Multiagent Real-Time-A* with Selection: Introducing Competition in Cooperative Search

Makoto Yokoo
NTT Communication Science Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun,
Kyoto 619-02 Japan
e-mail: yokoo@cslab.kecl.ntt.jp

Yasuhiko Kitamura
Faculty of Engineering,
Osaka City University
3-3-138 Sugimoto
Sumiyoshi-ku, Osaka 558 Japan
e-mail: kitamura@info.eng.osaka-cu.ac.jp

Abstract

A new cooperative search algorithm that introduces a GA-like selection mechanism is developed. In this algorithm, a state-space search problem is solved concurrently by multiple agents, each of which executes the Real-Time-A* algorithm. These agents compete for existence, i.e., each agent periodically reproduces offspring stochastically based on its fitness defined by the heuristic estimation value of its current state, so that an agent in a good state tends to reproduce many offspring while an agent in a bad state tends to be exterminated.

Experimental evaluations show that this algorithm is very effective for problems that can be divided into serializable subgoals (e.g., n-puzzles), although the agents do not have any knowledge about these subgoals. In particular, this algorithm can solve the 48-puzzle, which can not be solved by existing heuristic search algorithms consistently within a reasonable amount of time unless the knowledge about the subgoals is explicitly given.

Introduction

Since virtually all AI problems require some sort of search, search has been playing an important role in AI (Korf 1992). When solving a difficult search problem, a problem solver has to make critical choices for alternatives. While a correct choice leads to a relatively rapid search, a poor choice results in a very long search time. Although we can give the problem solver knowledge enable it to make correct choices (*heuristics*), we can not guarantee that the problem solver can always make good decisions.

Cooperative search methods try to reduce this difficulty by introducing multiple problem solvers (agents) (Clearwater, Huberman, & Hogg 1991; Hogg & Williams 1993; Knight 1993). Since a problem is solved concurrently by multiple agents, if at least one agent can make correct choices, a solution can be obtained in a reasonable amount of time. These agents cooperate by exchanging information found during the search process. Cooperative search methods have been applied to constraint satisfaction problems (Clearwater, Huberman, & Hogg 1991; Hogg & Williams 1993) and state-space search problems (Knight 1993; Kitamura, Teranishi, & Tatsumi 1996a; 1996b).

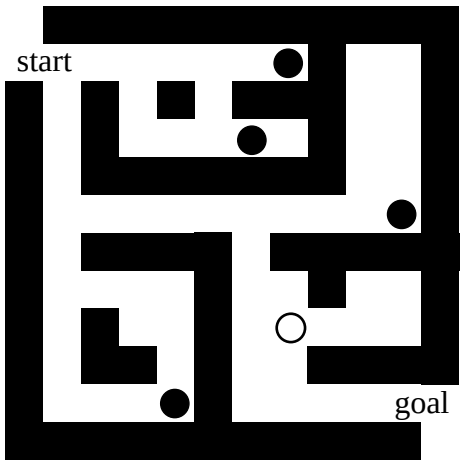


Figure 1: Search in 10×10 grid state-space (5 agents)

One drawback of existing cooperative search methods is that if agents have to make critical decisions repeatedly, those agents that made bad decisions early are unable to contribute to the ongoing search process. For example, in Figure 1, five agents are in a 10*10 grid state-space with obstacles. A circle represents an agent, the initial state is the upper-left corner, and the goal state is the bottom-right corner. Only one agent (the white circle) has been making correct choices[1], and the other agents are unable to contribute to the ongoing search process since it will take a very long time for them to recover from the incorrect choices.

We can assume agents involved in cooperative search to be limited resources, and hope to make the best use of them by concentrating search efforts only on promising states. However, it is very difficult to distinguish promising states from the other states. If we could do this correctly, mistakes on critical choices could be

[1] Agents use the Manhattan distance to the goal state as a heuristic guide.

avoided, and multiple agents would be unnecessary. By concentrating the agents on incorrect states, we lose the benefit of the diversity of decisions.

