Distributed Automated Reasoning: Issues in Coordination, Cooperation, and Performance

Douglas J. Mac Intosh, Susan E. Conry, *Member, IEEE*, and Robert A. Meyer, *Member, IEEE*

Abstract—**In many domains of interest to distributed artificial intelligence, the problem solving environment may be viewed as a collection of loosely coupled intelligent agents, each of which reasons based on its own incomplete knowledge of the state of the world. No agent has sufficient knowledge to solve the problem at hand so that coordinated cooperative problem solving is required to satisfy system goals. In the paper, a distributed reasoning system (DARES), is presented, in which agents have the ability to focus their attention on selective information interchange to facilitate cooperative problem solving. The experimental results presented demonstrate that agents in a loosely coupled network of problem solvers can work semi-independently, yet focus their attention with the aid of relatively simple heuristics when cooperation is appropriate. These results suggest that an effective cooperation strategy has been developed that is largely independent of initial knowledge distribution.**

I. Introduction

IN THIS PAPER we present a distributed automated reasoning system (DARES) that can perform automated reasoning in, and about, a distributed domain; something that could not previously be done using traditional theorem proving techniques. The technique we have developed for performing automated reasoning will be discussed, with emphasis on issues dealing with task coordination and agent cooperation. Experimental results will also be presented to illustrate the performance of our system, and its ability to perform distributed automated reasoning.

We have chosen distributed theorem proving as the focus of this work for several reasons. First, this domain provides a well defined basis upon which to build. There are very few subtle domain dependent factors that serve to obscure the meaning of the results. Secondly, distributed theorem proving can be viewed as an abstracted "generic domain" for investigation of issues in distributed problem solving. One can certainly represent the knowledge specific to a particular application domain as axioms and express problems to be solved as theorems to be established. Thus we can investigate the general issues of coordination, cooperation, and performance in a relatively domain independent fashion. Finally, the existing body of work on theorem proving provides a set of strategies and techniques that can be extended to distributed environments in a reasonable manner.

Automated reasoning has been used successfully in a wide variety of applications since Robinson's 1965 paper [12]. Automated reasoning remains an area of active research, with interest focused on such areas as development of more efficient strategies [14], parallel implementations of Prolog [5], and parallel architectures designed to perform automated reasoning in near linear time [1], [3]. Our interest is in issues that arise in distributed environments where no agent has complete knowledge. In these environments, successful problem solving requires that an agent ultimately be able to reason based on knowledge that was initially known only to other agents.

DARES is an automated reasoning system for distributed environments that gives an agent the ability to reason beyond the limitations of its local knowledge. DARES' environment can be viewed as a collection of distributed agents that cooperate to perform automated reasoning about the domain. Many concurrent reasoning tasks may proceed simultaneously, and an agent may be involved in solving any subset of the complete set of tasks. Since the domain is distributed, there is no global view. Each agent has its own knowledge of the domain expressed in the first order predicate calculus. Problem solving is initiated when each agent has been provided with one or more problems to solve (expressed as theorems to be established) and a set of axioms that embody its local knowledge.

When a reasoning task is assigned to a group of agents, each agent begins working toward a solution based on its local knowledge. However, since reasoning tasks are distributed among the agents, and each agent has incomplete knowledge, any one of these agents may soon discover that it cannot solve the reasoning task without acquiring further knowledge. DARES provides these agents with a mechanism for importing knowledge in an intelligent manner, so that there is no attempt made by an agent to acquire complete knowledge. Instead, an agent in need of external information assesses its local solution space and heuristically makes requests based on its local progress.

At other times during the reasoning process, an agent may decide that its local efforts do not indicate that progress is being made. When this situation arises, an agent will make a knowledge importation request based on this negative assessment in an attempt to import information that will help to provide insight and direction to its local efforts.

Thus, we see that DARES requires agents to assess their

Manuscript received November 6, 1990; revised April 19, 1990.

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, NY, and in part by the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008 (which supports the Northeast Artificial Intelligence Consortium (NAIC). Some of the results described in this paper were presented at the 1990 National Conference on Artificial Intelligence.

D. J. Mac Intosh is with BP Research, Cleveland, OH 44128.

S. E. Conry and R. A. Meyer are with the Electrical and Computer Engineering Department, Clarkson University, Potsdam, NY 13699.

IEEE Log Number 9102927.

local reasoning efforts in order to determine whether or not nonlocal knowledge is needed. When a knowledge request is made, the informational content of the request is based upon this assessment. This decision making intelligence inherent in DARES is what allows the system to perform automated reasoning about distributed domains that other systems have not yet achieved.

II. System Architecture

In a loosely coupled distributed system, an agent spends most of its cpu time in computation as opposed to communication. Since theorem proving by nature is computationally intensive, we have chosen a loosely coupled implementation for our distributed theorem prover. Each theorem prover agent spends most of its time performing binary resolution[1], with the balance spent on problem assessment and communication. Problem assessment helps determine what course of action to take next to further the proof locally. Communication between agents generally falls into one of the following categories: 1) a request is sent to one or more agents for information; 2) an agent returns information in response to a request.

The architecture for our theorem proving agent has been tailored to suit the characteristics of the problem solving environment mentioned above (i.e., loosely coupled, multiple concurrent tasks). The architecture of a single theorem proving agent is composed of several processes attached to a communications network. The problem solving system is then comprised of several nodes, each having this agent architecture. In each agent there is one mail process, and the remaining processes are each associated with a distinct problem solving activity. Each process in an agent has equal priority and active processes compete for cpu time in a round robin fashion. Under normal circumstances, every theorem prover process is active. The mail process is typically in a wait state and becomes active when new mail is received via the communications channel. Each theorem prover process in an agent has its own environment and is associated with one automated reasoning task identified by a unique tag. Theorem prover processes working on the same reasoning task in different agents throughout the network bear the same tag. In addition, no two theorem prover processes for a given agent may work on the same theorem. It need not be the case that every agent works on every theorem.

III. Distributed Theorem Proving Strategies

As is the case with single agent theorem provers, distributed theorem proving exhibits exponential behavior. It turns out, however, that some of the strategies used in classical theorem proving to help minimize the number of resolvents generated can also be used in the distributed case to reduce the content of information exchanged between agents. Development of these of strategies is essential, since the performance of distributed theorem proving can be greatly enhanced by them. If the computational effort in replying to a request is significant, it may have not been worth making the request in the first place. Similarly, in information-intensive domains requests that receive a bombardment of replies can be counterproductive.

