Group Decision and Negotiation, 2:301–317 (1993)
© 1993 Kluwer Academic Publishers

Search Reduction in Hierarchical Distributed Problem Solving

THOMAS A. MONTGOMERY
monty@vulcan.srl.ford.com
Ford Motor Company, Scientific Research Laboratory, P.O. Box 2053, MD #2036, Dearborn, MI 48121-2053

EDMUND H. DURFEE
durfee@caen.engin.umich.edu
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109

Abstract

Knoblock and Korf have determined that abstraction can reduce search at a single agent from exponential to linear complexity (Knoblock 1991; Korf 1987). We extend their results by showing how concurrent problem solving among multiple agents using abstraction can further reduce search to logarithmic complexity. We empirically validate our formal analysis by showing that it correctly predicts performance for the Towers of Hanoi problem (which meets all of the assumptions of the analysis). Furthermore, a powerful form of abstraction for large multiagent systems is to group agents into teams, and teams of agents into larger teams, to form an organizational pyramid. We apply our analysis to such an organization of agents and demonstrate the results in a delivery task domain. Our predictions about abstraction's benefits can also be met in this more realistic domain, even though assumptions made in our analysis are violated. Our analytical results thus hold the promise for explaining in general terms many experimental observations made in specific distributed AI systems, and we demonstrate this ability with examples from prior research.

Key words: abstraction, coordination, distributed artificial intelligence, multiagent systems, planning, search

1. Introduction

Problem solving performed by an individual artificial intelligence (AI) system can be characterized as search through a problem space; and, not surprisingly, distributed problem solving (performed by multiple artificially intelligent agents) can be characterized as a distributed search (Durfee and Montgomery 1991). Therefore, when attempting to improve performance in a distributed problem-solving

This research has been sponsored, in part, by the National Science Foundation under grants IRI-9015423 and IRI-9010645, by the University of Michigan Rackham Graduate School, and by a Bell Northern Research Postgraduate Award.

system, it is natural to see how recent results in single agent search can be applied to such multiagent systems. Recent results by Knoblock and Korf (Knoblock 1991; Korf 1987) show that abstraction can reduce search from exponential to linear complexity by dividing a large problem into a number of smaller problems. We extend these results in section 2 by showing that if these smaller problems are distributed to different agents to be solved in parallel, then the time to solution can be further reduced to logarithmic complexity.

This analysis can be applied to both task-level problem solving and meta-level problem solving. At the task level, section 3 presents experimental results from the Towers of Hanoi. Meta-level control was not the primary concern here as task-level problems were distributed for parallel execution using a static allocation scheme. Section 4 describes experiments from a robotic delivery task in which robots must coordinate their activities to avoid collisions. Here we apply hierarchical abstraction to the meta-level problem of coordination, as opposed to the task-level problem of path planning. An abstraction hierarchy in the delivery task can be built by grouping the robots into teams, then grouping the teams into teams, and so on. Then instead of *all* of the agents negotiating directly with *all* of the other agents, agents need only negotiate directly with the members of their team; team leaders handle negotiation between the teams. Thus, much as abstracting and distributing a task-level problem allows the subproblems to be solved in parallel, grouping agents into teams allows coordination to proceed in parallel.

In this article, we show the best-case potential for scaling up distributed artificial intelligence (DAI) systems to large numbers of agents; in the Towers of Hanoi the time to solution only grows as the log of the problem size. We also present analytical and experimental results in which this large complexity reduction is not possible. The delivery task does not meet many of the assumptions of the analysis, yet abstraction can still be useful. When we relax the assumptions even further, the use of abstraction becomes a hindrance by incurring useless overhead. The analysis also allows us to get a handle on the utilization we can expect, and we relate this back to domains such as the Distributed Vehicle Monitoring Testbed (DVMT) (Corkill and Lesser 1983; Durfee, Lesser, and Corkill 1987). In DAI (and in organization theory), the intuition is that solving problems cooperatively can significantly decrease problem-solving complexity in many, *but not all,* situations. ations. While other research has qualitatively verified this intuition, it has not allowed a systematic, domain-independent characterization of the situations in which cooperative problem solving is appropriate. Our work begins to *formally* quantify and *empirically* validate the above informal intuition.

2. Search reduction analysis

To analyze search reduction in hierarchical distributed problem solving, we must first define our terms. *Problem solving* is traditionally viewed as search through

a problem space defined by a set of *states* that the world might be in and a set of *operators* connecting the states. Problem solving is performed by searching for a series of operators that lead from the initial state to one of a set of allowable goal states. *Distributed problem solving* is characterized by the fact that more than one agent (e.g., processor) is involved in the search. Thus, to find a solution, each agent performs some search locally on a portion of the search space and may communicate with other agents to direct their search efforts toward mutually beneficial areas. (Agents are considered to be *distributed* if communication between them is not instantaneous.)

Hierarchical problem solving relies on a hierarchy of abstract problem spaces to focus the search process. Search proceeds by finding a solution to the problem in the most abstract space first. The skeletal plan produced is then fleshed out by successive problem solving in the more detailed problem spaces (in which operators are inserted between the steps in the skeletal plan, or the abstract operators in the skeletal plan are replaced by more detailed operators). The hope is that this iterative refinement will provide direction that successfully prunes the search in the more detailed spaces. In practice, however, it may be necessary to backtrack up the hierarchy and reject some or all of the skeletal plan.

