate tasks. Typically, several distinct applications are envisioned, with each application concentrated at a single node (as for example in a three-node system intended to do payroll, order entry, and process control). The aim is to find a way to reconcile any conflicts and disadvantages arising from the desire to carry out disparate tasks, in order to gain the benefits of using multiple machines (sharing of data bases, graceful degradation, etc.). Unfortunately, the conflicts that arise are often not simply technical (e.g., word sizes and data base formats) but include sociological and political problems as well [6]. The attempt to synthesize a number of disparate tasks leads to a concern with issues such as access control and protection, and results in viewing cooperation as a form of *compromise* between potentially conflicting perspectives and desires at the level of system design and configuration.

In distributed problem solving, on the other hand, a single task is envisioned for the system, and the resources to be applied have no other predefined roles to carry out. A system is constructed *de novo*, and as a result the hardware and software can be chosen with one aim in

Manuscript received January 28, 1980, revised September 1, 1980. This work was supported by the Department of National Defence of Canada, Research and Development Branch, and by the Advanced Research Projects Agency of the United States Department of Defense under Office of Naval Research Contract N00014-75-C-0643.

R. G. Smith is with the Defence Research Establishment Atlantic, Dartmouth, NS, Canada, B2Y 3Z7.

R. Davis is with the Artificial Intelligence Laboratory, Massachusetts Institute Of Technology, Cambridge, MA 02139.

mind: the selection that leads to the most effective environment for cooperative behavior. This also means that cooperation is viewed in terms of benevolent problem-solving behavior; that is, how can systems that are perfectly willing to accommodate one another act so as to be an effective team? Our concerns are thus with developing frameworks for *cooperative behavior between willing entities,* rather than frameworks for enforcing cooperation as a form of compromise between potentially incompatible entities.

This leads us to investigate the structure of interactions between cooperating nodes. We are primarily concerned with the *content* of the information to be communicated between nodes and the *use* of the information by a node for cooperative problem solving. We are less concerned with the specific *form* in which the communication is effected.

In this paper two forms of cooperation in distributed problem solving are considered: *task-sharing* and *result-sharing.* In the former, nodes assist each other by sharing the computational load for the execution of subtasks of the overall problem. In the latter, nodes assist each other by sharing partial results which are based on somewhat different perspectives on the overall problem. Different perspectives arise because the nodes use different KS's (e.g., syntax versus acoustics in the case of a speech-understanding system) or different data (e.g., data that is sensed at different locations in the case of a distributed sensing system).

For each form, the basic methodology is presented, and systems in which it has been used are described. The utility of the two forms is examined, and their complementary nature is discussed.

The physical architecture of the problem solver is not of primary interest here. It is assumed to be a network of loosely coupled, asynchronous nodes. Each node contains a number of distinct KS's. The nodes are interconnected so that each node can communicate with every other node by sending messages. No memory is shared by the nodes.

II. Cooperating Experts

A familiar metaphor for a problem solver operating in a distributed processor is a group of human experts experienced at working together, trying to complete a large task. This metaphor has been used in several AI systems [10]-[12], [18]. Of primary interest to us in examining the operation of a group of human experts is the way in which they interact to solve the overall problem, the manner in which the workload is distributed among them, and how results are integrated for communication outside the group.

It is assumed that no one expert is in total control of the others, although one expert may be ultimately responsible for communicating the solution of the top-level problem to the customer outside the group. In such a situation each expert may spend most of his time working alone on various subtasks that have been partitioned from the main task, pausing occasionally to interact with other members of the group. These interactions generally involve requests for assistance on subtasks or the exchange of results.

Individual experts can assist each other in at least two ways. First, they can divide the workload among themselves, and each node can independently solve some subproblems of the overall problem. We call this *task-sharing* (as in [11] and [18]). In this mode of cooperation, we are primarily concerned with the way in which experts decide who will perform which task. We postulate that one interesting method of effecting this agreement is via negotiation.

An expert (El) may request assistance because he encounters a task too large to handle alone, or a task for which he has no expertise. If the task is too large, he will first partition it into manageable subtasks, and then attempt to find other experts who have the appropriate skills to handle the new tasks. If the original task is beyond his expertise, he immediately attempts to find another more appropriate expert to handle it.

In either case, if E1 knows which other experts have the necessary expertise, he can notify them directly. If he does not know anyone in particular who may be able to assist him (or if the task requires no special expertise), then he can simply describe the task to the entire group.

If another expert (E2) believes he is capable of carrying out the task that E1 described, he informs E1 of his availability and perhaps indicates any especially relevant skills he may have. E1 may discover several such volunteers and can choose from among them. The chosen volunteer then requests additional details from El, and the two engage in further direct communication for the duration of the task.

Those with tasks to be executed and those capable of executing the tasks thus engage each other in a simple form of negotiation to distribute the workload. They form subgroups dynamically as they progress towards a solution.[1]

When subproblems cannot be solved by independent experts working alone, a second form of cooperation is appropriate. In this form, the experts periodically report to each other the partial results they have obtained during execution of individual tasks. We call this *result-sharing* (as, for example, in [12] and [13]). It is assumed in this mode of cooperation that problem partitioning has been effected *a priori* and that individual experts work on subproblems that have some degree of commonality (e.g., interpreting data from overlapping portions of an image).
An expert (El) reports a partial result for his subproblem to his neighbors (E2 and E3) when that result may have some bearing on the processing being done by them. (For example, a partial result may be the best result that E1 can derive using only the data and knowledge available to him.) E2 and E3 attempt l) to use El's result to confirm or deny competing results for their subproblems, or 2) to

[1]Subgroups offer two advantages. First, communication among the members does not needlessly distract the entire group. This is important because communication itself can be a major source of distraction and difficulty in large groups (see, for example, [91). Thus one of the major purposes of organization is to reduce the amount of communication that is needed. Second, the subgroup members may be able to communicate with each other in a language that is more efficient for their purpose than the language in use by the entire group.

