Are Many Reactive Agents Better Than a Few Deliberative Ones?

Kevin Knight
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
knight@isi.edu
(310) 822-1511

Abstract

Problem solvers fall along a wide spectrum ranging from highly deliberative to highly reactive. Highly deliberative systems are able to design optimally efficient solutions to problems, but they require complete world models and consume inordinate computational resources. Reactive systems move in real time but cannot guarantee efficient solutions. They are also subject to looping behavior. One way to generate incrementally more efficient solutions is to be incrementally more deliberative, e.g., to increase the amount of mental search between actions. This paper presents an alternative method for generating more efficient solutions: increasing the number of reactive agents simultaneously attacking a given problem. This method provides a second, orthogonal degree of freedom. We find that in many domains, increasing agents is dramatically superior to increasing single-agent deliberativeness. This is because solution quality improves rapidly as more reactive agents are added, but search time only increases linearly. This contrasts with adding more deliberativeness, which incurs exponentially increasing time costs. Ample empirical evidence is presented to support our conclusions.

1 Introduction

This paper considers two aspects of computational problem solving:

(1) search time-how long it takes to come up with a solution.

(2) solution quality—how good that solution is, in terms of resources needed to execute it.

There is an intuitive trade-off between (1) and (2). The longer we think about a problem, the better chance we have of finding a good solution. While search algorithms like A* [Hart *et* al., 1968] strive to limit (1) while optimizing (2), time limitations often force us to settle for suboptimal, or "satisficing" [Simon, 1957], solutions.

Different situations will place different emphases on (1) and (2). Consider the problem of sending an interplanetary probe to Neptune. In this case, it may be worth spending days or weeks to plot an optimal trajectory, since such calculations could save months of travel time. On the other hand, consider the case of Hernan Cortes, the Spanish conqueror of Mexico. While still a teenager in Spain, finding himself on the wrong end of a jealous husband's musket, Cortes immediately devised a plan to travel to the New World. The efficiency of his plan was not critical. What was important was that he get started right away.

This paper studies search time versus solution quality in the context of the Real-Time-A* (RTA*) algorithm devised by [Korf, 1990]. The next section reviews how RTA* interleaves planning and execution, and how this leads to a flexible time/quality trade-off. Subsequent sections introduce new algorithms and empirical results.

2 Real-Time Heuristic Search

Motivated by research on two-player games, [Korf, 1990] investigated single-agent search under the constraints of having to take action within a given time limit and/or having limited information about the environment. Sample single-agent search tasks include robot navigation, the blocks world, and the 8-puzzle (Figure 1). Korf's algorithm, called Real-Time-A* (RTA*), alternates between two phases: plan and execute. During each planning phase, RTA* makes a decision about which action to take, based on the current situation. It then executes the action in the world, and starts planning again. This continues until it reaches its goal. RTA* can vary the amount of planning versus executing it does by changing how deeply it looks into the future during the planning phases. Here is the algorithm:

1. Set variable N to the start state.
2. Generate all of the successor states of N. If any of the successors is the goal state, then move to the goal and quit.
3. Estimate the heuristic value of each successor S by performing a fixed-depth tree search rooted at S.
4. Let S1 be the successor with the best backed-up value. Let V2 be the value of the second-best successor. Take whatever action corresponds in the world to moving to state S1. Store state N in a hash table as a key with value V2. If the state N is ever generated again in step 2, use the value stored in the table instead of performing the search of step 3.

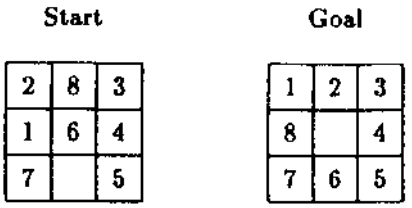

Figure 1: The 8-Puzzle

5. Set N to SI, and go to step 2.

Although RTA* may enter the same state several times, the values of previously visited states (stored in the hash table) prevent RTA* from entering a fixed loop. The depth of the tree search in step 3 determines how much time RTA* spends planning instead of executing actions.