Hierarchical distributed problem solving then combines the features of hierarchical and distributed problem solving—multiple, distributed agents solving a problem by searching through a hierarchy of problem spaces (Durfee and Montgomery 1991; Parunak 1992).

2.1. Single-level abstraction hierarchy

Knoblock showed that a single level of abstraction can reduce an exponential search $O(b^n)$ to $O(\sqrt{n}b^{\sqrt{n}})$. Like ours, his derivation assumes the problem to be solved has a solution of length n (n operators must be applied to get from the initial state to a goal state), and a search complexity that is a function of n. Knoblock also assumes an exponential search with branching factor b, giving a worst case complexity of $O(b^n)$. In our derivation, we generalize his results to an arbitrary function for the complexity, $f(n)$.

If we first solve an abstract problem, dividing the overall problem into k subproblems that must be solved serially, then the complexity becomes the sum of the complexity of solving the abstract problem plus the complexities of solving all of the individual subproblems. Since k is the length of the abstract problem, the total complexity is $f(k) + f(n'_1) + f(n'_2) + \ldots + f(n'_k)$ where n'_i is the length of the ith subproblem. Assuming that the problem can be divided into equal-sized subproblems, and that the number of subproblems can be chosen such that it is equal to the solution length of each subproblem, then $n'_i = k$ for all i, and the complexity becomes $(k + 1)f(k)$. If the length of the final solution with abstraction is the same as the length of the original solution ($k^2 = n$), then $k = \sqrt{n}$, and the complexity is:

$$O((\sqrt{n} + 1)f(\sqrt{n})) = O(\sqrt{n}f(\sqrt{n})) \tag{1}$$

Note that if $f(n) = b^n$, we duplicate Knoblock's result (b^n reduces to $\sqrt{n}b^{\sqrt{n}}$).

If we further assume that each subproblem is solved by a different agent (a different processor or team of agents), and we ignore the cost of assigning subproblems to agents, then the complexity is reduced even further. Instead of the subproblems' complexities adding when they are being solved serially, only the longest subproblem contributes to the overall time complexity when they are solved in parallel. Making similar assumptions as above, the time complexity becomes:

$$O(f(k) + max(f(n'_1), f(n'_2), \ldots, f(n'_k))) = O(2f(\sqrt{n})) = O(f(\sqrt{n})) \tag{2}$$

Thus, given concurrency, an $O(b^n)$ search can ideally be reduced to $O(b^{\sqrt{n}})$. As a quick sanity check of this result, note that for the k subproblems to be solved in parallel, there must be k agents working on them. Since $k = \sqrt{n}$, the decrease in complexity between expressions (1) and (2) is as expected: $\sqrt{n}$.

2.2. Multilevel abstraction hierarchy

Now we generalize to l levels of abstraction. Starting with the ground space and moving up the abstraction hierarchy, the first abstraction level divides the initial problem of length n into $\frac{n}{k}$ subproblems of length k. At each abstraction level above, k subproblems are abstracted into one subproblem of length k, until the top level is reached, at which point there is one problem left of length $\frac{n}{k^{l-1}}$. If each problem is solved serially, then the overall complexity is:

$$f\left(\frac{n}{k^{l-1}}\right) + \frac{n}{k^{l-1}}f(k) + \frac{n}{k^{l-2}}f(k) + \ldots + \frac{n}{k}f(k). \tag{3}$$

If we set $l = \log_k n$, then expression (3) reduces to $(1 + k + k^2 + \ldots + k^{l-1})f(k)$ which can be simplified as:

$$O\left(\frac{k^l - 1}{k - 1}f(k)\right) = O\left(\frac{k^{\log_k n} - 1}{k - 1}f(k)\right) = O\left(\frac{n - 1}{k - 1}f(k)\right) = O(n). \tag{4}$$

The final step in expression (4) is possible since k is assumed constant for a given problem space and abstraction hierarchy, while l must grow with the problem size. Thus, given a search of arbitrary complexity in the length of the solution, $O(f(n))$,

multiple levels of abstraction can reduce the complexity to a linear number of constant terms. In particular, we get Knoblock's result of reducing an exponential search to linear complexity.

If we assume that each subproblem is solved by a separate agent (processor), then after the most abstract problem is solved, its $\frac{n}{k^{l-1}}$ subproblems can be solved in parallel, after which the next $\frac{n}{k^{l-2}}$ subsubproblems can be solved in parallel, and so on. In effect, all the coefficients in expression (3) are eliminated, giving:

$$f\left(\frac{n}{k^{l-1}}\right) + (l - 1)f(k). \tag{5}$$

Substituting $l = \log_k n$, this reduces to $(\log_k n)f(k)$ which is $O(\log_k n)$. Therefore, concurrent processing can potentially reduce an exponential search problem not just to linear complexity but to logarithmic.