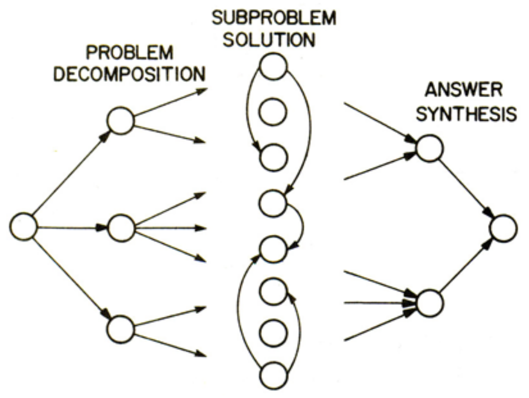


Fig. 1. Phases of distributed problem solving.

aggregate partial results of their own with El's result to produce a result that is relevant to El's subproblem as well as their own, or 3) to use El's result to indicate alternative lines of attack that they might take to solve their own subproblems.

III. A Perspective On Distributed Problem Solving

In this section we present a model for the phases that a distributed problem solver passes through as it solves a problem (Fig. 1). The model offers a framework in which to anchor the two forms of cooperation that are the primary focus of this paper. It enables us to see the utility of the two forms, the types of problems for which they are best suited, and the way in which they are complementary.[2]

In the first phase, the problem is decomposed into subproblems. As Fig. 1 shows, the decomposition process may involve a hierarchy of partitionings. In addition, the process may itself be distributed in order to avoid bottlenecks. Decomposition proceeds until kernel (nondecomposable) subproblems are generated. Consider as an example a simple distributed sensing system (DSS). In the problem decomposition phase, the subproblems of detecting objects in specific portions of the overall area of interest are defined and distributed among the available sensors.

The second phase involves solution of the kernel subproblems. As shown in the figure, this may necessitate communication and cooperation among the nodes attempting to solve the individual subproblems. In the DSS example, communication is required in the subproblem solution phase 1) if objects can move from one area to another so that it is helpful for sensors to inform their neighbors of the movement of objects they have detected, or 2) if it is difficult for a single sensor to reliably detect objects without assistance from other sensors.

Answer synthesis is performed in the third phase; that is, integration of subproblem results to achieve a solution to the overall problem. Like problem decomposition, answer synthesis may be hierarchical and distributed. In the DSS example, the answer synthesis phase involves generation of a map of the objects in the overall area of interest.

For any given problem, the three phases may vary in complexity and importance. Some phases may either be missing or trivial. For example, in the traffic-light control problem considered in [13], the problem decomposition phase involves no computation. Traffic-light controllers are simply placed at each intersection. For a DSS, the problem decomposition is suggested directly by the spatial distribution of the problem.[3]

There is also no answer synthesis phase for traffic-light control. The solution to a kernel subproblem is a smooth flow of traffic through the associated intersection. There is no need to synthesize an overall map of the traffic. Thus the solution to the overall problem *is* the solution to the kernel subproblems. (This is generally true of control problems; note that it does not mean, however, that communication among the nodes solving individual subproblems is not required.)

Many search problems (like symbolic integration [16]) also involve a minimal answer synthesis phase. Once the problem has been decomposed into kernel subproblems and they have been solved, the only answer synthesis required is recapitulation of the list of steps that have been followed to obtain the solution. However, for some problems the answer synthesis phase is the dominant phase. An example is the CONGEN program [4]. CONGEN is used in molecular structure elucidation. It generates all structural isomers that are both consistent with a given chemical formula and that include structural fragments known to be present in the substance (superatoms). In the problem decomposition phase, CONGEN generates all structures that are consistent with the data (by first generating intermediate structures, then decomposing those structures, and so on until only structures that contain atoms or superatoms remain). At this point, the superatoms (like the atoms) are considered by name and valence only. In the answer synthesis phase, the superatoms are replaced by the actual structural fragments they represent and are embedded in the generated structures. Because embedding can often be done in many ways, a sizable portion of the overall computation is accounted for by this phase.

IV. Caveats For Cooperation

One of the main aims in adopting a distributed approach is to achieve high-speed problem solving. In order to do this, situations in which processors "get in each other's way" must be avoided. This obviously depends on the problem itself (e.g., there are problems for which data or computation cannot be partitioned into enough mostly independent pieces to occupy all of the processors). Performance also depends, however, on the *problem-solving architecture*. It is therefore appropriate to consider frameworks for cooperation.

[2]It will be apparent that the model is also applicable to centralized problem solving. The distinct phases, however, are more obvious in a distributed problem solver, primarily because communication and cooperation must be dealt with explicitly in this case.

[3]Note that the problem solver must still implement even an obvious decomposition. Nodes must still come to an agreement as to which node is to handle which portion of the overall area.

