DARES: A Distributed Automated REasoning System *

S. E. Conry and D. J. MacIntosh† and R. A. Meyer
Clarkson University
Potsdam, NY 13699
conry @ sun.soe.clarkson.edu

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 this paper, we present DARES, a distributed reasoning system in which agents have the ability to focus their attention on selective information interchange to facilitate cooperative problem solving. The experimental results we present 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 which is largely independent of initial knowledge distribution.

Introduction

In this paper we present DARES, the Distributed Automated REasoning System we have used to investigate the role of knowledge in distributed problem solving. Automated reasoning has been used successfully in a wide variety of applications since Robinson's 1965 paper [8]. Automated reasoning remains an area of active research, with interest focused on such areas as development of more efficient strategies [10] and parallel architectures designed to perform automated reasoning in near linear time [1, 2]. 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 which 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 which 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.

When a reasoning task is assigned to a group of agents, each agent begins working towards 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 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 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 it to perform automated reasoning about distributed domains that other systems have not yet achieved.

*This work was supported, in part, by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

†Current address: BP America, Cleveland, OH 44128

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: (i) a request is sent to one or more agents for information; (ii) 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.

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.

Figure 1 is a flow diagram which depicts a high level view of our approach to distributed theorem proving. DARES is based on a traditional saturation level type theorem prover.

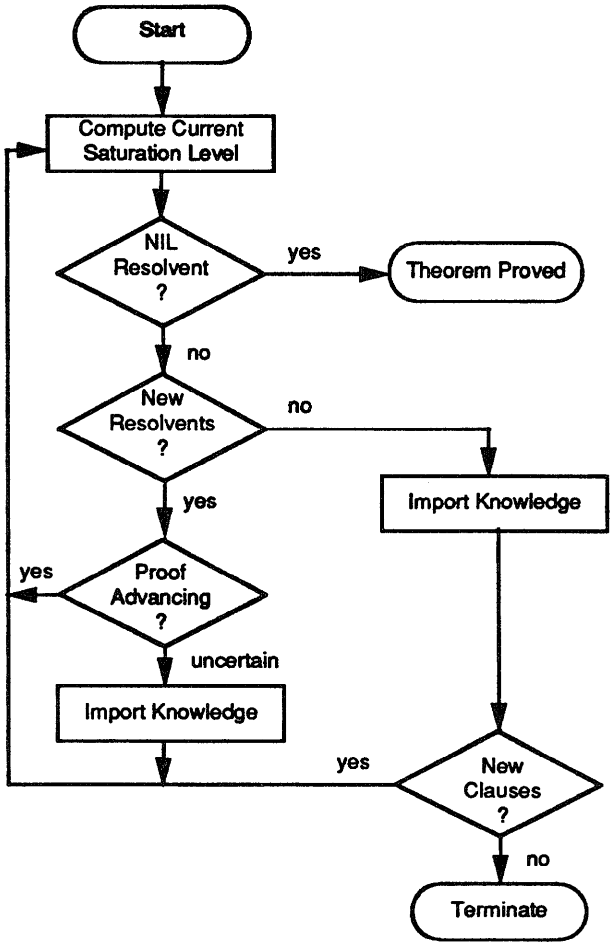


Figure 1: Distributed automated reasoning flow diagram.

We have previously 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 triggers a theorem prover agent to attempt to import relevant information from other agents as shown in Figure 1. If an agent is successful in importing new knowledge, the resolution process continues. Otherwise the distributed theorem proving pro-

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

cess 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 Figure 1, our system must evaluate whether or not progress is being made locally towards a solution. The *Forward Progress* test is made at the conclusion of each resolution level to heuristically determine whether the proof has advanced towards 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.

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 which 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 which 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 are recognized by the *Unit Preference* [11] 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 Figure 1), since a scenario involving zero predicate symbols can only occur if the current level of resolution fails to generate new resolvents.)

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 $P = \{C_1, ..., C_n\}$ where each C_i for $0 < i \leq n$ is a clause heuristically determined to have a high likelihood of furthering the proof towards 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* [12] and *Unit Preference* [11]. 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 towards a NIL resolvent. This heuristic is based on clause length and clause ancestry. Clauses whose ancestry 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 $C_i = \{L_{i1}, ..., L_{im_i}\}$ 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 $C_1, ..., C_n$ and has the form $L = C_1 \cup ... \cup C_n$.