2.3. Assumptions of the analysis

Our analysis is based on the following assumptions [as taken verbatim from Knoblock (1991) with our comments in brackets]:

1. *The number of abstraction levels is* $\log_k$ *of the solution length.* Thus, the number of abstraction levels must increase with the size of the problems.
2. *The ratio between levels is the base of the logarithm, k.*
3. *The problem is decomposed into subproblems that are all of equal size.* If all the other assumptions hold, the complexity of the search will be the complexity of the largest subproblem in the search.
4. *The hierarchical planner produces the shortest solution.* The analysis holds as long as the length of the final solution is linear in the length of the optimal solution. [Here, optimal solution means the solution that would be found without hierarchical planning.]
5. *There is only backtracking within a subproblem.* This requires that a problem can be decomposed such that there is no backtracking across abstraction levels or across subproblems within an abstraction level. [In other words, the downward refinement property (Bacchus and Yang 1991) must hold.]

In addition, our extensions to the analysis require that:

1. *There are at least as many agents as there are bottom (most detailed) level subproblems (i.e., there are* $\frac{n}{k}$ *or, equivalently* k^{l-1} *agents).* We expect one agent to solve the most abstract problem, ship all but one of the subproblems to other

agents, and then begin working on the one subproblem that it kept. Agents receiving subproblems repeat this process until the bottom level is reached. This implies that one agent is kept busy throughout the entire problem-solving cycle, while some number of agents, $k^{l-1} - k^{l-2}$, are not enlisted until the most detailed subproblems are reached. Thus, even though larger problems require larger numbers of agents, the average load on the agents actually decreases with increased problem size.

2. *The process of distributing subproblems does not increase the time complexity.* In general, determining which agents should work on which subproblems is a nontrivial problem. In the DAI literature this has been referred to as the connection problem (Davis and Smith 1983), while in operating systems it can be seen as a variant of load-balancing. However, if we assume that the amount of time an agent takes to decide who to send k subproblems to is a function of k, and that sending messages is time bounded, then this assumption is met. Essentially the derivation proceeds as above, but now $f(k)$ includes the maximum message delay and the time to decide where to ship subproblems.

3. *Collecting results from subproblems does not contribute to the problem complexity.* If we again assume a time-bounded message delay and that the amount of time to assemble the results of k subproblems is a function of k, then this assumption is met. In terms of our analysis, the process of collecting results would just be the reverse of solving and distributing subproblems, benefiting equally from parallelism.

Therefore, our analysis predicts that it is possible to achieve logarithmic time complexity by using an abstraction hierarchy and multiple agents, but this requires a large number of assumptions to be met. In the next section we describe experiments in the Towers of Hanoi in which *all* of these assumptions of *are* met. Then in section 4 we begin to relax some of these assumptions, presenting both analytical results and experimental results (from the delivery task) in which *many* of these assumptions *are not* met. In both cases, it is clear that the use of abstraction can benefit distributed problem solving in much the same way that it benefits single-agent problem solving.

3. Search reduction in ideal case

In order to verify empirically the formal analysis in the previous section, we ran a series of experiments on the Towers of Hanoi problem. The abstraction hierarchy of the Towers of Hanoi meets all of the assumptions of the analysis. It is created by ignoring the smallest disk at the first level of abstraction, then the smallest remaining disk at successive levels, until ultimately only the largest disk remains at the most abstract level. Problem solving proceeds by solving the problem at the most abstract level, then creating subproblems at successively more detailed levels of the hierarchy in which the goals of the subproblems are to (1)

achieve the preconditions of the operators in the current abstract plan and (2) reach the final goal state.

3.1. Solution time

We ran our experiments on a TI Explorer II and recorded average CPU times to complete the task. For the distributed (multiagent) version, we simulated concurrent execution of the subproblems by implementing the agents in MICE (Montgomery and Durfee 1990), charging real time for agent reasoning, and allowing any number of messages to be sent by different agents at one time. Figure 1 presents the elapsed time to solve problems of various sizes[1] for:

(x) *A single agent without abstraction.* In this case the time to solve the problem grows exponentially with the size of the problem.
(o) *A single agent with abstraction.* Duplicating Knoblock's result, the time for a single agent using hierarchical abstraction to solve the problem grows linearly with the size of the solution.
(*) *Multiple agents with abstraction.* Here the benefits of parallel execution of the subproblems reduce the elapsed time to logarithmic complexity in the solution length.

Thus, the results presented in Figure 1 verify the predictions of the analysis. Figure 1 also shows the sum of the CPU time used by the agents in the distributed version (+). As expected, this line has a greater slope than the single agent case (o) due to the overhead of sending messages and keeping track of the other agents; however, it is also still linear in the length of the solution.

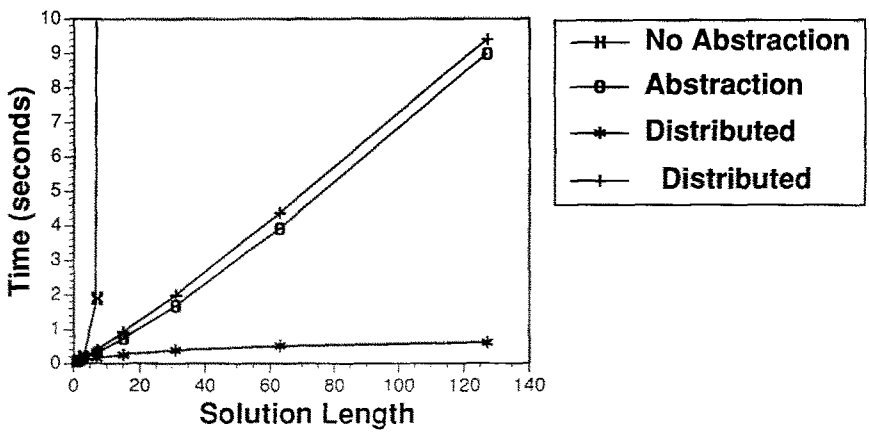

Figure 1. Elapsed time to solve the Towers of Hanoi problem.