It is common in AI problem solvers to partition expertise into domain-specific KS's, each of which is expert in a particular part of the overall problem. KS's are typically formed empirically, based on examination of different types of knowledge that can be brought to bear on a particular problem. In a speech-understanding problem, for example, knowledge is available from the speech signal itself, from the syntax of the utterances, and from the semantics of the task domain [7]. The decisions about which KS's are to be formed is often made in concert with the formation of a hierarchy of levels of data abstraction for a problem. For example, the levels used in the hierarchy of the HEARSAY-II speech-understanding system were parametric, segmental, phonetic, surface-phonemic, syllabic, lexical, phrasal, and conceptual [7]. KS's are typically chosen to handle data at one level of abstraction or to bridge two levels (see, for example, [7] and [15]).

Interactions among the KS's in a distributed processor are more expensive than in a uniprocessor because communication in a distributed architecture is generally much slower than computation. The framework for cooperation must therefore minimize communication among processors. Otherwise, the available communication channels may be saturated so that nodes are forced to remain idle while messages are transmitted.[4]

As a simple example of the difficulty that excessive communication can cause, consider a distributed processor with 100 nodes that are interconnected with a single broadcast communication channel. Assume that each of the nodes operates at 10^8 instructions per second; the computation and communication load is shared equally by all nodes, and the problem-solving architecture is such that one bit must be communicated by each node for every ten instructions that it executes. With these parameters it is readily shown that the communications channel must have a bandwidth of at least 1 Gbit/s (even ignoring the effect of contention for the channel) [18]. With a smaller bandwidth, processors are forced to stand idle waiting for messages.

There are, of course, many architectures that do not lead to channel bandwidths of the same magnitude. However, the point remains that special attention must be paid to internode communication and control in distributed problem solving if large numbers of fast processors are to be connected.

The framework for cooperation must also distribute the. processing load among the nodes in order to avoid computation and communication bottlenecks. Otherwise, overall performance may be limited by concentration of disproportionate amounts of computation or communication at a small number of processors. It is also the case that the control of processing must itself be distributed. Otherwise, requests for decisions about what to do next could in time accumulate at a "controller" node faster than they could be processed.[5] Distribution of control does, however, lead to difficulties in achieving globally coherent behavior since control decisions are made by individual nodes without the benefit of an overall view of the problem. We will illustrate this problem in Section VII.

[4]The focus here is on speed but the other reasons for adopting a distributed approach are also relevant — for example, reliability (i.e., the capability to recover from the failure of individual components, with graceful degradation in performance) and extensibility (i.e., the capability to alter the number of processors applied to a problem).

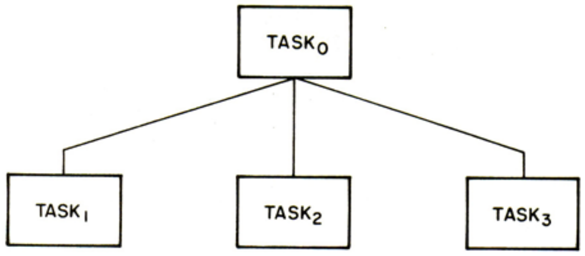


Fig. 2. Task-sharing.

V. Task-Sharing

Task-sharing is a form of cooperation in which individual nodes assist each other by sharing the computational load for the execution of subtasks of the overall problem. Control in systems that use task-sharing is typically goal-directed; that is, the processing done by individual nodes is directed to achieve subgoals whose results can be integrated to solve the overall problem.

Task-sharing is shown schematically in Fig. 2. The individual nodes are represented by the tasks in whose execution they are engaged.

The key issue to be resolved in task-sharing is how tasks are to be distributed among the processor nodes. There must be a means whereby nodes with tasks to be executed can find the most appropriate idle nodes to execute those tasks. We call this the *connection problem*. Solving the connection problem is crucial to maintaining the focus of the problem solver. This is especially true in AI applications because they do not generally have well-defined algorithms for their solution. The most appropriate KS to invoke for the execution of any given task generally cannot be identified *a priori*, and there are usually far too many possibilities to try all of them.

In the remainder of this section, we consider negotiation as a mechanism that can be used to structure node interactions and solve the connection problem in task-shared systems. Negotiation is suggested by the observation that the connection problem can also be viewed from the perspective of an idle node. It must find another node with an appropriate task that is available for execution. In order to maximize system concurrency, both nodes with tasks to be executed and nodes ready to execute tasks can proceed simultaneously, engaging each other in a process that resembles contract negotiation to solve the connection problem.

In the contract net approach to negotiation [18], [19], a contract is an explicit agreement between a node that

[5]Such a node would also be a hazard to reliability since its failure would result in total failure of the system.

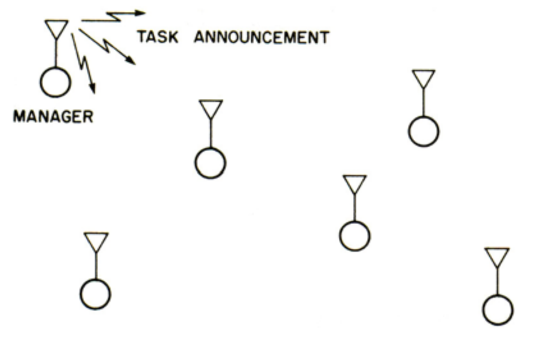


Fig. 3. Sending a task announcement.

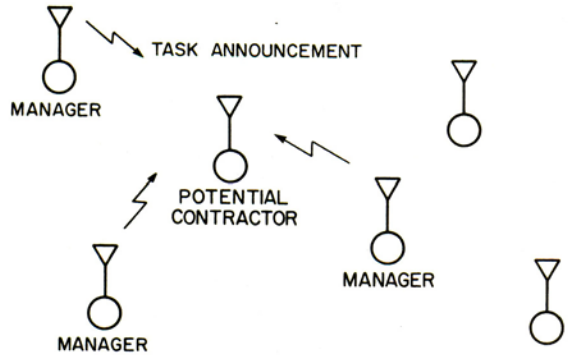


Fig. 4. Receiving task announcements.