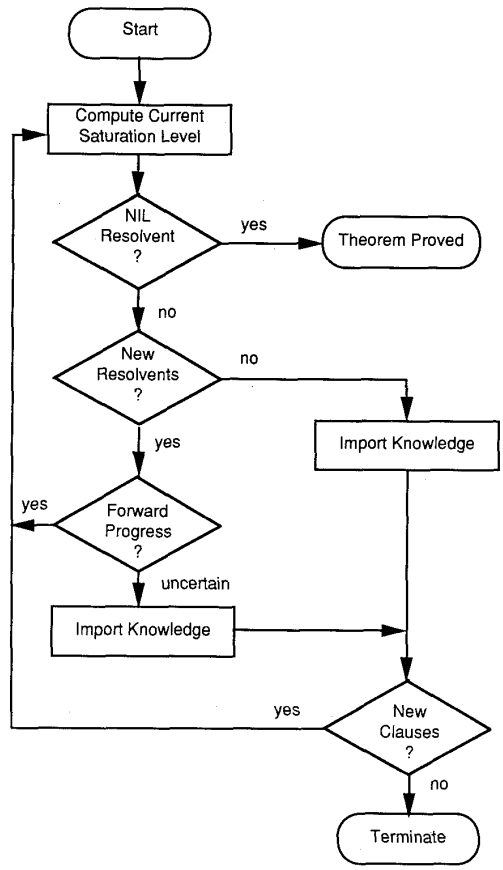


Fig. 1. Distributed automated reasoning flow diagram.

Fig. 1 is a flow diagram that depicts a high level view of our approach to distributed theorem proving. DARES is based on a traditional saturation level type theorem prover [12].

It has been noted that in our distributed environment no one agent can achieve the task at hand by itself. Therefore, we do not terminate the resolution process simply because the current resolution level has failed to generate new resolvents. In fact, reaching this point causes a theorem prover agent to attempt to import relevant information from other agents as shown in Fig. 1. If an agent is successful in importing new knowledge, the resolution process continues. Otherwise the distributed theorem proving process terminates with no conclusion being drawn about the current theorem.

It is not sufficient just to wait for local resolution to halt prior to importing new knowledge, since mechanical theorem proving in general may not terminate when dealing with incomplete knowledge. Therefore, as shown in Fig. 1, our system must evaluate whether or not progress is being made

[1] The binary resolution performed by each agent in DARES uses a tautology and subsumption reduction strategy [2] to minimize the number of resolvents generated, and it uses the set of support strategy [16] to limit its search space. Furthermore, the set of resolvents generated by each level of resolution is sorted by length [15], so that shorter clauses are resolved first during the next level.

locally toward a solution. The *forward progress* test is made at the conclusion of each resolution level to heuristically determine whether the proof has advanced toward a NIL resolvent. This test cannot give a definitive answer. It can only say "Yes, progress has been made," or that it "does not know." On this basis, local resolution moves to the next level only if the forward progress test concludes that progress has been made. Otherwise, nonlocal knowledge is imported just as if the current level had failed to produce new resolvents.

A. Forward Progress Heuristic

Given a negated theorem and a set of axioms that is adequate for establishing that theorem, a single agent with complete knowledge will, given enough time, eventually determine that the theorem is valid by producing a NIL resolvent. However, when the same axiom set and theorem are distributed over several agents, it is possible that no agent will have sufficient local knowledge to establish the result. It is therefore possible for each agent under this circumstance, without a forward progress heuristic, to perform resolution forever and not prove a theorem that conventional systems have no difficulty in proving.

The purpose of our forward progress heuristic is to guarantee that no agent in our system will enter a mode in which it can perform resolution forever, without eventually attempting to import nonlocal knowledge that may lead to an inconsistency. This safeguard gives DARES the ability to prove any theorem that conventional systems can prove given the same set of global system knowledge. What the forward progress heuristic does not do is to give DARES any advantage over the single agent case when the single agent is faced with nontermination.

When it is apparent that an agent is not progressing, the forward progress heuristic triggers the importation of nonlocal information in order to increase the agent's knowledge relative to the task at hand. Experimental data indicate that this heuristic does indeed enhance system performance, even when a DARES agent is not faced with nontermination.

In defining the forward progress heuristic, we make use of two related heuristics: the *proof advancing heuristic* and the *monotonic search heuristic*. The proof advancing heuristic is the component of the forward progress heuristic that is used to detect local advancement. This heuristic either detects advancement, or is uncertain whether or not the proof is advancing. The monotonic search heuristic is the mechanism that the forward progress heuristic relies upon if uncertainty about proof advancement persists.

The *proof advancing heuristic* is determined at each level of resolution by examination of the newly generated resolvents. A resolvent R is said to be *advancing the proof* if:

a) given two clauses C and D with literal length c and d respectively, the clause length r of R is less than $(c + d) - 2$,
or
b) R is a single literal,
or
c) R was generated from a single literal clause.

If any resolvent generated at saturation level i advances the proof, then we say the proof advancing heuristic is *satisfied* at level i.

In general, when two clauses are resolved using binary resolution, the resolvents will always have length no greater than the sum of the lengths of the two parent clauses, minus the two literals that are consumed by the resolution process. Condition a) in the proof advancing heuristic considers a proof to be advancing whenever a resolvent is generated with length less than this upper bound. Conditions b) and c) in the proof advancing heuristic definition recognize that some resolvents are desirable even though their length equals the upper bound. For example, when a single literal clause C is resolved with clause D of length n, the resolvent has length $c + d - 2 = 1 + n - 2 = n - 1$. The importance of these resolvents is recognized by the *Unit Preference* [15] strategy.

Whenever condition a), b), and/or c) occurs during resolution, the proof is considered to be *advancing*. If none of these occur, we do not know if the proof is making progress. If it is not clear whether or not a proof is advancing, and this uncertainty persists, some mechanism for forcing a knowledge request is required. This is where the monotonic search heuristic comes into play.

The *monotonic search heuristic* is defined as follows:

Let α_n be the total number of distinct predicate symbols found in the set of newly generated resolvents at saturation level n. The search for a proof is said to be *monotonic* at level i if for $i > 1$, $\alpha_{i-1} > \alpha_i$.

The *forward progress heuristic* is used to detect an apparent lack of forward progress in the proof. This lack of progress is defined in terms of the proof advancing and monotonic search heuristics.