3.2. *Utilization*

Figure 2 displays the average CPU utilization of the agents in the distributed version. Since the number of agents required to solve the problem grows linearly with the problem size ($\frac{n}{k}$ agents are required), but the elapsed time only grows logarithmically, we see that the average CPU utilization actually decreases with increased problem size. This conforms to the predictions of the analysis in which the expected average utilization is:

$$\frac{k(n-1)}{(k-1)nl} = \frac{k(n-1)}{(k-1)n(\log_k n)}. \tag{6}$$

(There are $\frac{n}{k}$ agents available for $lf(k)$ time units, giving $\frac{n}{k}lf(k)$ available CPU cycles, of which $(1 + k + k^2 + \ldots + k^{l-1})f(k) = \frac{k^l - 1}{k - 1}f(k) = \frac{n - 1}{k - 1}f(k)$ are used.) This decreasing utilization implies that abstraction would be well suited to solving problems in computer networks in which there are large numbers of computers available, each with a small amount of excess CPU cycles. The analysis also implies that reasonably good speedups can be achieved with dramatically fewer agents. For example, if we reduce the number of agents by a factor of k, then we add $(k - 1)f(k)$ to the complexity of expression (5). However, since k is constant, this represents only a constant factor decrease in performance. Ultimately, however, if the number of agents are dramatically reduced, the performance will approach the single-agent case of linear time complexity.

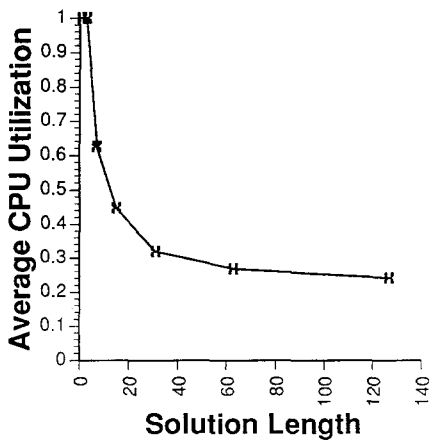

Figure 2. Average CPU utilization in distributed Towers of Hanoi.

If a separate agent were used for every node in the abstraction hierarchy, for a total of $\frac{n-1}{k-1}$ agents, then the utilization of the agents would become $\frac{1}{l} = \frac{1}{\log_k n}$ (because each agent is only busy during the time that problem solving on its level is taking place). However, if there is a continuous supply of problems, all of the agents can be kept busy most of the time. After the top agent solves the most abstract problem and ships the subproblems off to other agents, it can start working on the next problem. Eventually the entire "pipeline" (hierarchy of agents) will be filled.

The next question to ask, then, is how much processing power is lost waiting for the pipeline to fill. All $\frac{n-1}{k-1}$ agents are busy after time $(l-1)f(k)$; therefore, there are $\frac{(n-1)(l-1)}{k-1}f(k)$ total available CPU cycles while the pipeline is filling. When propagating top-down, there are $k + k^2 + \ldots + k^{l-1}$ agents idle during the first $f(k)$ time units, $k^2 + k^3 + \ldots + k^{l-1}$ during the second, and so on. This gives the total CPU cycles lost as $(k + 2k^2 + \ldots + (l-1)k^{l-1})f(k)$, which can be rewritten as $\left(\frac{(l-1)k^l}{k-1} - \frac{k(k^{l-1}-1)}{(k-1)^2}\right)f(k)$. Dividing the lost cycles by the total available gives the fraction of available CPU cycles wasted while waiting for processing to propagate from the top agent down to the leaves as:

$$\frac{k - n + (k-1)n(l-1)}{(k-1)(n-1)(l-1)} = \frac{k - n + (k-1)n(\log_k n - 1)}{(k-1)(n-1)(\log_k n - 1)}. \tag{7}$$

Of course, this pipeline can be filled top-down or bottom-up—top-down processing can be viewed as a contracting/subcontracting process (Davis and Smith 1983), while bottom-up processing can be viewed as data-driven hypothesis generation (Corkill and Lesser 1983). In the case that processing proceeds bottom-up, the fraction of available processing power unused while filling the hierarchy is just one minus the quantity in expression (7). Because of the symmetry between top-down and bottom-up processing, the ratio in expression (7) is also the processor *utilization* when processing propagates bottom-up, and one minus expression (7) is the utilization for top-down.

This analysis can help to explain analytically some of the empirical results previously reported in the literature. In Corkill and Lesser's Distributed Vehicle Monitoring Testbed (DVMT) (Corkill and Lesser 1983), hypotheses are generated bottom-up, so it takes some time before the agents integrating high-level hypotheses have data on which to work. While Corkill and Lesser note that processing is lost "as nodes wait for something to do," we can now set a lower bound on the amount of processing that must be lost waiting. For example, if we have a three-level hierarchy with $k = 2$ and $n = 8$, and if processing proceeds bottom-

up, then $\frac{2}{7}$ of the available processing power is lost by agents waiting for something to do.