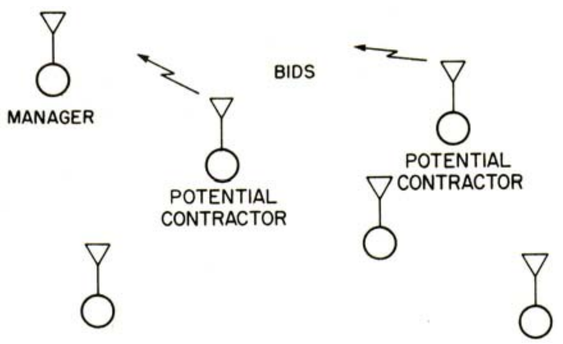


Fig. 5. Bidding.

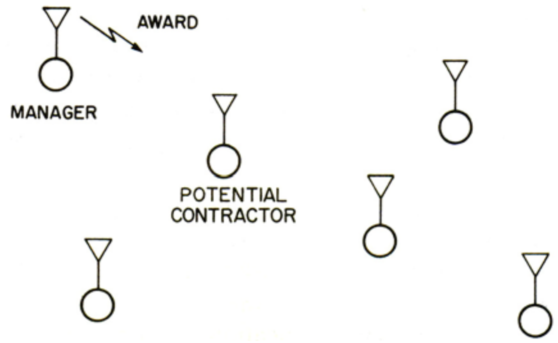


Fig. 6. Making an award.

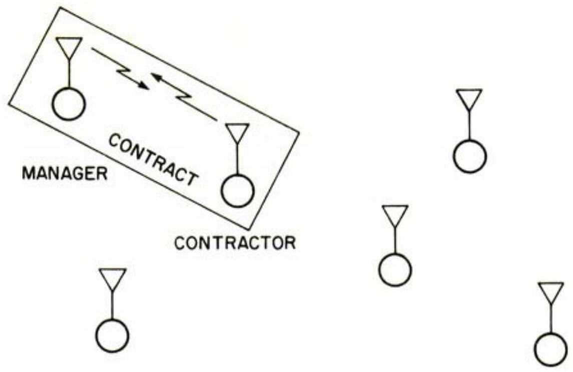


Fig. 7. Manager-contractor linkage.

generates a task (the manager) and a node willing to execute the task (the contractor). The manager is responsible for monitoring the execution of a task and processing the results of its execution. The contractor is responsible for the actual execution of the task. Individual nodes are not designated *a priori* as manager or contractor; these are only roles, and any node can take on either role dynamically during the course of problem solving. Nodes are therefore not statically tied to a control hierarchy.

A contract is established by a process of local mutual selection based on a two-way transfer of information. In brief, the manager for a task advertises the existence of the task to other nodes with a task announcement message (Fig. 3). Available nodes (potential contractors) evaluate task announcements made by several managers (Fig. 4) and submit bids on those for which they are suited (Fig. 5). An individual manager evaluates the bids and awards contracts for execution of the task to the nodes it determines to be most appropriate (Fig. 6). Manager and contractor are thus linked by a contract (Fig. 7) and communicate privately while the contract is being executed.

The negotiation process may then recur. A contractor may further partition a task and award contracts to other nodes. It is then the manager for those contracts. This leads to the hierarchical control structure that is typical of task-sharing. Control is distributed because processing and communication are not focused at particular nodes, but rather every node is capable of accepting and assigning tasks. This avoids bottlenecks that could degrade performance. It also enhances reliability and permits graceful degradation of performance in the case of individual node failures. There are no nodes whose failure can completely block the contract negotiation process.

We have only briefly sketched the negotiation process. Several complications arise in its implementation, and a number of extensions to the basic method exist that enable efficient handling of specialized interactions where the full complexity is not required (e.g., when simple requests for information are made). See [19] for a full treatment.

The following is an example of negotiation for a task that involves gathering of sensed data and extraction of signal features. It is taken from a simulation of a distributed sensing system (DSS) [17]. The sensing problem is partitioned into a number of tasks. We will consider one of these tasks, the *signal* task, that arises during the initialization phase of DSS operation.[6]

[6]The DSS in general is an example of a system that uses both task-sharing and result-sharing. Task-sharing is used to initialize the system (the problem decomposition phase of Fig. 1).

The managers for this task are nodes that do not have sensing capabilities but do have extensive processing capabilities. They attempt to find a set of sensor nodes to provide them with signal features. The sensor nodes, on the other hand, have limited processing capabilities and attempt to find managers that can further process the signal features they extract from the raw sensed data.

Recall that we view node interaction as an agreement between a node with a task to be performed and a node capable of performing that task. Sometimes the perspective on the ideal character of that agreement differs depending on the point of view of the participant. For example, from the perspective of the *signal* task managers, the best set of contractors has an adequate spatial distribution about the surrounding area and an adequate distribution of sensor types. From the point of view of the potential *signal* task contractors, on the other hand, the best managers are those closest to them, in order to minimize potential communication problems.

Each message type in the contract net protocol has slots for task-specific information. The slots have been chosen to capture the types of information that are usefully passed between nodes to determine appropriate connections without excessive communication. For example, *signal* task announcements include the following slots.

1) A *task abstraction* slot is filled with the task type and the position of the manager. This enables a potential contractor to determine the manager to which it should respond.