RTA* is useful in both complete- and incomplete-information domains. When information about the world is incomplete, it is impossible to plan out an entire solution ahead of time. In such a case, interleaving planning and execution is necessary. The algorithm's utility in complete-information domains comes because large search spaces impose *practical* limitations to lookahead search. While the optimal solution to a 24-puzzle problem may contain 100 moves, current computers would take months or years to exhaustively search a tree to that depth. RTA* solves such problems by making the move that seems locally best, recording that move in its hash table, and repeating until the goal is reached. No current techniques based on heuristic search can find optimal solutions to the 24-puzzle, yet RTA* returns a solution in seconds. The catch is that the solution is not optimal.

Korf demonstrated that by increasing the lookahead horizon, he could induce RTA* to come up with shorter solutions to the 8-puzzle (using the standard Manhattan distance heuristic function). Figure 2 illustrates this phenomenon. The top curve is the one reported by Korf: it is the actual number of steps "executed" by RTA*. The lower curve represents the number of steps left after we have removed the cycles from the solution path.[1] Of course, if we were using RTA* in a real-time application, we would not be able to remove those cycles—the cost would have already been incurred. For the remainder of this paper, "solution quality" refers to the length of a solution with cycles deleted.

Of course, high quality plans come at a cost. As we increase the lookahead horizon, we produce better moves, but individual moves require more time to contemplate. Figure 3 shows the well-known exponential nature of tree search.[2]

[1]We have found that a slight modification to the RTA* algorithm allows it to delete cycles during the search.

[2]In this and subsequent figures, time means user time, in seconds, of a C implementation of RTA* running on an IBM-PC/RT. Due to the large number of runs, most experiments in this paper were performed on the 8-puzzle rather than larger puzzle sizes, but see Section 5 for results from the 15-puzzle.

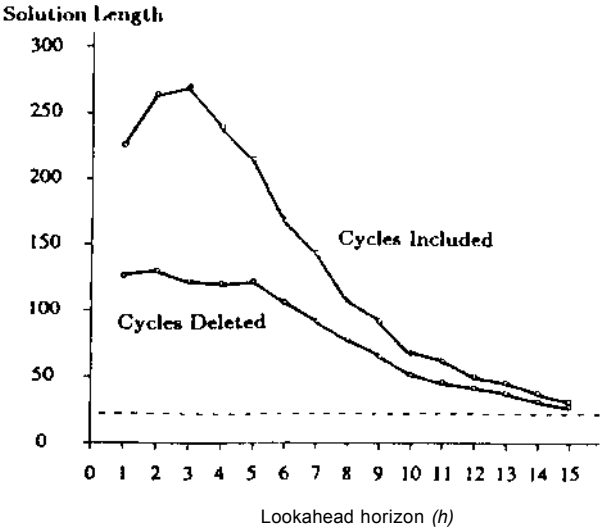

Figure 2: Solution Quality as a Function of Lookahead Horizon in the 8-puzzle. Plotted points are average values of RTA* running on 500 randomly generated problems. (Dotted line = optimal solution quality).

The next step is to compare solution quality and search time, as shown in Figure 4. Each point along the curve marks a particular choice of lookahead horizon. This data confirms one of the surprising results of [Korf, 1990]: if our goal is simply to find a solution—any solution—to the 8-puzzle, the fastest way to do it is to set the lookahead horizon to 1. That is: don't plan, just move. Be reactive.

One way to interpret the data in Figure 4 is as follows: if you have t seconds to spend looking for a solution, expect to find a solution with quality $q = f(t)$. Likewise, if you desire a solution of quality q, expect to spend $t = f^{-1}(q)$ seconds looking for it. Thus, Figure 4 gives us a whole range of deliberativeness and reactivity to choose from.

3 Multiple Agent Search

The problem with relying on Figure 4 is that RTA*'s behavior is highly erratic. The data points in Figure 4 are averages of 500 trials each. Figure 5 shows a 5000-triai histogram of solution quality for a reactive agent (lookahead horizon of 1). Why the unpredictability? Since RTA* makes decisions based on limited lookahead, various alternatives often look equally good. In that case RTA* must make a random choice. Of course, it may end up finding a terrible solution, and taking a long time to boot.[3]

How can we fix this problem? Taking a clue from Figure 4, we might run 500 independent agents to completion, then consult the agent that found an average-

[3]Actually, there are two sources of variation: one is RTA*'s random choice mechanism, and the other is the fact that some instances of the 8-puzzle are harder that others. The latter source of variation has a minimal effect, however: no instances require 100-move solutions, but RTA* routinely returns such solutions.