Predicting the optimal performance of a system of agents is an important step in evaluating alternative approaches to coordination, and our analysis can be used to make such predictions. For example, in Durfee, Lesser, and Corkill (1987) DVMT researchers calculated the optimal improvement in processing time due to multiple agents by hand-simulating the shortest data path through the hierarchy. Their hand simulations, which included implementation specific details due to knowledge source interactions and communication delays, showed that p agents could never achieve a factor of p speedup.[2] Our analysis can be used to generate similar results without going to the trouble of hand simulations. In the DVMT, the optimal processing time for a vehicle track is achieved when there is one agent for every vehicle location in the track. Processing proceeds in a bottom-up fashion by creating vehicle location hypotheses and then combining them into longer and longer track hypotheses. A track of length 8 amounts to a 4 level problem with a branching factor of 2. According to expression (6), the average utilization of those 8 agents would be $\frac{15}{32}$, implying that the best possible speedup is no more than $\frac{15}{32} * 8$, or less than 4 times.

3.3. Discussion

So far we have seen that combining parallelism and hierarchical problem solving can be very powerful: in well-behaved problems like the Towers of Hanoi, it can reduce exponential search to logarithmic time complexity. Furthermore, this exponential improvement in time complexity is paid for by only a linear increase in the number of agents (even when a separate agent is desired at each node in the hierarchy, $\frac{n-1}{k-1}$ instead of $\frac{n}{k}$ agents are required—still linear in the size of the problem). Given a continuous stream of problems and an agent at each node in the hierarchy, we can even approach full utilization of the agents (processors) involved. Unfortunately, real-world problem domains are not likely to meet all of the necessary assumptions to achieve this best-case performance. Therefore, in the next section we show the application of abstraction to a more realistic domain: the delivery task.

4. Search reduction in realistic case

In this section we look at the effects of relaxing some of the assumptions of the analysis. First we see analytically how a fixed number of levels of abstraction

affect our results, then we see empirically how abstraction performs in the delivery task domain.

4.1. Fixed number of levels

For some problem domains it is easy to meet the assumption that the number of levels of the hierarchy, l, increases with the size of the problem. In the Towers of Hanoi, this is done by having a separate level of abstraction for each disk. In many more realistic problem domains, however, the number of levels of abstraction may be fixed. For example, our work in the delivery task has only a single level of abstraction (agents grouped into teams).

Here we draw on others' results (Bacchus and Yang 1992) to illustrate the effect on our analysis of having a fixed number of abstraction levels. In such a case, the ratio between levels of the hierarchy, k, is no longer constant since it must now grow with the length of the solution, n, according to $k = \sqrt[l]{n}$. Thus, for l levels of abstraction in which all subproblems are solved by a single agent, expression (4) becomes $O\left(\frac{n-1}{\sqrt[l]{n}-1} f(\sqrt[l]{n})\right) = O(nf(\sqrt[l]{n}))$, which gives the result reported by Bacchus of $O(nb^{\sqrt[l]{n}})$ when $f(n) = b^n$.

When the subproblems are solved in parallel and the number of levels are fixed, expression (5) becomes $O(lf(\sqrt[l]{n})) = O(f(\sqrt[l]{n}))$. Therefore, given a fixed number of levels, abstraction no longer reduces an exponential search to linear complexity for a single agent and logarithmic complexity for multiple agents. Instead, it reduces the complexity to a more slowly growing exponential with a further factor of n speedup possible with parallelism. Thus, when the assumption of a variable number of levels of abstraction is violated, the full performance improvement to logarithmic time cannot be realized. Nonetheless abstraction may still be useful, as we shall see.

4.2. The delivery task

We now turn to a delivery task domain in which a number of robots must deliver packages to specified locations without colliding. This domain captures many of the coordination issues that are present in air traffic control, intelligent vehicle highway systems, cooperative robotic reconnaissance, and factory floor control. When such a group of autonomous agents must coordinate their activities, they could do so by individually negotiating with every other agent in the environment, but this results in a large amount of communication when large numbers of agents are involved (for example, the individual negotiations among 20 agents depicted in Figure 3). Alternatively, they can group themselves into teams of agents, have team leaders negotiate between teams, and only individually negotiate within teams (for example, the team negotiation pattern in Figure 3). When even more

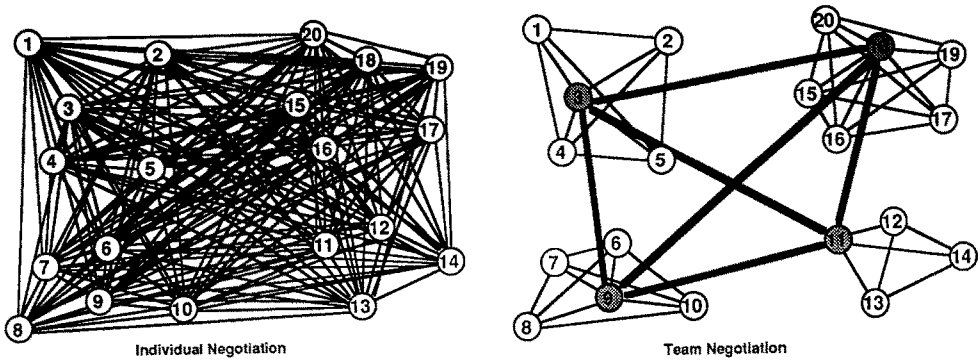

Figure 3. Individual vs. team negotiation patterns.

agents are involved, this abstraction into teams can be extended to an arbitrary number of levels by creating teams of teams, teams of teams of teams, and so on.