2) The *eligibility specification* slot contents indicate that bidders must have sensing capabilities and must be located in the same area as the manager. This reduces extraneous message traffic and bid processing by explicitly specifying the attributes of a contractor that are deemed essential by the manager.

3) The *bid specification* slot contents indicate that a bidder must specify its position and the name and type of each of its sensors. This reduces the length of bid messages by specifying the information that a manager needs to select a suitable set of contractors.

The potential contractors listen to the task announcements from the various managers. If eligible, they respond to the nearest manager with a bid that contains the information specified in the task announcement. The managers use this information to select a set of bidders and then award *signal* contracts. The award messages specify the sensors that a contractor must use to provide signal-feature data to its manager.

Use of the contract net protocol in a DSS makes it possible for the sensor system to be configured dynamically, taking into account such factors as the number of sensor and processor nodes available, their locations, and the ease with which communication can be established.

Negotiation offers a more powerful mechanism for *connection* than is available in current problem-solving systems. The connection that is effected with the contract net protocol is an extension to the *pattern-directed invocation* used in many AI programming languages (see [5] for an in-depth discussion). It is most useful when tasks require specialized KS's, when the appropriate KS's for a given task are not known *a priori* and when the tasks are large enough to justify a more substantial transfer of information before invocation than is generally allowed in problem solvers.

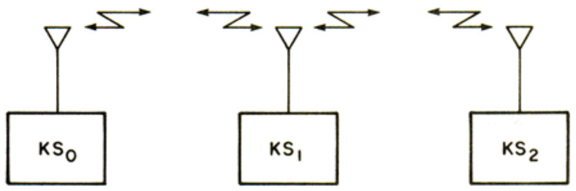


Fig. 8. Result-sharing.

VI. Result-Sharing

Result-sharing is a form of cooperation in which individual nodes assist each other by sharing partial results, based on somewhat different perspectives on the overall problem. In systems that use result-sharing, control is typically data-directed; that is, the computation done at any instant by an individual node depends on the data that it has available, either locally or from remote nodes. An explicit hierarchy of task–subtask relationships does not exist between individual nodes. Result-sharing is shown schematically in Fig. 8. The individual nodes are represented by KS's.

A simple example of the use of result-sharing is the development of consistent labelings for "blocks world" images [20]. A blocks world image is a line drawing that shows the edges of a collection of simple objects (e.g., cubes, wedges, and pyramids) in a scene. Each image is represented as a graph with nodes that correspond to the vertices of the objects in the image and arcs that correspond to the edges that connect the vertices. The goal is to establish a correspondence between nodes and arcs in the graph and actual objects.

A physically realizable vertex can be given a set of *a priori* possible labels based on the number of lines that meet at the vertex and the angles between the lines (e.g., "L," "T," "ARROW," and "FORK") [20]. A vertex is further specialized by the character of the lines that compose it (e.g., a line can define a convex boundary between surfaces of an object, a boundary between light and shadow, and so on). These labelings are established by examining the vertices in isolation. Ambiguity arises because generally more than one label is possible for each vertex. However, the number of possible labels can be reduced (often to a single label) by considering the constraints imposed by the interactions between vertices that share edges in an object. Very few of the large number of combinatorially possible vertex types can share an edge in a physically realizable object. Thus the key to achieving consistent image labeling is to compare the label set of each vertex with those of its neighbors and discard inconsistent labels.

If we partition the problem so that a processor node is responsible for one vertex in the image, then the basic

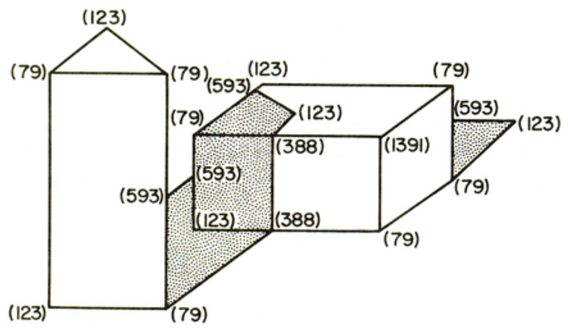


Fig. 9. Sample blocks world problem (from [21]).

result-sharing process is evident. Nodes communicate their local label sets to their neighbors. Each node uses these remote label sets together with consistency conditions to prune its own label set. It then transmits the new label set to its neighbors. The process continues until unique labels have been established for all nodes or no further pruning is possible. (This process of iterative refinement of label sets is called *relaxation, constraint propagation,* or *range restriction.)*[7]

Fig. 9 shows a simple image considered by Waltz. The numbers shown in parentheses beside each vertex indicate the number of *a priori* labelings possible for that vertex, in the absence of any intervertex constraints.[8]

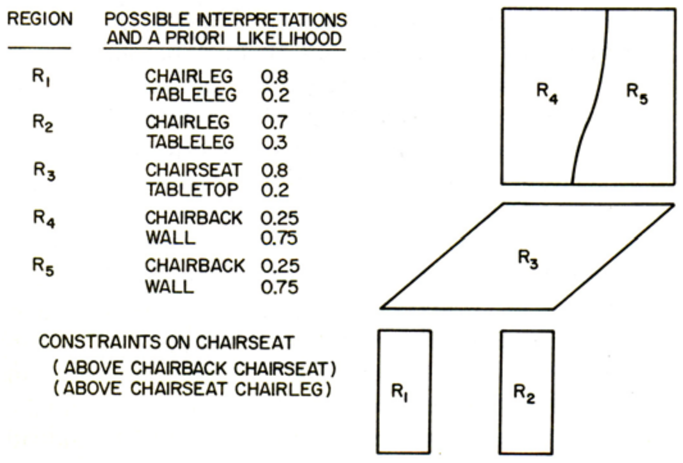
In addition to the ambiguity that arises in images from the blocks world, real images also suffer from ambiguity that arises as a result of noisy data and inaccurate feature detectors. The image is again considered to be a graph, but in this case the nodes correspond to small regions of the image [2], [22]. Examples of labels in this context are line segments with specified orientation, or objects (e.g., doors, chairs, and wastebaskets).