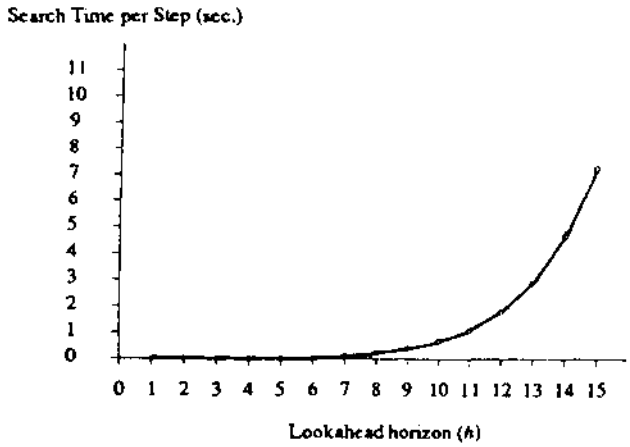

Figure 3: Search Time per Move as a Function of Lookahead Horizon in the 8-Puzzle

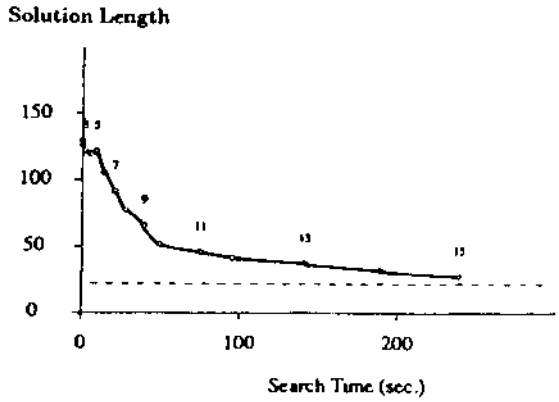

Figure 4: Solution Quality vs. Search Time in the 8-Puzzle. Optimal Solution quality is represented by dashed line. Small numbers indicate different values of the lookahead horizon *(h)*. Points plotted are averages of 500 trials each.

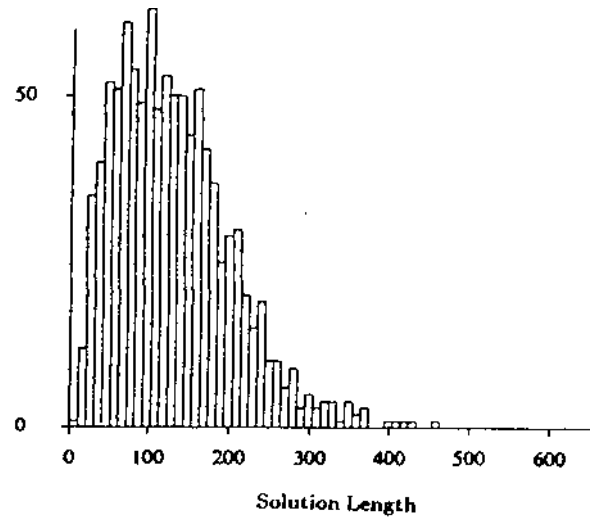

Figure 5: Variance in Solution Quality over 5000 8-Puzzle Problems (lookahead horizon h = 1).

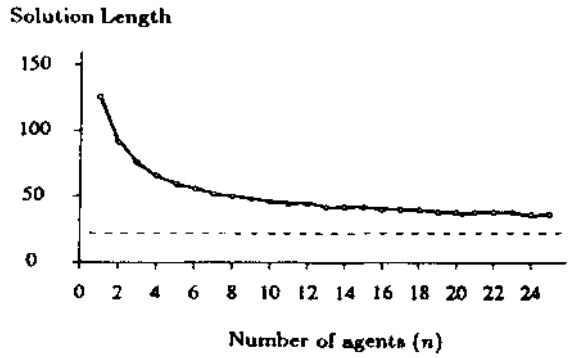

Figure 6: Solution Quality as a Function of the Number of Reactive Agents Solving Instances of the 8-Puzzle. Compare with Figure 2.

length solution. But then, we might as well take the *best* solution instead of the average one. What kind of solution quality can we expect to see if we take the best of n agents attacking a single problem?