Definition Given L, the Priority Literal Set for $P = \{C_1, ..., C_n\}$, 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 $L_{min} = \{Q_1, ..., Q_n\}$ 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 $NL_{min} = \{R_1, ..., R_n\}$, 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 which 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 [3, 4, 5, 9]), although DARES makes no specific use of this information at this time.

Experimental Results

There are three key issues which 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 which result from particular knowledge distributions minimized, so that DARES' automated reasoning behavior is not misconstrued?
3. What affect does the amount of shared knowledge throughout the network have on system performance?

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:

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

where	k	=	number of active agents
	N	=	number of relevant axioms for the reasoning task
	C	=	$\frac{N}{k}$
	m_i	=	number of local axioms known to agent i
	R	=	is the Redundancy Factor

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.

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 tc 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.

We minimize the effects that a particular knowledge distribution has on general behavior, by performing many 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 many 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 experiment as a vehicle for demonstrating DARES' distributed automated reasoning behavior.

The performance characteristics for a typical experiment are given in Figure 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 in Figure 2, respectively. Since DARES performs distributed automated reasoning approximately 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 among the agent's 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.

We observe that 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. We have also observed that 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, we find that 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 Figure 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 towards the single agent case becomes steeper and the width of these peaks decreases.

Figure 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. What we see happening as network size becomes larger, is that 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, we see larger networks outperforming smaller ones, since requested information near the end of the proof is more readily available in larger networks due to 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 Figure 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 Figures 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 Figure 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 Figure 2. However, the most notable feature of this surface is that it is very *spiked.* This is a demonstration of how sensitive distributed automated reasoning is to its knowledge distribution. Each one of the peaks (spikes) represents reduced system performance 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 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*

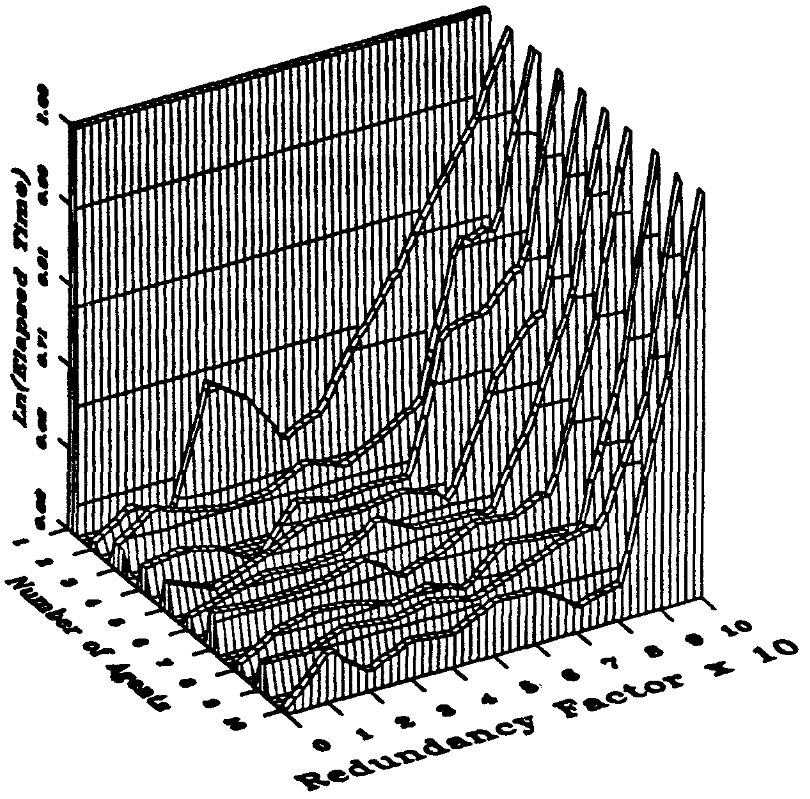
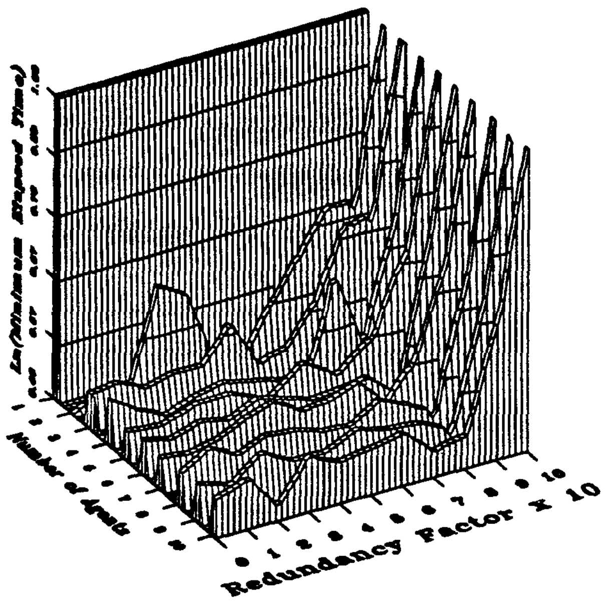