As with the blocks world problem, the aim is to establish unique labels for each node by considering contextual information from adjacent nodes. In this case, no absolute constraints are possible. Instead, the constraints (or *compatibilities* as they have also been called) express a degree of certainty that the labels associated with neighboring nodes are consistent (e.g., a line segment with a particular orientation detected at one node has a high degree of compatibility with another line segment with the same orientation detected at an adjacent node).

The method is initialized by associating a set of labels with each node on the basis of local feature detection. A numerical certainty measure is also assigned to each label. As before, nodes then communicate their local label sets to their neighbors. Instead of pruning its label set, each node uses these remote label sets to update the certainty measures associated with the labels in its own label set. The

[7]Whereas a high-level protocol has been developed to facilitate task-sharing [19], no analogous protocol has emerged from research on result-sharing. We are presently examining the structure of communication for result-sharing with a view to extending the contract net protocol to better incorporate it.

[8]Waltz actually solved this problem using a centralized algorithm that considered only one vertex at a time. The algorithm required 80 iterations to produce a unique labeling for this image.



REGION	POSSIBLE INTERPRETATIONS AND A PRIORI LIKELIHOOD	
R_1	CHAIRLEG	0.8
	TABLELEG	0.2
R_2	CHAIRLEG	0.7
	TABLELEG	0.3
R_3	CHAIRSEAT	0.8
	TABLETOP	0.2
R_4	CHAIRBACK	0.25
	WALL	0.75
R_5	CHAIRBACK	0.25
	WALL	0.75

CONSTRAINTS ON CHAIRSEAT
(ABOVE CHAIRBACK CHAIRSEAT)
(ABOVE CHAIRSEAT CHAIRLEG)

Fig. 10. MSYS: Sample problem.

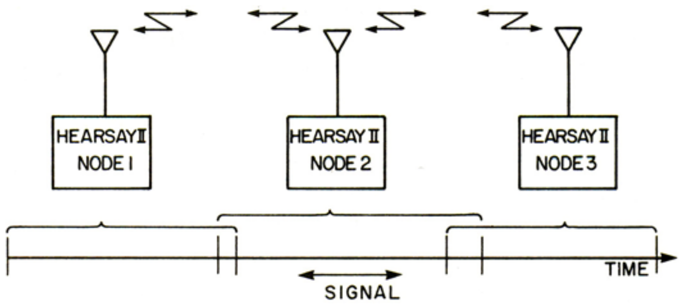


Fig. 11. Distributed interpretation: segmentation.

updating is done on the basis of the interactions among labels described above, and strengthens or weakens the certainty measure for each label.[9] This process continues until unique labels have been established for all nodes (i.e., one label at each node has a large certainty measure with respect to those associated with the other labels for that node) or no further updating is possible.

Fig. 10 shows a sample collection of regions of the type considered by MSYS [2]. For each region, possible interpretations and their *a priori* likelihoods are shown. Also shown are the constraints placed on any region that is to be interpreted as a "chairseat." These constraints increase the certainty that the collection of regions should be interpreted as a chair.

Lesser and Erman [14] have experimented with distribution of the HEARSAY-II speech-understanding system [12]. Distribution has been effected by partitioning each utterance into segments overlapping in time and assigning each segment to a node. Fig. 11 shows the style of segmentation that has been implemented.

Each node attempts to develop an interpretation for the data to which it has access. It does this by creating partial interpretations or hypotheses and testing them for plausibility at each stage of the processing. (This is the classic AI paradigm of hypothesize and test.) A solution is constructed through the incremental aggregation of mutually constraining or reinforcing partial solutions while inconsistent partial solutions die out. Tentative decisions are

[9]Examples of updating algorithms are given in [23].

made on the basis of partial information and then reevaluated when further information becomes available (either in the form of more data or in the form of partial interpretations received from other nodes). Constraint in the speech-understanding domain is offered by the need for consistency of interpretation of the overlapping segments and by the syntactic and semantic constraints that one part of an utterance may place on another part.

The methods used by distributed HEARSAY-II are quite similar to those used by the image labeling systems. Successive refinement of hypotheses is effected in a manner similar to the updating of label sets. However, the image labeling systems achieve cooperation solely by mutual constraint or restriction on the results achieved by individual nodes. Distributed HEARSAY-II takes a more general approach. It achieves cooperation by both mutual restricttion and by mutual aggregation of results achieved by individual nodes (i.e., partial interpretations achieved at neighboring nodes are combined to form more complete interpretations).[10]

In initial tests, the result-sharing approach in distributed HEARSAY-II has demonstrated an interesting ability to deal with ambiguity and uncertainty in data and knowledge. In a variation on the standard *search versus knowledge* trade-off, the result-sharing approach suggests an *aggregation versus knowledge* trade-off: aggregating partial, inexact solutions can at times be much easier than attempting to produce a single, complete and exact solution, and may in fact result in almost no loss of accuracy.