A proof is said to exhibit an apparent lack of *forward progress* at saturation level i if:

1) the proof advancing heuristic is not satisfied at saturation level $i - 1$,
and
2) the proof advancing heuristic is not satisfied at saturation level i,
and
3) the search is not monotonic at level i.

The forward progress heuristic guarantees that if proof advancement is uncertain and the number of predicate symbols is nondecreasing in successive levels of resolution, a knowledge request is made. (Whenever the number of predicate symbols is a decreasing function in successive levels of resolution, the number of predicate symbols must eventually shrink to zero. If this situation occurs, it will be detected by the *New Resolvents* test (refer to Fig. 1), since a scenario involving zero predicate symbols can only occur if the current level of resolution fails to generate new resolvents.)

B. Priority Set

In our environment, when an agent has reached a point where it is evident that new information must be acquired in order to continue problem solving, it formulates a *Priority Set* P.

Definition A *Priority Set* P has the form $P = \{C_1, \ldots, C_n\}$ where each C_i for $0 < i \leq n$ is a clause heuristically determined to have a high likelihood of furthering the proof toward a NIL resolvent. P is said to have length n, where n is the number of clauses in P.

The heuristic we use to determine the *likelihood* of a clause extracts some ideas found in two conventional resolution strategies: *Set of Support* [16] and *Unit Preference* [15]. However, our heuristic is more than just a combination of these two strategies. Our importation heuristic determines a *likelihood* that a clause will be relevant in furthering a proof toward a NIL resolvent. This heuristic is based on clause length and clause ancestry. Clauses whose ancestries do not lead back to the negated theorem have no *likelihood* and are assigned the value of 0. Clauses having an ancestry link to the negated theorem have a *likelihood* whose value is the reciprocal of the clause length. Single literal clauses with a negated theorem ancestry have the maximum likelihood of 1.

As a first cut in distributed theorem proving, one could simply form the Priority Set P using all clauses possessing maximum likelihood. Then P could be sent to all agents, with each agent being requested to return any clause that can resolve with one or more members in P. Unfortunately, P could potentially be large, requiring significant processing on behalf of each agent receiving the request. A better strategy would be to first remove any clause in P, which is subsumed by another clause in P, as any reduction in the size of P reduces the overhead other agents incur while processing the request.

Though use of subsumption in this way reduces the size of P, it still has the potential of being relatively large. An alternative approach makes use of a Minimal Literal Set $L_{\min}$ derived from P that is defined as follows:

Definition: Let each clause C_i in a Priority Set P of length n be of the form $C_i = \{L_{i1}, \ldots, L_{im_i}\}$ where L_{ij} is a literal, and:

1) $1 \leq i \leq n$;
2) $m_i > 0$ and is the number of literals in clause i;
3) $1 \leq j \leq m_i$.

Then the *priority literal set* L is defined to be the union of literals found in clauses $C_1, \ldots, C_n$ and has the form $L = C_1 \cup \ldots \cup C_n$.

Definition Given L, the priority literal set for $P = \{C_1, \ldots, C_n\}$, we define $L_{\min}$, the *minimum priority literal set* for P as follows:

$L_{\min} = L - L'$, where $L' = \{L_{jk} \in L \mid$ there is a literal L_{pq} in L, such that L_{jk} is subsumed by $L_{pq}\}$.

After computing the minimal priority literal set $L_{\min}$ from the priority set P, the agent could transmit $L_{\min}$ to other agents and request knowledge about clauses they may have that resolve with one or more literals in $L_{\min}$. If this were done, an agent responding to this request would then systematically attempt to perform resolution with each literal in $L_{\min}$ against its local clause set, complementing each $L_{\min}$ literal and attempting to unify it with a literal in each of its local clauses. Recognizing this fact, in an attempt to minimize the effort of an agent replying to a request, the requesting agent complements each literal in $L_{\min}$ prior to making the request. We call the resulting set the minimum priority negated literal set and define it as follows:

Definition Given a minimum priority literal set $L_{\min} = \{Q_1, \ldots, Q_n\}$ of length n, where each Q_i is a literal for $0 < i \leq n$, then the *minimum priority negated literal set* $NL_{\min}$ has the form $NL_{\min} = \{R_1, \ldots, R_n\}$, where each $R_i = \neg Q_i$ for $0 < i \leq n$.

After computing the Minimal Priority Negated Literal Set $NL_{\min}$ from the Priority Set P, the agent transmits $NL_{\min}$ to other agents and requests knowledge about clauses they may have that unify with one or more literals in $NL_{\min}$.

Up to now we have concentrated on explaining how an agent determines when a request needs to be made, and how it formulates the content of the request. We have said nothing about what happens if the first attempt to import knowledge fails, nor have we given much insight into the procedure followed by an agent replying to the request.

The first thing the requesting agent does is to determine the range of likelihoods possible for its clause set. Beginning with the clauses having highest likelihood, the agent computes its Minimum Literal Priority Set and broadcasts a request based on this set. This set is a function of likelihood ($NL_{\min}$(l)). If the request based on maximum likelihood fails to import nonlocal knowledge, the likelihood constraint is relaxed to its next possible value and the agent makes another request. This process continues until the requesting agent is successful in importing knowledge, or the agent has exhausted its clause set.

We have found that it is beneficial for agents replying to knowledge requests to also incorporate a likelihood dependency. When an agent makes a request with a high likelihood, the knowledge requested is specific in nature and it should receive information back that is also relatively specific. As requests are made based upon lower likelihoods, we have observed that the requested information encompasses a wider spectrum and is more general in nature.

In DARES, we have incorporated a simple strategy into request processing, which links the likelihood of a request to the scope of the search space that an agent considers when making its reply. We require that the likelihood l, used to formulate $NL_{\min}$(l), be used to determine which clauses in a theorem prover's environment are to be considered when evaluating the request. In order for a clause to be deemed a candidate for consideration, it must have length no greater than the maximum length of any clause found in the Priority Set used to generate $NL_{\min}$(l).

When a clause satisfies the requirements of the request, it is tagged to be considered later as part of the reply. The significance of tagging potential clauses during the unification process is twofold: once a clause is tagged, it is never again considered when subsequent requests are made by the same agent with respect to the current theorem under investigation. Secondly, subsumption is used among the tagged clauses to minimize what is returned to the requesting agent. This tagging mechanism helps avoid redundancy in what is returned in

response to subsequent requests. In addition, tagging can be viewed as an aggregation of knowledge about other agents' activities (not unlike the behavior evident in the scientific community metaphor [6]–[8], [13]), although DARES makes no specific use of this information at this time.