Figure 2: Performance Characteristics for an Experiment.



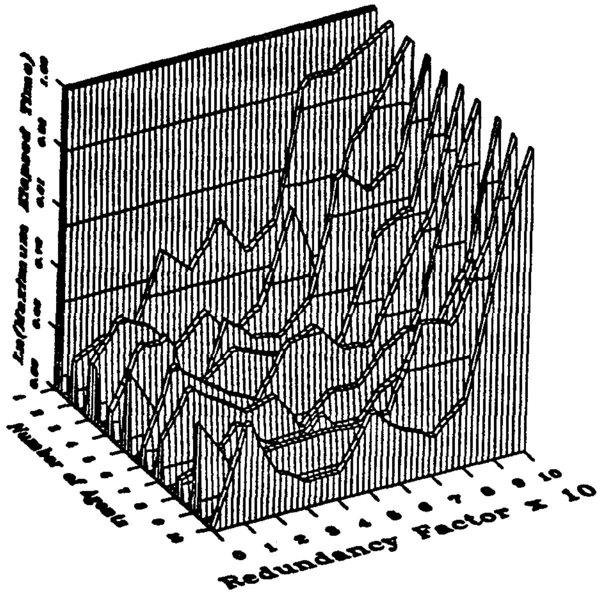


Figure 3: Lower Bound Time Characteristics

Figure 4: Upper bound time characteristics

can affect system performance. But 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.

Concluding Remarks

In this paper, we have described DARES, our distributed automated reasoning system. 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 which is largely independent of initial knowledge distribution.

DARES has been implemented in a distributed testbed facility, SIMULACT [6], 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. A number of additional experiments have been performed using DARES [7], but discussion of these experiments is beyond the scope of this paper.

References

[1] P. E. Allen, S. Bose, E. M. Clarke, and S. Michaylov. PARTHENON: A parallel theorem prover for non-horn clauses. In *9th International Conference on Automated Deduction*, pages 764–765. Springer-Verlag, May 1988.

[2] M. P. Cline. *A Fast Parallel Algorithm for N-ary Unification with A.I. Applications.* PhD thesis, Clarkson University, Potsdam, NY 13676, April 1989.

[3] M. S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):70–80, January 1981.

[4] W. A. Kornfeld and C. E. Hewitt. The scientific community metaphor. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):24–33, January 1981.

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

[6] D. J. MacIntosh and S. E. Conry. SIMULACT: A generic tool for simulating distributed systems. In *Proceedings of the Eastern Simulation Conference*, pages 18–23, Orlando, Florida, April 1987. The Society for Computer Simulation. (also available as NAIC Technical Report TR-8713).

[7] D. J. MacIntosh. *Distributed Automated Reasoning: The Role of Knowledge in Distributed Problem Solving.* PhD thesis, Clarkson University, Potsdam, NY 13699, December 1989.

[8] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, January 1965.

[9] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):61–70, January 1981.

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

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

[12] L. Wos and G. A. Robinson. Paramodulation and set of support. In *Proceedings of the IRIA Symposium on Automatic Demonstration*, pages 267–310. Springer-Verlag, 1968.