If negotiation in a hierarchy of teams starts at the top (most abstract) level and works its way down to individuals negotiating with their fellow team members, then the analysis presented in section 2 can be directly applied to the time taken to complete all negotiations. The search complexity $f(n)$ would describe the complexity of searching for nonconflicting behaviors (i.e., negotiating), and n would relate to the number of agents involved. For example, elsewhere (Montgomery and Durfee 1992) we analyzed our hierarchical communication protocol (Durfee and Montgomery 1990) and found it to have $O(n^3)$ complexity where n is the number of agents.[3] Grouping agents into teams (a single-level hierarchy) has the potential to reduce such an $O(n^3)$ protocol to $O(n^2)$ [by expression (1)], and if the teams can negotiate in parallel, then the time complexity is reduced to $O(n^{1.5})$ [by expression (2)]. If the agents are grouped into a hierarchy of teams, then ultimately the time complexity of the negotiation can be reduced to logarithmic time.

One way to realize such complexity reductions is to have the highest team leaders negotiate first, then the subteam leaders negotiate with each other, and so on. Alternatively, negotiation can start at the bottom level (individuals negotiating with their fellow team members) and work its way up. Negotiation in either direction (top-down or bottom-up) can meet the assumptions of the analysis, but in reality it may be necessary to employ elements of both: negotiating *bottom-up* gives team leaders the chance to accumulate information about what their team members are planning to do; however, since ultimately it is individuals that execute plans, the decisions made by team leaders must flow back *down* to the individuals.

4.3. Experiments

Our implementation of agents in the delivery task has both bottom-up and top-down elements. Agents negotiate with their fellow team members first; this gives

the team leaders sufficient knowledge to begin team-level negotiations. During the course of team-level negotiations, the leaders may need to request further details from their members. After team-level negotiations finish, the leaders instruct their members on how they need to adapt their plans to avoid conflict with the other teams. Therefore, since there could be significant communication up and down the hierarchy, an assumption—that the costs of distributing subproblems and collecting the results are negligible—may be violated.

The formal analysis also assumes that subproblems are of equal size. However, since we generate delivery problems randomly, different teams may face coordination problems with very unequal levels of difficulty. It is also not clear whether the implementation meets the assumption that linearly comparable solutions are reached with and without abstraction. We get different plans for delivering the same packages depending on whether or not teams are used (the presence of teams can change both the assignments of packages to agents, and the order in which conflicts are identified and resolved). However, in spite of the fact that many of the assumptions of the analysis are not strictly met in this domain, we show below that abstraction by grouping agents into teams still holds promise. It appears that the analysis is tolerant of violations of some of the underlying assumptions and thus can be used in developing domain-independent explanations for phenomena seen in specific DAI systems (as in section 3.2).

In our experiments, we grouped the agents into teams that were restricted to working in one quadrant of the overall environment. These quadrants overlapped at the edges so that, in delivering packages that spanned multiple teams' regions, the agents from one team could drop packages off at the boundary for agents from another team to pick up (see Figure 4). This also made it necessary for the teams to negotiate with each other to avoid colliding in the boundary regions. Packages

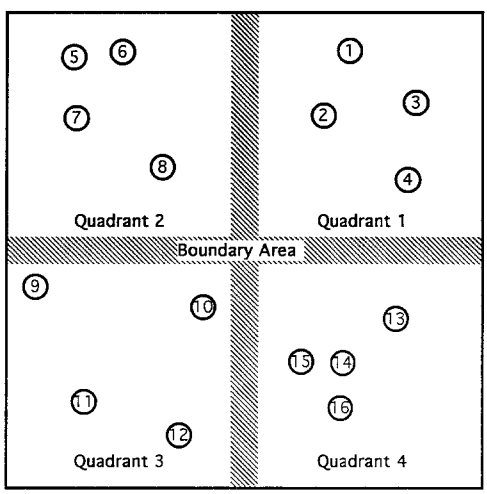

Figure 4. An example random environment.

were distributed randomly in the environment, with randomly chosen destination locations.

In the first set of experiments, we distributed the packages evenly among the quadrants and restricted a package's destination to be within the same quadrant as its initial location. Thus, the division of agents into teams matched the coordination problem precisely, causing no backtracking between subproblems (dropping packages off for another team to deliver). Figure 5 shows the average amount of time for agents that negotiate as individuals to deliver a set of packages versus the time for the same number of agents grouped into four teams to deliver the same packages. The number of packages to be delivered is equal to the number of agents; thus, each agent has one delivery to make. We see that when (1) the hierarchy is well matched to the problem, (2) there is no need to "backtrack" at a given level of the hierarchy, and (3) the subproblems are of approximately equal size, then grouping agents into teams shows a definite benefit.