IV. Experimental Results

There are five key issues that have been addressed in our experiments using DARES. They are:

1) How is DARES' problem solving behavior affected as the number of active agents is varied?
2) How are anomalies in system behavior that result from particular knowledge distributions minimized, so that DARES' automated reasoning behavior is not misconstrued?
3) What effect does the amount of shared knowledge throughout the network have on system performance?
4) How does the amount of shared knowledge in the system and/or network size influence the interactions among agents?
5) To what extent does the forward progress heuristic affect distributed automated reasoning?

In order to measure the amount of shared knowledge among agents in a distributed reasoning network, we introduce the notion of a *relevant axiom* and the *redundancy factor.*

Very simply, an axiom is considered *relevant* if it can be used in some proof of the current reasoning task. If S is a clause set and P a logical consequence of S, then in general there may be more than one subset of S from which P can be deduced. We do not address this issue here. Instead, we presume that each and every clause in S is required in order to derive P.

The *redundancy factor (R)* of a network is a global measure of the amount of relevant knowledge shared among the agents. When no relevant knowledge is shared between agents, the redundancy factor is 0. When every agent has complete relevant knowledge the network redundancy factor is 1. For distributions falling within these boundaries, we define the redundancy factor to be:

$$R = \frac{\sum_{i=1}^{k}(m_i - C)}{k(N - C)} \tag{1}$$

where k is the number of active agents, N is the number of relevant axioms for the reasoning task, $C = (N/k)$, m_i is the number of local axioms known to agent i, and R is the redundancy factor.

A. Performance Data

Each of our experiments corresponds to one distributed reasoning task. For every experiment, data is collected over a wide range of values for each system parameter. As these parameters are varied, each new run begins with a different random distribution of knowledge for the same task. For each distribution, there are three constraints that must be met. First, each active agent must initially be given at least one axiom. Secondly, multiple copies of an axiom are not permitted in an agent's local environment. (But n copies of an axiom may exist across several agents.) Lastly, the total number of axioms distributed throughout the system must equal the number specified by the current redundancy factor.

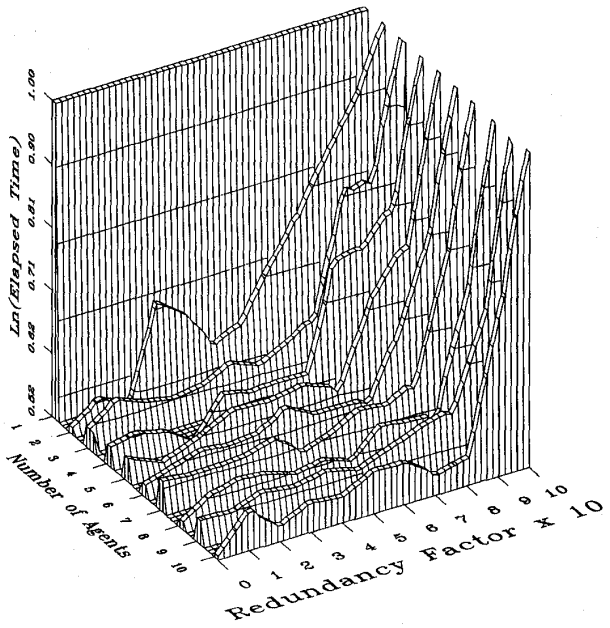


Fig. 2. Performance characteristics for an experiment.

There are two data collection parameters in each experiment: k and r. Parameter k corresponds to the number of agents actively engaged in the reasoning task, and r is the redundancy factor. Given a theorem to prove which is comprised of M axioms and N negated theorem clauses, the experiment is done utilizing k agents, where k is varied between 1 and M. For each value of k, the knowledge distribution is varied between 0 and 100% redundancy.

In each experimental scenario, we minimize the effects that a particular knowledge distribution has on general behavior, by performing on the order of ten passes at a given data point (k, r), with each pass having a different distribution. Likewise, the behavioral characteristics of DARES can be determined by performing a number of different experiments, where each experiment is based on a unique automated reasoning task. In fact, this has been done. The results presented here are based upon analysis of many different experiments. The figures incorporated in this paper reflect the results from one typical experiment as a vehicle for demonstrating DARES' distributed automated reasoning behavior.

The performance characteristics for this typical experiment are given in Fig. 2. These characteristics are normalized to the nondistributed case, in which a single agent performs the complete reasoning task alone.

The two system parameters for DARES' data collection experiment, k and r, correspond to the "Number of Agents" and "Redundancy Factor" axes, respectively, in Fig. 2. Since DARES performs distributed automated reasoning approxi-

mately one order of magnitude faster than its nondistributed counterpart, we have taken the natural log of the elapsed time data for the experiment. Furthermore, we have normalized this data so that the single agent case has a value of unity. The redundancy factor axis has been scaled by a factor of 10 for legibility.

In general, when there is a very high level of redundancy within the agents' local knowledge, DARES' runtime rapidly begins to approach that of the single agent. This behavior can be attributed to a very low interaction rate among agents, since each agent has sufficient local knowledge to advance the proof to near completion. The forward progress heuristic detects this advancement and does not initiate any knowledge importation requests. Instead, local advancement continues until resolution fails to generate any new resolvents. It is at this point that a knowledge request is made to import nonlocal information.

Requests made in high redundancy environments tend to be very specific in nature, since an agent has advanced the proof nearly to completion prior to making the request. When a request of this sort is made, the knowledge sought is readily available in the network. This is a direct consequence of the other agents having near complete knowledge, and of the fact that they have had ample time to advance their local efforts.

In general, as the number of agents in a network increases, replies to knowledge requests tend to have better informational content. This relationship between network size and reply content is a direct consequence of simply having more agents in the network to query, each agent having a different local perspective derived from its differing initial knowledge distribution. In Fig. 2 the effect of this relationship is evident in the high redundancy areas. Note that as the number of agents increases, the slope associated with the rapid approach toward the single agent case becomes steeper and the width of these peaks decreases.

Fig. 2 also suggests that performing distributed theorem proving is best done by many agents possessing little redundancy. In this situation, each agent can be viewed as a specialist. At the start of the distributed theorem proving process, each agent advances its part of the proof as far as it can before making a knowledge request. At the time such a request is made, the agent has begun to concentrate its efforts on its local advancement and imports knowledge relative to this acquired focus. Since redundancy is low, we see the agents becoming specialized in different areas, which reduces search space overlap between agents and leads to enhanced system performance.