Figure 6 shows how the number of agents affects solution quality. The figure depicts maximally reactive agents (lookahead horizon of 1). Note that solution quality improves with each additional agent, just as it improved when we increased the lookahead horizon of a single agent.

4 Increasing Deliberativeness versus Increasing the Number of Agents

At this point, we have two distinct methods for improving solution quality. We already know the exponential time costs associated with increasing the lookahead horizon. The next step is to investigate the cost of increasing the number of agents. Then we will be able to construct a new time versus quality curve.

The cost depends crucially on how the multiple agents are implemented. There are at least three possibilities:

(1) End to end—run several agents, one after another, on a sequential machine.

(2) Parallel—run all agents simultaneously, each on its own processor.

(3) Dovetail—simulate parallelism on a sequential machine by repeatedly giving each agent a time slice.

In case (1), search time increases linearly with the number of agents $(t = k_1 n)$. Here, k_1 is simply the average solution time of a single trial. In case (2), search time decreases with number of agents. This is because when one processor finds a solution, all processors can halt. The more processors we have, the more likely it is that one of them will find a very good solution very quickly. In the limit, we will find optimal solutions. At that point, adding more processors will cease to improve either solution quality or search time.

Case (3) is a very practical method for sequential machines. Like parallel search, dovetailing terminates when any one of the independent agents succeeds. In the case of large n, time increases linearly with n $(t = k_3 n)$. If n is

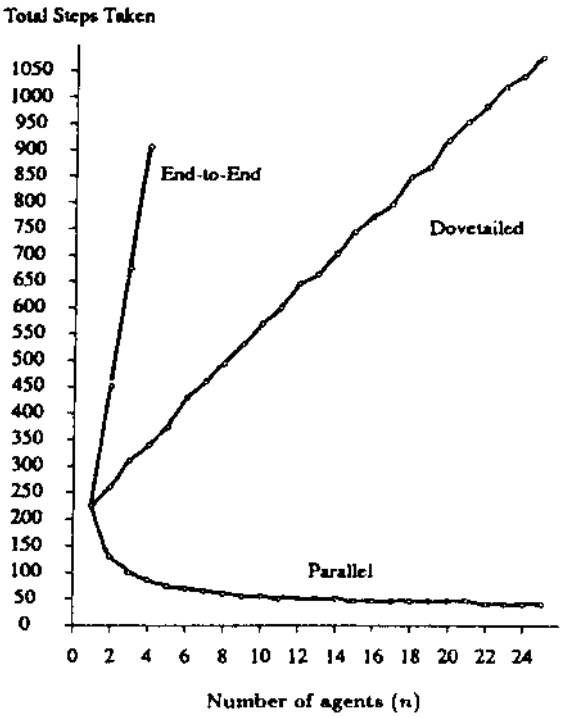

Figure 7: Search Time as a Function of Number of Agents in the 8-Puzzle. (Total steps taken is proportional search time here, since *h* is fixed at 1). Compare with Figure 3.

so large that near-optimal solutions are being generated, then doubling n will simply double the search time. But the slope constant is much smaller than case (1). Instead of the average solution time, k_3 is the near-optimal solution time. At smaller values of n, there are two opposing forces at work. Increasing n improves solution quality, so fewer steps are needed. But since there are multiple agents to dovetail among, search time will suffer. Experimental results are summarized in Figure 7. In this figure, lookahead horizon (h) is held constant at 1. With the horizon constant, search time is a (linear) function of the number of steps taken by all agents. (Since steps can be measured more accurately than search time, the figure uses steps.)

Still holding the lookahead horizon constant at 1, we can compute a search time versus solution quality curve for dovetailed agents. Each data point in Figure 8 represents a different value of n.

Now we can compare the two methods of improving solution quality: increasing *h* (Figure 4) and increasing n (Figure 8). The following table includes average search time and solution quality for three possible combinations of *h* and n.