If we relax our assumptions further by not restricting package destinations to their originating quadrants, then we get very different behavior. Now, as shown in Figure 6, the total time to coordinate activities and deliver the packages is initially worse for the teams of agents than for the individuals. This is primarily because the teams of agents must first drop packages off at their boundary, and then renegotiate to complete the deliveries of packages dropped off in their respective regions—essentially, the teams are violating the assumption of no backtracking. In contrast, the individuals only negotiate once and then deliver their packages all the way to their destinations. Eventually, however, as the number of agents is increased, the combinatorics of the negotiation process overwhelms the individuals and the teams show an advantage again.

5. Conclusion

Abstraction has long been recognized as capable of improving the efficiency of search (Newell and Simon 1972; Sacerdoti 1973; Stefik 1981); and recently, detailed complexity analyses of this ability have been presented (Bacchus and Yang 1992; Knoblock 1991; Knoblock, Tenenberg, and Yang 1991; Korf 1987). We have extended this research to multiagent systems, showing both analytically and empirically that search can be reduced to logarithmic time complexity using hierarchical distributed problem solving. In presenting this ideal case, we have enumerated a sufficient set of conditions that allow it to be achieved. We have also shown how abstraction can be applied to both task-level and meta-level problem solving through our experiments in the Towers of Hanoi and delivery task problem domains.

Perhaps more importantly, however, the perspective used in this article of viewing coordination as a distributed search (Durfee and Montgomery 1991) sheds light on some of the fundamental problems of DAI. In particular, DAI is often concerned with achieving coherence and coordination among a set of distributed agents. Because of the interdependency between agents that this implies, it is very

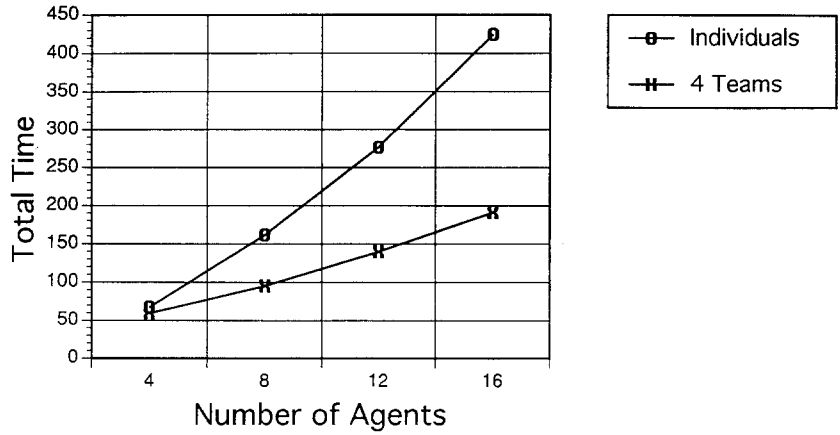

Figure 5. Total time for individuals and teams to complete bounded deliveries.

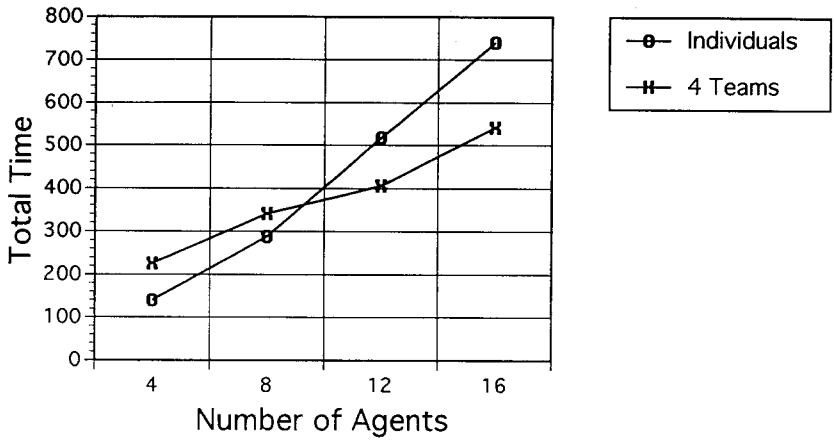

Figure 6. Total time for individuals and teams to complete unbounded deliveries.

difficult to find the nearly independent subproblems that are required to achieve the full complexity reduction possible. However, assuming that the problems can be decomposed into nearly independent subproblems, then our analysis also shows what can be achieved in various circumstances. For example, if the number of levels of the hierarchy is fixed, then the best complexity reduction we can hope for is from $O(f(n))$ to $O(f(\sqrt[k]{n}))$. Thus if a single agent requires exponential search to solve a problem without abstraction, then a set of cooperating agents using abstraction will still require exponential time, though it will be a much slower growing exponential.

We have also shed some light on past research in DAI, such as showing analytically the minimum processing power lost waiting for data to reach a certain level in the hierarchy. This helps explain, for instance, why p processors could not

achieve a factor of p improvement in the DVMT (Corkill and Lesser 1983; Durfee, Lesser, and Corkill 1987). The analysis can also be applied to systems using the contract net protocol (Davis and Smith 1983). If contracting is performed as a functional decomposition such that a manager sends all subproblems down to subordinates, then $\frac{n-1}{k-1}$ agents are required, and full utilization can be achieved after the hierarchy fills. However, if contracting is viewed more as load balancing, such that an agent may bid on its own subtask, then only $\frac{n}{k}$ agents are required, but it is no longer clear that full utilization can be realized.