Although system performance is best with many agents in a low redundancy environment, there is only a small increase in system performance as the number of agents is increased. We observe that as network size becomes larger, initial knowledge importation requests occur earlier. These requests are based on less local effort and are more general in nature, thus leading to the initial importation of larger volumes of knowledge. Therefore, agents in smaller networks acquire a focus sooner than agents in larger networks. However, as the distributed effort nears completion, larger networks seem to outperform smaller ones, since requested information near the end of the proof is more readily available in larger networks due to

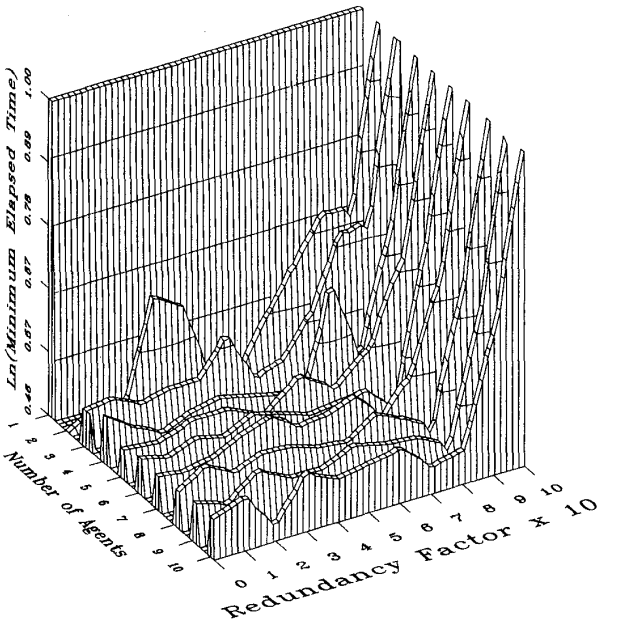


Fig. 3. Lower bound time characteristics.

network size and the local efforts of more agents. It appears that the performance of larger networks near the end of the proof is sufficient to offset the advantage smaller networks have at the start, thus we see a slight improvement in system performance as network size increases.

The behavioral characteristics displayed in Fig. 2 reflect DARES' general runtime characteristics. This surface plot was generated from data recorded by DARES and represents the average value of all passes made for each data point. The validity of our conclusions is supported by a comparison of the characteristics of both the lower and upper bound surfaces. These surfaces are shown in Figs. 3 and 4 respectively.

The lower bound characteristic surface is a plot of the minimum runtime over all passes made for a given data point. Note that there are no major discrepancies in the plateau regions between the general and minimum surfaces. In fact, the smoothness of the plateau in the average value surface suggests that we have indeed minimized dependencies associated with particular distributions.

The upper bound characteristic surface shown in Fig. 4 is a plot of the maximum of all runtimes recorded by DARES for a given data point. This plot also has (to a limited degree) the same general shape as the average time characteristic surface in Fig. 2. However, the most notable feature of this surface is that it is very *spiked*. This is a demonstration of the degree to which distributed automated reasoning is sensitive to its knowledge distribution. Each one of the peaks (spikes) represents reduced system performance that is directly related to the knowledge distribution used.

There are two important observations to be made with respect to the upper bound surface: First, since its overall shape is basically that of the average time surface, we have

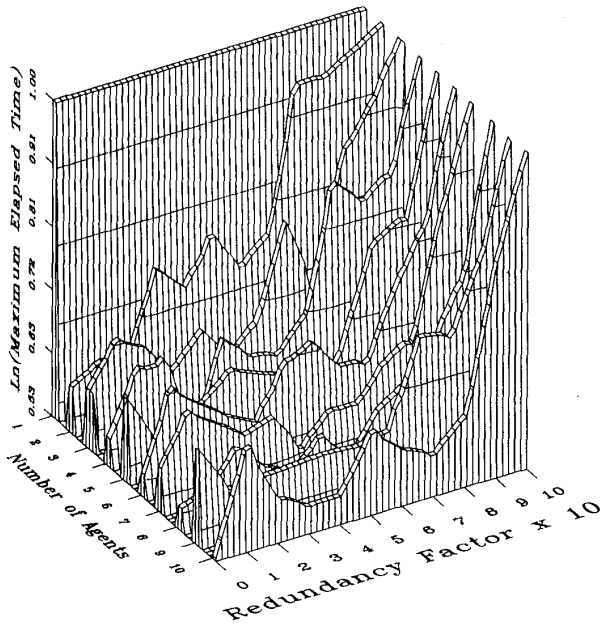


Fig. 4. Upper bound time characteristics.

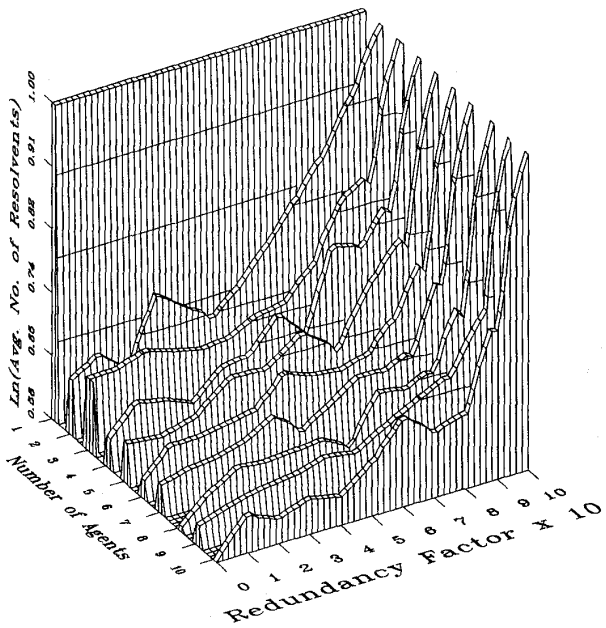


Fig. 5. Average number of resolvents per agent for an experiment.

further evidence supporting the hypothesis that the average time characteristics do represent those of general behavior. Secondly, the noise in this plot indicates how a *relatively poor distribution* can affect system performance. More importantly, it demonstrates that DARES in its worst case environment performs automated reasoning at a reasonable rate relative to the that of the nondistributed case.