Lookahead horizon (*h*)	Number of agents (*n*)	Solution Quality	Search Time
1	1	125.4	1.27 sec.
10	1	49.6	76.76 sec.
1	8	49.0	2.77 sec.

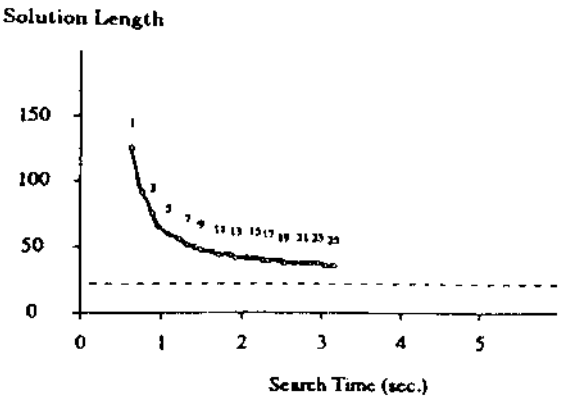

Figure 8: Solution Quality versus Search Time for Reactive Agents in the 8-Puzzle. Small numbers indicate varying numbers of agents. Execution is dovetailed on a sequential machine. Compare with Figure 4, especially the x-axis.

The first two lines in the table represent single-agent search, reactive and deliberative. The third line represents multiagent search. The dramatic result here is that eight reactive agents—dovetailed on a sequential machine—can match the solution quality of a single deliberative agent, and do so spending only a fraction of the time. This demonstrates the superiority of adding more reactive agents over increasing the deliberativeness of a single agent.

The benefit of multiagent search derives from the wide variation in solution quality for a single agent. The time cost is only *linear* in the number of agents. On the other hand, the benefit of deliberation derives from the knowledge gained by looking ahead. But the time cost is *exponential* in the lookahead horizon. The benefits are comparable, but the costs are not.

We can now state our results in terms of parallel speedup, i.e., uniprocessor time divided by multiprocessor (multiagent) time. To make a fair comparison, it is necessary to fix the desired solution quality, as has been done in the above table. It shows 8 agents achieving the same result as 1, but doing it faster by a factor of 27, dovetailed on a sequential machine, and *by a factor of 225 tn parallel.* This is a superlinear speedup, and it holds for all fixed values for solution quality. Superlinear speedups offer tremendous savings and have been reported most notably in [Mehrotra and Gehringer, 1985; Janakiram *et* a/., 1988; Rao and Kumar, 1988]. Of course, our speedup is relative to RTA*, not to the best sequential algorithm for generating solutions of fixed quality. Superlinear speedups are strictly impossible in such cases, since the dovetailed algorithm can be run on a sequential machine. By dovetailing agents, we have created a new uniprocessor algorithm against which new parallel algorithms must be measured.

The preceding discussion applies to the offline use of RTA*. In real-time, incomplete-information domains, theoretical superlinear speedups over the best single-agent algorithm *are* possible.

5 Other Domains

The following table shows some results for the 15-puzzle, the 4x4 version of the 8-puzzle:

Lookahead horizon (h)	Number of agents (n)	Solution Quality	Search Time
1	1	1232.6	15.3 sec.
1	3	726.9	18.5 sec.
1	20	328.7	44.9 sec.
1	40	275.5	74.3 sec.
1	80	213.4	113.0 sec.

Here, the lookahead horizon *(h)* is held constant at 1, while the number of agents (n) varies. Notice that moving from 1 to 3 agents yields a large improvement in solution quality at virtually no cost in search time.

We have also obtained a full set of empirical results for the (8-block) Blocks World domain. The results are just as compelling as those for the N-puzzle. It is far more advisable to tackle a blocks-world problem with many reactive agents than a few deliberative ones. For example:

Lookahead horizon (h)	Number of agents (n)	Solution Quality	Search Time
1	1	115.1	0.58 sec.
7	1	66.4	91.56 sec.
1	7	48.6	0.79 sec.

Work on applying our ideas to planning systems is currently under way. Planners like PRODIGY [Minton *et al.*, 1989] solve hard problems, but do not guarantee good quality solutions; other planners provide near-optimal solutions but do not scale up. We are exploring ways to bridge this gap by randomizing the arbitrary decisions made by a planner and employing multiple agents.