VII. Task-Sharing and Result-Sharing: A Comparison

Task-sharing is used to organize problem decomposition through formation of *explicit* task–subtask connections between nodes. The resultant hierarchy is also useful as a means of structuring answer synthesis. Task-sharing assumes that kernel subproblems can be solved by individual nodes working independently with minimal internode communication and that the major concern is efficient matching of nodes and tasks for high-speed problem solving. It is most useful for problem domains in which it is appropriate to define a hierarchy of tasks (e.g., heuristic search) or levels of data abstraction (e.g., audio or video signal interpretation). Such problems lend themselves to decomposition into a set of relatively independent subtasks with little need for global information or synchronization. Individual subtasks can be assigned to separate processor nodes; these nodes can then execute the subtasks with little need for communication with other nodes. If this is the case, then task-sharing is a sufficiently powerful form of cooperation to handle all three phases of distributed problem solving.

Result-sharing is used to facilitate subproblem solution when kernel subproblems are such that they cannot be solved by individual nodes working independently without-significant communication with other nodes. Result-sharing offers no mechanism for problem decomposition. Hence it can only be used alone as a form of cooperation for problems in which problem decomposition and distribution of subproblems to individual nodes are handled by an agent external to the distributed problem solver. Result-sharing does offer a minimal mechanism for answer synthesis. It is useful in this regard to the extent that the same result-sharing mechanism can be used for overall answer synthesis as well as subproblem solution.[11]

Result-sharing is most useful in problem domains in which 1) results achieved by one node influence or constrain those that can be achieved by another node (i.e., the results are relevant to each other), 2) sharing of results drives the system to converge to a solution to the problem (i.e., results received from remote nodes do not cause oscillation), and 3) sharing of results drives the system to a *correct* solution to the problem.

Minimization of internode communication is important for both the task-sharing and result-sharing forms of cooperation because of the computation/communication speed imbalance in distributed processors. The contract net protocol uses mechanisms like the eligibility specification slot in task announcements to reduce extraneous bid messages, while distributed HEARSAY-II uses a range of interesting mechanisms to limit the number of hypotheses communicated between nodes. One strategy, for example, is to only consider transmission of results for which no further refinement or extension is possible through local processing. (This type of result has been called “locally complete” [14].)

It has previously been stated [13] that the major advantage of result-sharing is its tolerance to uncertainty. However, it is interesting to note that task-sharing can also be used to achieve tolerance to uncertainty. Consider, for example, an application in which three nodes are trying to achieve a consistent interpretation of data that is taken from overlapping portions of an image. In a result-sharing approach they attempt to achieve consensus by communicating partial interpretations of the data. In a task-sharing approach, the three nodes each process their own part of

[10]Nodes in a result-sharing system are faced with a connection problem analogous to that described for task-sharing systems. In the result-sharing case, a node must select, from among all results generated, the particular results to be transmitted, as well as the other nodes to which they are to be transmitted. Similarly, upon receipt of a result, a node must decide whether or not to accept it, and what action to take based on the received result. Furthermore, we cannot generally assume that a node will communicate only with its neighbors. This would preclude the possibility of solving problems that involve *nonlocal interactions* between subproblems (e.g., although shadow regions in blocks world images may not be adjacent, they must be consistent with respect to their relation to the source of illumination). The problem is that we must distinguish between physical adjacency in the communication network and causal or *information-impact* adjacency.

[11]There still remain the problems of deciding when to terminate problem solving activity and deciding which node will communicate the answer to the customer outside the group. In the distributed HEARSAY-II system, all nodes will eventually derive an interpretation for the whole utterance. This may be acceptable for a three node system, but will lead to an unacceptable amount of communication for a larger system.

the data but then, instead of communicating their partial interpretations directly to each other, they communicate them to a fourth node (a manager in contract net terms) that has the task of sorting out the inconsistencies. This node periodically retasks the three other nodes, using the most current data and partial interpretations.[12]

This brief example does bring out a major difference between the two approaches, namely, that result-sharing is a more *implicit* form of cooperation than task-sharing. Cooperation and convergence are achieved by careful design of individual KS's so that they can make meaningful use of results received from remote KS's. Task-sharing, on the other hand, makes the cooperation explicit by setting up formal lines of communication and inserting nodes whose specific task is to integrate the partial interpretations from the nodes that operate on the actual data.

The example also illustrates one of the major unsolved problems in distributed problem solving — how to achieve coherent behavior with a system in which control is distributed among a number of autonomous nodes. When the number of tasks or results that could be processed exceeds the number of available nodes, then nodes with tasks or results to share must compete for the attention and resources of the group.

In the case of task-sharing, mechanisms must be designed that give some assurance that individual subproblems are actually processed, that processors do not get in each other's way in trying to solve identical subproblems while other subproblems are inadvertently ignored. In addition, it is very important that the subproblems that eventually lead to solutions be processed in preference to subproblems that do not lead to solutions. We have suggested negotiation as a mechanism for dealing with these difficulties and have designed the contract net protocol with them in mind [19]. However, it is apparent that much work remains to be done.

In the case of result-sharing, there must be some assurance that nodes influence each other in such a way as to converge to a correct solution. Just as partial results received from a remote node can suggest fruitful new lines of attack for a problem, they can also be distracting. In recent work on result-sharing systems, it has been seen that certainty measures generated at different nodes can be particularly difficult to integrate. In distributed HEARSAY-II, for example, it was found that certainty measures used in a centralized approach are not necessarily appropriate for a distributed formulation. The effect was that remotely generated results sometimes caused nodes to pursue lines of attack that were not as fruitful as the ones they had been pursuing before receipt of those results. Some evidence of this phenomenon can be inferred from the experiments that were performed in which some results were lost in transmission. In some cases, system performance actually improved, an indication that nodes sometimes distract their neighbors. Once again, much work remains to be done in this area.