B. Resolvent Behavior

So far, in our discussion of experimental results, we have concentrated on DARES' elapsed time characteristics. But what about DARES' resolvent behavior? One significant factor in a distributed problem solving system such as DARES is the amount of communication overhead incurred due to inter-agent cooperation. In DARES, the communication overhead is directly related to the resolvent behavior. Each request for knowledge not locally available imposes communication overhead, as does each transmission of resolvents made by one agent to another. In this subsection, we discuss the ways in which redundancy and network size appear to impact the resolvent behavior (and consequently the communication overhead).

In conventional automated reasoning, there is a strong correlation between the number of resolvents generated by a theorem proving strategy, and the amount of runtime required to perform the task. One would expect this correlation to hold when performing distributed automated reasoning. Figs. 5 and 6 demonstrate that this is indeed the case.

Fig. 5 is a surface plot reflecting the total number of resolvents generated throughout the entire network for our example reasoning task, divided by the number of active agents. Since the average number of resolvents generated per

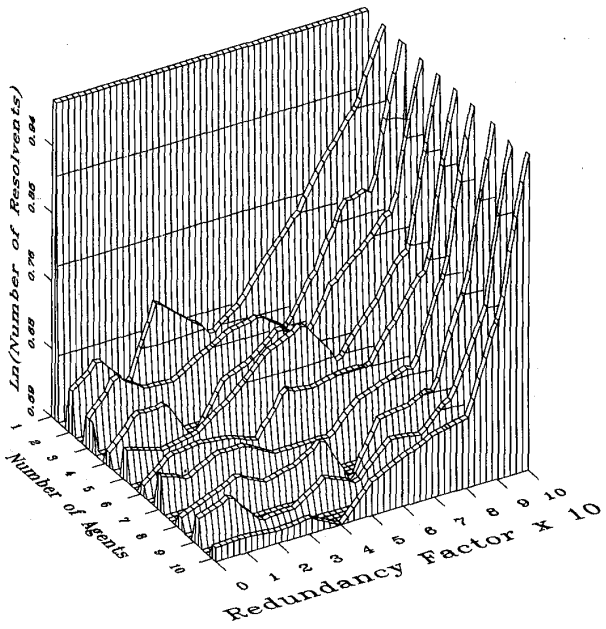


Fig. 6. Number of resolvents generated by first agent completing reasoning task.

agent is significantly less than in the equivalent nondistributed case, we have again taken the natural log of this number. These results have also been normalized so that the single agent case has a value of unity. Similarly, Fig. 6 depicts the number of resolvents locally known to the agent that reaches a solution to the experiment's reasoning task first.

Looking at these resolvent surfaces, it is clear that there is one phenomenon that is more pronounced here than in the

time plots. For low redundancy, as the number of agents increases, the number of resolvents decreases at a greater rate than does the amount of time required to perform the reasoning task. This difference can probably be attributed to an increase in communications overhead encountered as the number of agents performing the task increases. This increase in communication overhead occurs because an agent broadcasts its knowledge importation requests to all agents. As the number of agents in the network increases, so does the number of requests handled by each agent. Although this relationship between communication overhead and network size exists, it does not appear to hamper DARES effectiveness in performing distributed automated reasoning, as low redundancy and large network size result in DARES' best time performance characteristics.

The correlation between runtime and number of resolvents in conventional theorem proving led us to anticipate the results just presented. It was not surprising that this correlation existed in DARES. However, as we began to look at DARES' imported resolvent behavior, some very interesting behaviors were observed.

Before discussing the imported resolvent behavior, it should be noted that we have changed the angle of view in figures depicting this behavior. These surfaces have been rotated by 180° around the Z axis in order to obtain the best angle for viewing them. As a result, it should be observed that the closest point in these diagrams now represents maximum redundancy and minimum network size. In the previous surface diagrams, this point represented minimum redundancy and maximum network size. This observation angle is used for all remaining figures in this paper.

Fig. 7 is a surface plot for the typical reasoning experiment showing the relationship between the average number of resolvents imported per agent and the knowledge redundancy factor. This relationship appears to be linear and independent of network size. For 100% redundancy, Fig. 7 shows that no knowledge is imported by the agents. This is to be expected, since each agent has complete knowledge and is able to perform the reasoning task independently. With the exception of the two agent network, there is approximately a linear increase from 0 to 6 in the average number of imported resolvents per agent as redundancy is decreased from 100% to 0%.

This linear relationship has been observed in DARES for many other reasoning tasks. As is to be expected, the maximum average number of imported resolvents occurring at 0% redundancy is a characteristic of the reasoning task being performed. The reason for this apparent linear behavior is not known.

Fig. 8 shows the relationship between the number of imported resolvents by the agent completing the reasoning task first and the redundancy factor. For this case, the relationship still appears to be somewhat linear, but is dependent upon network size. As the network size is increased from two to ten agents in this problem, there is approximately a linear increase in the number of imported resolvents from 7 to 10 respectively. Furthermore, in every case, the number of resolvents imported by the agent finishing first is greater than the average number of imported resolvents per agent shown in Fig. 7.

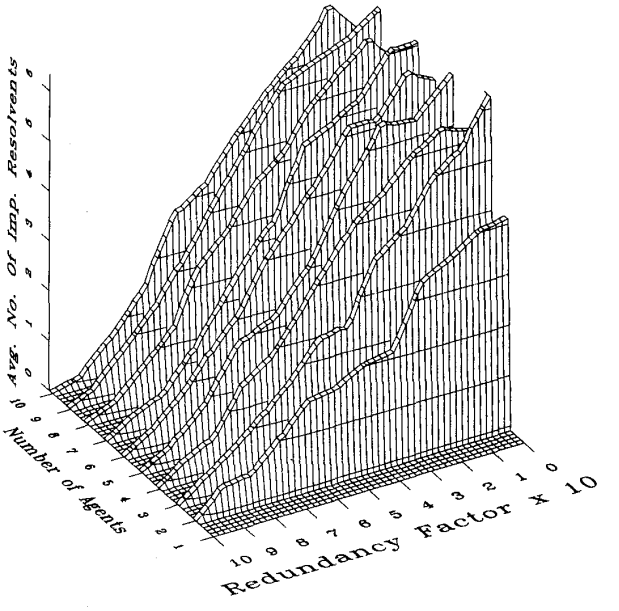


Fig. 7. Number of imported resolvents per agent for an experiment.