In this paper, we develop a cooperative search algorithm that introduces competition among agents, just like the selection mechanism in genetic algorithms (GAs) (Goldberg 1989). More specifically, each agent solves a state-space search problem using the Real-Time-A* algorithm (Knight 1993; Korf 1990). Periodically, each agent reproduces offspring stochastically based on its fitness defined by the heuristic estimation value of its current state. In other words, if an agent is in a state with a good heuristic evaluation value, it tends to have more offspring in the next generation, while if an agent is in a state with a bad evaluation value, it tends to be exterminated. By introducing the selection mechanism, this algorithm can concentrate agents on promising states without sacrificing the diversity of decisions too much.

Experimental evaluations show that this algorithm is so effective for n-puzzles that it can solve the 48-puzzle. A careful examination of the algorithm execution traces reveals that the algorithm with the selection mechanism utilizes the existence of serializable subgoals (Korf 1988; Newell & Simon 1972), although the agents do not have any knowledge about these subgoals.

In the following of this paper, we briefly describe the Real-Time-A* algorithm and the Multiagent Real-Time-A* algorithm. Then, we show the Multiagent Real-Time-A* algorithm with selection, and present experimental results that illustrate the efficiency of this algorithm. Finally, we give discussions clarifying the reason for the dramatic speed-up in solving n-puzzles and the relation with GAs.

Multiagent Real-Time-A* Algorithm

Real-Time-A*

The goal for a state-space search problem (Pearl 1984) is to find a path from an initial state to a goal state. A typical example problem is the 8-puzzle (Figure 2). In this puzzle, there exist eight numbered tiles arranged on a 3×3 board. The goal is to transform the given initial state to the goal state by sliding tiles onto an empty square.

State-space search algorithms can be divided into two groups: *off-line* and *real-time*. Off-line algorithms, such as the A* algorithm (Pearl 1984), compute an entire solution path before executing the first step in the path. Real-time algorithms, such as the Real-Time-A*(Korf 1990), perform sufficient computation to determine a plausible next move, execute that move, then perform further computation to determine the following move, and so on, until the goal state is reached. These algorithms can not guarantee to find the optimal solution, but usually find a suboptimal solution more rapidly than off-line algorithms.

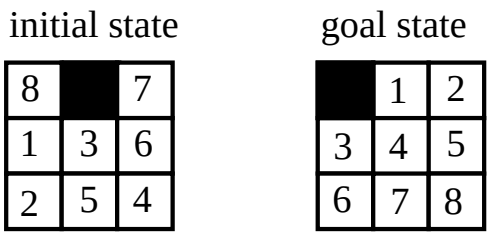


Figure 2: Example of a state-space search problem (8-puzzle)

The Real-Time-A* algorithm can be considered as a kind of hill climbing search. However, the algorithm revises a table of heuristic estimates of the distances from each state to the goal state during the search process. Therefore, the algorithm is not trapped in a local minimum and is guaranteed to be *complete* in the sense that it will eventually reach the goal, if certain conditions are satisfied[2].

The Real-Time A* repeats the following steps[3] until the goal state is reached. Let x be the current state of the problem solver.

1. Calculate $f(x') = h(x') + k(x, x')$ for each neighbor x' of the current state x, where $h(x')$ is the current heuristic estimate of the distance from x' to the goal state, and $k(x, x')$ is the distance between x and x'.
2. Move to a neighbor with the minimum $f(x')$ value. Ties are broken randomly.
3. Update the value of $h(x)$ to the second-best $f(x')$ value.

The reason for updating $h(x)$ to the second-best value is that if the problem solver visits x again, the problem solver will take the best of the alternatives that were not chosen (the second-best).

It must be noted that there are two alternative interpretations of the Real-Time A* algorithm. One interpretation is that this algorithm is a fast search algorithm that can produce an acceptable solution very rapidly. Another interpretation is that this algorithm is on-line, i.e., it represents the situation in which an agent is interleaving planning and actions in the real-world. In this paper, we employ the first interpretation, and try to further improve the efficiency of the Real-Time A* algorithm. Ofcourse, in some applications, e.g., the exploration in unknown environment by multiple robots, we can employ the second interpretation for the Multiagent Real-Time A*. However,

[2]These conditions are as follows: the problem space is finite, all edge costs are positive, there exists a path from every state to the goal state, and the values of initial heuristic estimates are finite.