6 Agent Communication and Dispersal

The results of the previous sections indicate that where substantial variation in solution time and quality exists, many reactive agents should be employed instead of a few deliberative ones. In this section, we consider two issues that naturally arise: (1) if the agents are allowed to communicate and coordinate, can their performance be improved, and (2) how can agents disperse themselves in the absence of random tie-breaking?

We consider one rudimentary communication scheme- Agents communicate by sharing a single hash table, which records the states visited by all. Thus, one agent can benefit from the experience of another, who may have already mapped out a portion of the search space. Empirical experiments show that communicating reactive agents yielded solutions about 10% shorter than non-communicating agents. Search time savings vary with the number of agents: for 2 agents, there is a 2.6% improvement; for 10 agents a 6.1% improvement; for 23 agents, a 7.7% improvement. This communication scheme is easy to implement, and there is clearly room for more intelligent schemes.

The second issue is dispersal. Domains like the 8-puzzle use a small set of effectively discrete heuristic estimators. This leads RTA* to perform a large number of random tie-breaks, since alternative moves often look equally good. Fortunately, these tie-breaks also serve to disperse multiple agents. But domains like path planning through obstacles [Russell and Wefald, 1991] involve an infinite number of real-valued estimators. In such domains, our reactive agents simply move about in a single clump, since what looks best to one agent also looks best to another.

We have investigated one algorithm for effectively dispersing agents. This algorithm treats heuristic estimates as probabilities. We obtained probabilities by solving 100 sample problems using RTA*, and recording at each action cycle: (1) what the backed-up estimates were for various alternatives, and (2) what the best alternative really was. Here, "best" means "on an optimal path to the goal" (optimal paths were computed at each point with IDA* [Korf, 1985]). We define Stochastic RTA* as an algorithm that uses such probabilities to occasionally make what RTA* would consider a bad move. Agents using Stochastic RTA* disperse themselves automatically. In our initial experiments, a single (reactive) stochastic agent returned solutions of equal quality compared to a normal RTA* agent, but consumed 12%) more time. This slight decrease in single-agent performance washes out when multiple agents are employed.

7 Related Work

[Korf, 1988] is the only other work to address multiple agents in the context of real-time heuristic search. It reports initial experimental results, but it does not compare multiple agents with increased deliberation, nor does it measure solution quality. Also, it does not analyze the results in terms of superlinear speedup.

Beam search is another closely related algorithm. Roughly, multiple agent RTA* is to beam search as single agent RTA* is to beam search with a beam of width one. RTA* is guaranteed to find a solution, by looping back if necessary, while beam search may prune solutions completely. RTA* can also be used in reactive or deliberative mode, and in real-time or offline domains. Furthermore, multiple agent RTA* can be implemented straightforwardly on a general-purpose multiprocessor, whereas beam search involves large overhead costs due to synchronization [Bisiani, 1989].

Previous work in parallel processing has pioneered the use of multiple processors for reliability and performance enhancement. [Mehrotra and Gehringer, 1985] report superlinear speedups when individual processors have varying runtimes due to randomization. [Smith and Maguire, 1989b] investigate using parallelism and randomization to tackle OR-parallelism in PROLOG. [Janakiram *et al.*, 1988] also tackle this blind search problem and remark that it would be interesting to pursue randomizing heuristic search. They also analyze the expected speedup for various running time probability distributions (but unfortunately not for the log-normal distribution of the type seen in our experiments). [Smith and Maguire, 1989a] and [Goldberg and Jefferson, 1987]

present similar work; none of these papers addresses heuristic search or solution quality.

AI has seen work in parallel heuristic search. [Kumar and Rao, 1987] and [Rao and Kumar, 1988] report super linear speedups in depth-first IDA*. [Rao and Kumar, 1992J study speedup under varying assumptions about the the distribution of solution states. [Saletore and Kale, 1990] investigate how to achieve a reliable, consistent linear speedup. These algorithms concentrate on optimal or near optimal solutions, whereas we are concerned a flexible tradeoff between solution quality and search time. Also, these algorithms do not interleave planning and execution, which is necessary in incomplete-information domains. Other parallel work can be found in [Powley and Korf, 1991], [Huang and Davis, 1989], and [Li and Wah, 1991].