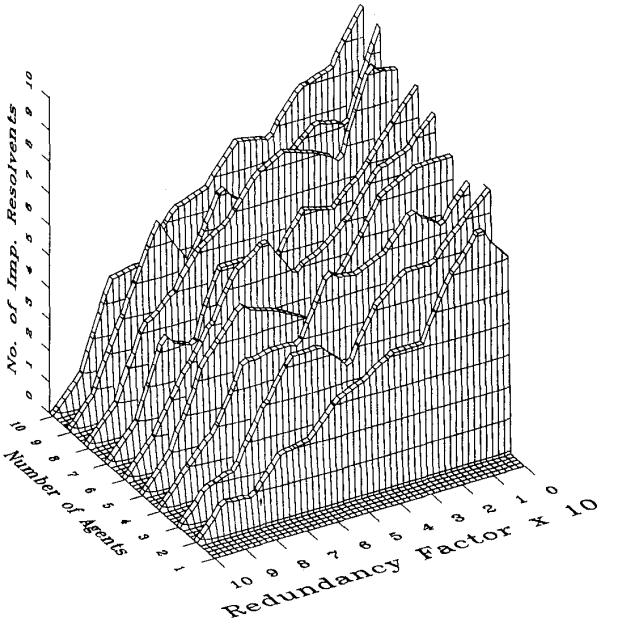


Fig. 8. Number of resolvents imported by first agent completing reasoning task.

We do have some insight as to why the agent finishing the reasoning task first imports more knowledge than the network average. This has to do with the fact that when an agent has incomplete knowledge, it must at some point import information from the other agents in order to complete the task. Looking at the history recorded by DARES when performing

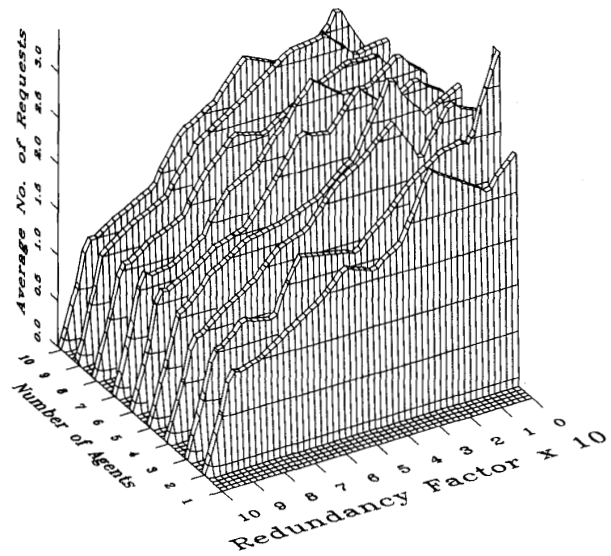


Fig. 9. Number of importation requests per agent for the experiment.

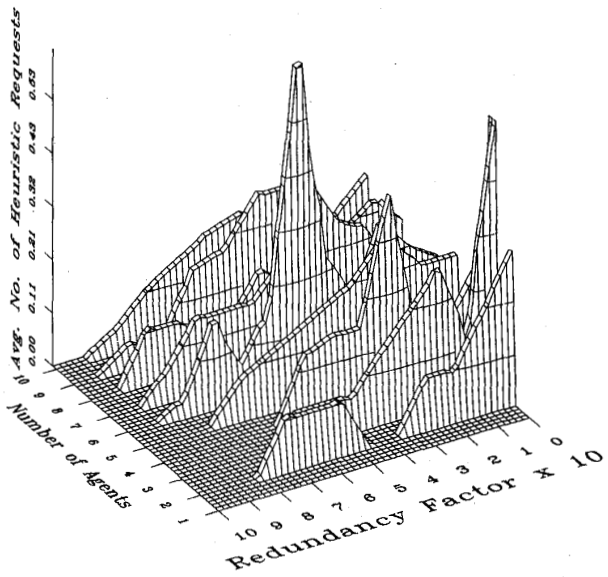


Fig. 10. Number of heuristic requests per agent for the experiment.

automated reasoning, it can be seen that the agent finishing the reasoning task first is also the agent making its last knowledge request first. Therefore, at the time the task is completed by the first agent, it has imported more information (on the average) than the other agents, since they have not yet made their final request.

Also of interest to us, with respect to DARES' resolvent behavioral characteristics, is the number of requests made by an agent and the way in which this number is affected by network size and/or knowledge redundancy. We would like to ascertain where the forward progress heuristic comes into play during the knowledge importation process.

In Fig. 9, the average number of knowledge importation requests made per agent for this typical reasoning experiment is given. We see that for 100% redundancy, no knowledge is imported by the agents. This is expected, since each agent has complete knowledge and does not need to import nonlocal information. Furthermore, the forward progress heuristic should be able to detect advancement when an agent has complete knowledge.

The straight line of constant slope between 90% and 100% redundancy in Fig. 9 is a consequence of the way in which the data for the this experiment was generated and plotted. For the two agent network and this reasoning task, there is no value for redundancy between 90.9% and 100% (see (1)). Furthermore, data was collected for the same redundancy values throughout the experiment, so the two agent network determined the values of redundancy that were sampled. In the experiment, independent of network size, the average number of requests made per agent was 0 and 1 for 100% and 90.9% redundancy, respectively. The straight line connecting these successive data points is the result of using a continuous plotting program with discrete data.

As redundancy decreases, the average number of requests made by an agent steadily increases to a maximum value occurring at 0% redundancy. Furthermore, this phenomenon appears to be independent of network size. The similarity of this relationship to that of the average number of imported resolvents shown in Fig. 7 is very interesting.

The average number of knowledge importation requests made per agent that were initiated by the forward progress heuristic in this sample experiment is shown in Fig. 10. This Fig. gives us some insight as to where, and how much this heuristic is used to initiate knowledge requests during distributed automated reasoning. For this example, typically 0.3 requests were made per agent by the forward progress heuristic. This implies that for the most part, DARES is driven by requests due to local resolution halting, not because local advancement cannot be detected. This behavior is exactly what we would like to see. Furthermore, we have noticed that when a request is initiated by this heuristic, it can significantly enhance system performance.

Fig. 10 indicates that middle range to low values of redundancy are more likely to result in knowledge requests due to the lack of local advancement detection. It also appears to be the case that as network size increases, the forward progress heuristic is triggered more often.