[12]In organization theoretic terms, the fourth nodes carries out an "integrating role." Hierarchical control of this type is a standard mechanism used by human organizations to deal with uncertainty [9], [8]. In the contract net approach, the managers for tasks are in the best position to perform such duties.

VIII. Conclusion

Two complementary forms of cooperation in distributed problem solving have been discussed: task-sharing and result-sharing. These forms are useful for different types of problem and for different phases of distributed problem solving. Task-sharing is useful in the problem decomposition and answer synthesis phases of distributed problem solving. It assumes that subproblem solution can be achieved with minimal communication between nodes. Result-sharing is useful in the subproblem solution phase when kernel subproblems cannot be solved by nodes working independently without communication with other nodes. It is also helpful to some extent in the answer synthesis phase — in particular, for problems in which the answer synthesis phase is essentially a continuation of the subproblem solution phase. We eventually expect to see systems in which both forms of cooperation are used, drawing upon their individual strengths to attack problems or which neither form is sufficiently powerful by itself.

Acknowledgment

These ideas have evolved as a result of many discussions with our colleagues. The contributions of Victor Lesser, Rick Hayes-Roth, and Steve Zucker have been particular helpful.

References

[1] J. L. Baer, "A survey of some theoretical aspects of multiprocessing," *Comput. Surveys*, vol. 5, pp. 31–80, Mar. 1973.

[2] H. G. Barrow and J. M. Tenenbaum, "MSYS: A system for reasoning about scenes," SRI International, Menlo Park, CA, SRI AIC TN 121, Apr. 1976.

[3] E. K. Bowdon, Sr., and W. J. Barr, "Cost effective priority assignment in network computers," in *FJCC Proc.*, vol. 41. Montvale, NJ: AFIPS Press, 1972, pp. 755–763.

[4] R. E. Carhart, D. H. Smith, H. Brown, and C. Djerassi, "Applications of artificial intelligence for chemical inference — XVII: An approach to computer-assisted elucidation of molecular structure," *J. Amer. Chem. Soc.*, vol. 97, pp. 5755–5762, Oct. 1, 1975.

[5] R. Davis and R. G. Smith, "Negotiation as a metaphor for distributed problem solving," in preparation.

[6] C. R. D'Olivera, *An Analysis Of Computer Decentralization*, Massachusetts Inst. Technol., Cambridge, MIT LCS-TM-90, Oct. 1977.

[7] L. D. Erman and V. R. Lesser, "A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge," in *Proc. 4th Int. Joint Conf. Artificial Intelligence*, Sept. 1975, pp. 483–490.

[8] M. S. Fox, "An organizational view of distributed systems," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-11, pp. 70–80, Jan. 1981, this issue.

[9] J. R. Galbraith, "Organizational design: An information processing view," in *Organizational Psychology*, 2nd ed., D. A. Kolb et al., Eds. Englewood Cliffs, NJ: Prentice-Hall, 1974, pp. 313–322.

[10] C. Hewitt, "Viewing control structures as patterns of passing messages," *Artificial Intelligence*, vol. 8, pp. 323–364, 1977.

[11] D. B. Lenat, "Beings: Knowledge as interacting experts," in *Proc. 4th Int. Joint Conf. Artificial Intelligence*, Sept. 1975, pp. 126–133.

[12] V. R. Lesser, R. D. Fennell, L. D. Erman, and D. R. Reddy, "Organization of the HEARSAY II speech understanding system," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-23, pp. 11–24, Feb. 1975.

[13] V. R. Lesser et al., "Working papers in distributed computation I: Cooperative distributed problem solving," Dep. Comput. and Inform. Sci., Univ. of Massachusetts, Amherst, July 1979.

[14] V. R. Lesser and L. D. Erman, "Distributed interpretation: A model and experiment," *IEEE Trans. Comput.*, vol. C-29, pp. 1144-1163, Dec. 1980.

[15] H. P. Nii and E. A. Feigenbaum, "Rule-based understanding of signals," in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth, Eds. New York: Academic, 1978, pp. 483–501.

[16] N. J. Nilsson, *Problem Solving Methods In Artificial Intelligence*. New York: McGraw-Hill, 1971.

[17] R. G. Smith and R. Davis, "Applications of the contract net framework: Distributed sensing," *Proc. ARPA Distributed Sensor Net Symp.*, pp. 12–20, Dec. 1978.

[18] R. G. Smith, "A framework for problem solving in a distributed processing environment," Dep. Comput. Sci., Stanford Univ., Stanford, CA, STAN-CS-78-700 (HPP-78-28) Dec. 1978.

[19] _____, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. C-29, pp. 1104–1113, Dec. 1980.

[20] D. L. Waltz, "Generating semantic descriptions from drawings of scenes with shadows," Massachusetts Inst. Technol., Cambridge, MIT AI-TR-271, Nov. 1972.

[21] P. H. Winston, *Artificial Intelligence*. Reading, MA: Addison-Wesley, 1977.

[22] S. W. Zucker, R. A. Hummel, and A. Rosenfeld, "An application of relaxation labelling to line and curve enhancement," *IEEE Trans. Comput.*, vol. C-26, pp. 394–403, Apr. 1977.

[23] S. W. Zucker, Y. G. Leclerc, and J. L. Mohammed, "Continuous relaxation and local maxima selection," Dep. Elec. Eng., McGill Univ., Rep. 78-15R, Dec. 1978.