8 Conclusions and Future Work

The RTA* algorithm yields a tradeoff between search time and solution quality. Increasing RTA*'s lookahead horizon yields better solutions, but the search time increases exponentially. This paper has investigated another method for improving solution quality. It uses n agents, each of which is repeatedly given a time slice. Search time only increases linearly with n, but solution quality improves very rapidly. When solution quality is held constant, employing n parallel agents yields superlinear speedups.

There are several directions in which to expand this work: (1) investigate new algorithms for agent dispersal and communication, building on work described in Section 6; (2) investigate the behavior of heterogeneous collections of agents, e.g., agents that use different heuristic evaluation functions, or agents with varying levels of deliberativeness; and (3) apply the method to a wide range of search and planning domains, e.g., ones with different solution densities, action costs, and heuristic value distributions.

9 Acknowledgements

This work was supported in part by the National Science Foundation under contract IRI-8858085. Thanks to Yolanda Gil, Milind Tambe, and Gary Knight for suggestions and assistance.

References

[Bisiani, 1989] R. Bisiani. Beam: An accelerator for speech recognition. In Proc. IEEE Conf. on Acoustics, Speech and Signal Processing, 1989.

[Goldberg and Jefferson, 1987] A. Goldberg and D. Jefferson. Transparent process cloning: A tool for load management of distributed programs. In International Conference on Parallel Processing, 1987.

[Hart et a/., 1968] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on SSC, 4, 1968.

[Huang and Davis, 1989] S. Huang and L. Davis. Parallel iterative A* search: An admissible distributed heuristic search algorithm. In Proc. IJCAI, 1989.

[Janakiram et al, 1988] V. Janakiram, D. Agrawal, and R. Mehrotra. A randomized parallel backtracking algorithm. IEEE Trans, on Computers, 37(12), 1988.

[Korf, 1985] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 27(1), 1985.

[Korf, 1988] R. Korf. Multi-agent heuristic search. In Proc. of the Workshop on Distributed AI, Lake Arrowhead, CA, 1988.

[Korf, 1990] R. Korf. Real-time heuristic search. Artificial Intelligence, 42(2-3), 1990.

[Kumar and Rao, 1987] V. Kumar and V. N. Rao. Parallel depth first search. International Journal of Parallel Programming, 1987.

[Li and Wah, 1991] G. Li and B. Wah. Parallel iterative refining A* search. In International Conference on Parallel Processing, 1991.

[Mehrotra and Gehringer, 1985] R. Mehrotra and E. Gehringer. Superlinear speedup through randomized algorithms. In International Conference on Parallel Processing, 1985.

[Minton et al., 1989] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. Artificial Intelligence, 40(1-3), 1989.

[Powley and Korf, 1991] C. Powley and R. Korf. Single-agent parallel window search. IEEE PAMI, 13(5), 1991.

[Rao and Kumar, 1988] V. N. Rao and V. Kumar. Superlinear speedup in state-space search. In Proc. Foundation of Software Technology and Theoretical Computer Science, 1988.

[Rao and Kumar, 1992] V. Rao and V. Kumar. On the efficiency of parallel backtracking. IEEE Trans, on Parallel and Dist. Systems, (To appear), 1992.

[Russell and Wefald, 1991] S. Russell and E. Wefald. Do the Right Thing: Studies in Limited Rationality. MIT Press/Cambridge, MA, 1991.

[Saletore and Kale, 1990] V. Saletore and L. Kale. Consistent linear speedups to a first solution in parallel state-space search. In Proc. AAAI, 1990.

[Simon, 1957] H. A. Simon. Models of Man. Wiley, New York, 1957.

[Smith and Maguire, 1989a] J. Smith and G. Maguire. Exploring 'multiple worlds' in parallel. In International Conference on Parallel Processing, 1989.

[Smith and Maguire, 1989b] J. Smith and G. Maguire. Transparent concurrent executions of mutually exclusive alternatives. In 9th IEEE International Conference on Distributed Computing Systems, 1989.