V. Conclusion

In this paper, we have described DARES, our distributed automated reasoning system. This system extends the use of automated reasoning to domains and environments that have not previously been handled by conventional reasoning techniques. Experiments with DARES have provided us with enhanced insight into the role of knowledge in distributed problem solving. We have seen cases in which performance can be very sensitive to initial knowledge distribution, but the average case statistics indicate that one must be unlucky to

encounter such a distribution when knowledge is randomly distributed.

More importantly, the experimental results we have presented demonstrate that agents in a loosely coupled network of problem solvers can work semi-independently, yet focus their attention with the aid of relatively simple heuristics when cooperation is appropriate. These results suggest that we have developed an effective cooperation strategy that is largely independent of initial knowledge distribution. In addition, the behavior we have observed in analyzing the number of resolvents generated by agents during problem solving and the patterns of knowledge exchange indicate that the overhead (in terms of communication costs) is not excessive.

The results presented in this paper can be viewed in a number of ways. One of these is associated with the design of distributed intelligent systems. It has been observed that the best performance appears to be gained in low redundancy situations with many agents. This would suggest that systems with a number of "specialist" agents and little overlap in the areas of agent specialization could solve problems effectively.

It is interesting to speculate about the relationship between the results we have observed and the behavior observed in other distributed problem solving systems. For example, agents in the DVMT [9] are able to focus their attention in achieving coherent behavior through an iterative exchange of hypotheses among agents. The use of partial global planning [4] acts to further guide problem solving behavior so that agent activity is more coherent. In DARES, agents exchange partial results based on heuristics that have a strong unit preference and clause ancestry bias. It is possible that the underlying reasons for the improved behavior of these systems is due to the fact that relatively high level abstracted knowledge is being used to guide the search in both cases.

DARES has been implemented in a distributed testbed facility, SIMULACT [10], that runs on a network of Lisp machines. SIMULACT has provided a well instrumented environment in which applications of this kind are easily developed and maintained.

REFERENCES

[1] P. E. Allen, S. Bose, E. M. Clarke, and S. Michaylov, "PARTHENON: A parallel theorem prover for nonhorn clauses," in *9th Int. Conf. Automated Deduction*, May 1988, pp. 764–765.

[2] C. L. Chang and R. C. Lee, *Symbolic Logic and Mechanical Theorem Proving*. New York: Academic, 1973.

[3] M. P. Cline, "A Fast Parallel algorithm for N-ary unification with A.I. applications," Ph.D. thesis, Clarkson Univ., Potsdam, NY, Apr. 1989.

[4] E. H. Durfee and V. R. Lesser, "Using partial global plans to coordinate distributed problem solvers," *Proc. Tenth Int. Joint Conf. Artificial Intell.*, 1987, pp. 168–175.

[5] B. S. Fagin and A. M. Despain, "The performance of parallel prolog programs," *IEEE Trans. Computers*, vol. C-39, pp. 1434–1445, Dec. 1990.

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

[7] W. A. Kornfeld and C. E. Hewitt. The scientific community metaphor, *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, pp. 24–33, Jan. 1981.

[8] V. R. Lesser and D. D. Corkill, "Functionally accurate, cooperative distributed systems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, pp. 81–96, Jan. 1981.

[9] V. R. Lesser and D. D. Corkill, "The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks," *AI Magazine*, vol. 4, no. 3, pp. 15–33, Fall 1983.

[10] D. J. MacIntosh and S. E. Conry, "SIMULACT: A generic tool for simulating distributed systems," in *Proc. Eastern Simulation Conf.*, Orlando, FL The Society for Computer Simulation, Apr. 1987, pp. 18–23. (also available as NAIC Technical Report TR-8713).

[11] D. J. MacIntosh, "Distributed automated reasoning: The role of knowledge in distributed problem solving," Ph.D. thesis, Clarkson University, Potsdam, NY, Dec. 1989.

[12] J. A. Robinson, "A machine-oriented logic based on the resolution principle," *J. ACM*, 12(1):23–41, Jan. 1965.

[13] R. G. Smith and R. Davis, "Frameworks for cooperation in distributed problem solving," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, pp. 1–70, Jan. 1981.

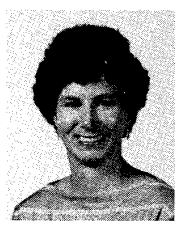
[14] L. Wos, *Automated Reasoning: 33 Basic Research Problems*. Engelwood Cliffs, NJ: Prentice-Hall, 1988.

[15] L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning: Introduction and Applications*. Engelwood Cliffs, NJ: Prentice-Hall, 1984.

[16] L. Wos and G. A. Robinson, "Paramodulation and set of support," in *Proc. IRIA Symp. Automatic Demonstration*, 1968, pp. 267–310.

Douglas J. MacIntosh received the B.S. degree in electrical and computer engineering in 1983, the M.S. degree in electrical and computer engineering in 1985, and the Ph.D. degree in engineering science in 1989, all from Clarkson University, Potsdam, NY.

He is an Engineering Specialist for BP America, which he joined in 1989, and works at the Warrensville Research and Environmental Science Center, Cleveland, OH. He is a member of the Information Technology Croup. His research interest include distributed problem solving, knowledge networking, and distributed collaborative systems.



Susan E. Conry (M'77) received the bachelor's degree in mathematics in 1971, and the M.S. and Ph.D. degrees in electrical engineering in 1973 and 1975, all from Rice University, Houston, TX.

She is an Associate Professor in the Department of Electrical and Computer Engineering at Clarkson University. She has been interested in various aspects of parallel and distributed computation for many years. Most recently, her research has been largely concerned with distributed problem solving and distributed planning systems.

Robert A. Meyer (S'69–M'70–S'71–M'72) received the bachelor's, and master's degrees in 1970, and the Ph.D. degree in 1974, all in electrical engineering from Rice University, Houston, TX.

He is an Associate Professor in the Department of Electrical and Computer Engineering at Clarkson University, Potsdam, NY. He worked in industry with Exxon Corp. and Texas Instruments, Dallas, TX. prior to his graduate work. Since joining to Clarkson University, he has taught courses in nearly all areas of computer engineering and conducted research on various problems involving specialized applications of computers. For a period of 10 years, he has worked closely with the U.S. Air Force Rome Laboratories (formerly Rome Air Development Center) on problems involving the interaction of computers, communication networks, information and control. His research interests most recently have focused on distributed AI problem solvers and the application of artificial intelligence techniques in communications network management.