Viewing coordination as a distributed search process has proven to be a very fruitful research direction to take. It has allowed us to adapt research on single-agent problem solving to distributed systems, and given us insights into problems that have long been faced by DAI researchers. We intend to continue this line of research by seeing what further insights on distributed problem solving can be gained. Promising areas include automating the creation and evaluation of hierarchies (Knoblock, Tenenberg, and Yang 1991), and determining the expected value of a hierarchy based on the probability of being able to refine the abstractions (Bacchus and Yang 1992).

Notes

1. Note that the problem size for the Towers of Hanoi grows exponentially with the number of disks in the problem: $n = 2^d - 1$ where d is the number of disks.
2. The difficulty of achieving a factor of p speedup with p agents has long been recognized. Amdahl's law (Amdahl 1967) presents this from the perspective of a serial bottleneck. Holding the problem size constant, no matter how many additional processors are used, a computational task cannot be performed any faster than its serial portion. More recently, however, the perspective taken by Gustafson (1988) shows that by *varying* the problem size with the number of agents, a scaled speedup can be achieved that approaches p for p agents. Gustafson's perspective is used here; it is appropriate since most problems tackled in DAI do scale up or down with the number of agents. For example, the computational task of coordinating the movements of a number of robots scales directly with the number of robots involved (processors available).
3. The intuition behind this complexity is that each agent must negotiate with every other agent (which is $O(n^2)$), plus whenever an agent changes its plans during the course of one of these initial $O(n^2)$ dialogues, it must re-negotiate (doublecheck) those changes with an average of $\frac{n}{2}$ agents.

References

Amdahl, Gene M. (1967). "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities." In *AFIPS Conference Proceedings*. Reston, VA: AFIPS Press, pp. 483–485.

Bacchus, Fahiem, and Qiang Yang. (1991). "The Downward Refinement Property." In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia, Los Altos, CA: Morgan Kaufmann, pp. 286–292.

Bacchus, Fahiem, and Qiang Yang. (1992). "The Expected Value of Hierarchical Problem-Solving." In *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, Boston: AAAI Press, pp. 369–374.

Corkill, Daniel D., and Victor R. Lesser. (1983). "The Use of Meta-level Control for Coordination in a Distributed Problem Solving Network." In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, Federal Public of Germany, Los Altos, CA: Morgan Kaufmann, pp. 748–756. Also appeared in *Computer Architectures for Artificial Intelligence*, Benjamin W. Wah and G.J. Li (eds.), IEEE Computer Society Press, 1986, pp. 507–515.

Davis, Randall, and Reid G. Smith. (1983). "Negotiation as a Metaphor for Distributed Problem Solving." *Artificial Intelligence* 20(1), 63–109. Also appeared in Alan H. Bond and Les Gasser (eds.), *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, 1988, pp. 333–356.

Durfee, Edmund H., and Thomas A. Montgomery. (1990). "A Hierarchical Protocol for Coordinating Multiagent Behaviors." In *Proceedings of the National Conference on Artificial Intelligence*, Boston: AAAI Press, pp. 86–93.

Durfee, Edmund H., and Thomas A. Montgomery. (1991). "Coordination as Distributed Search in a Hierarchical Behavior Space." *IEEE Transactions on Systems, Man, and Cybernetics* 21(6) (Special Issue on Distributed AI).

Durfee, Edmund H., Victor R. Lesser, and Daniel D. Corkill. (1987). "Coherent Cooperation among Communicating Problem Solvers." *IEEE Transactions on Computers* C-36(11), 1275–1291. Also appeared in Alan H. Bond and Les Gasser (eds.), *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, 1988, pp. 268–284.

Gustafson, John L. (1988). "Reevaluating Amdahl's Law." *Communications of the ACM* 31(5), 532–533.

Knoblock, Craig A., Josh D. Tenenberg, and Qiang Yang. (1991). "Characterizing Abstraction Hierarchies for Planning." In *Proceedings of the National Conference on Artificial Intelligence*, Anaheim, CA, July, Boston: AAAI Press.

Knoblock, Craig A. (1991). "Search Reduction in Hierarchical Problem Solving." In *Proceedings of the National Conference on Artificial Intelligence*, Anaheim, CA, July, Boston: AAAI Press.

Korf, Richard E. (1987). "Planning as Search: A Qualitative Approach." *Artificial Intelligence* 33(1), 65–88.

Montgomery, Thomas A., and Edmund H. Durfee. (1990). "Using MICE To Study Intelligent Dynamic Coordination." In *Proceedings of the Tools for Artificial Intelligence Conference*, Washington, DC, November.

Montgomery, Thomas A., and Edmund H. Durfee. (1992). "Search Reduction in Hierarchical Distributed Problem Solving." In *Proceedings of the 1992 Distributed AI Workshop*, Glen Arbor, MI, February.

Newell, A., and H.A. Simon. (1972). *Human Problem Solving*. Prentice Hall.

Parunak, H. Van Dyke. (1992). "How To Describe Behavior Space." In *Proceedings of the 1992 Distributed AI Workshop*, Glen Arbor, MI, pp. 303–316.

Sacerdoti, Earl D. (1973). "Planning in a Hierarchy of Abstraction Spaces." In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann, pp. 412–422.

Stefik, Mark. (1981). "Planning and Meta-Planning (MOLGEN: part 2)." *Artificial Intelligence* 16, 141–170.