[3]This algorithm shows the procedure for the case the depth of the look-ahead horizon is 1.

the technique we introduce in this paper (the selection/rearrangement of agents) can not be performed without requiring additional costs if actions are performed in the real-world.

Multiagent Real-Time A*

In the Multiagent Real-Time-A* algorithm (Knight 1993), multiple agents solve a common state-space search problem concurrently using the Real-Time-A*. Each agent has enough knowledge to solve the problem alone and maintains its history, i.e., the sequence of the states it has visited. When an agent reaches the goal state, its history is a solution.

Although the agents do not need to communicate to find a solution, we assume that agents communicate by sharing a single table of heuristic estimates[4]. Thus, one agent can benefit from the experience of another.

Since there are a large number of random tie-breaks in the Real-Time A*, the current states of agents are dispersed eventually even though the agents share the same table of heuristic estimates.

Selection in Multiagent Real-time A*

We introduce a GA-like selection mechanism into the Multiagent Real-Time-A* by the following procedure. Let us assume the heuristic estimate of agent *i's* current state is represented as h_i, and the number of agents is n. Let us count a single concurrent move of agents as one step. Once in a certain number of steps T, each agent stochastically generates its offspring, then the agent is terminated. The total number of agents in the next generation is fixed to n. Each offspring inherits the history of its parent, and restarts the search process from the state where the parent was terminated. The next generation is created by the following procedure.

for $i = 1$ to n **do**
  if $h_i = \min_{k \in \{1,2,\ldots,n\}} h_k$ **then**
    create one offspring of agent i;
  else select one agent from $\{1, 2, \ldots, n\}$
    and create one offspring, where the selection probability of j is $\frac{1}{h_j \sum_{k=1}^{n} 1/h_k}$;
  end if;
end do;

The selection probability of agent j in each iteration is proportional to $1/h_j$[5]. We can assume that $1/h_j$ represents the *fitness* of agent j. Also, the agent with the best heuristic estimate always has at least one offspring[6].

[4]The shared h values are no longer *admissible* from the other agents' perspective, even if the initial h values are admissible. However, non-admissible heuristic estimates do not affect the completeness of the algorithm.

[5]We assume h_j is non-zero except the goal state.

[6]This strategy is called *elitism* in GA studies (Goldberg 1989).

Table 1: Evaluation in grid state-space search (selection interval=100)

algorithm	steps	solution length
1 agent	6795.1	403.8
10 agents, without selection	1948.7	404.4
10 agents, with selection	1572.3	400.2

For example, assume that there exist only two agents, and the estimate of agent 1's current state is 99, and the estimate of agent 2's current state is 1. Then, from the above procedure, with a 99% chance, agent 2 has two offspring, and with a 1% chance, each agent has one offspring.

Evaluations

In this section, we show the effect of the selection mechanism in the Multiagent Real-Time A* using experimental evaluations for typical benchmark problems, i.e., grid state-space search problems and n-puzzles.

Grid State-Space Search

The first problem is a grid state-space search problem such as the one in Figure 1. There exists a grid state-space with randomly positioned obstacles. We allow motions along the horizontal and vertical dimensions, but not diagonal motions. The initial state is at the upper-left corner and the goal state is at the bottom-right corner. We generate 120×120 grid state-spaces, in which the ratio of obstacles is 40%. An initial value in the table of heuristic estimates is the Manhattan distance to the goal state.

We show the average of required steps for 100 trials (10 trials for each of 10 randomly generated problems) in Table 1. A concurrent single move of agents is counted as one step. We show the case that only one agent solves the problem, the case that 10 agents solve the problem without the selection mechanism, and the case that 10 agents solve the problem with the selection mechanism, in which the selection interval is 100, i.e., a selection is performed once in 100 steps. Furthermore, we show the average length of the obtained solution, i.e., the length of the path after removing loops. As the table shows, by introducing the selection mechanism, we can obtain about a 20% speed-up.

To illustrate the effect of the selection interval and the number of agents, we show results from the 10 agents' case, varying the selection interval (Figure 3(a)), and from the case that the selection interval is fixed to 100, while the number of agents is changed (Figure 3(b)).

We can see that if the selection interval is too short (e.g., 1), the agents are concentrated too much in one direction, and introducing the selection degrades the

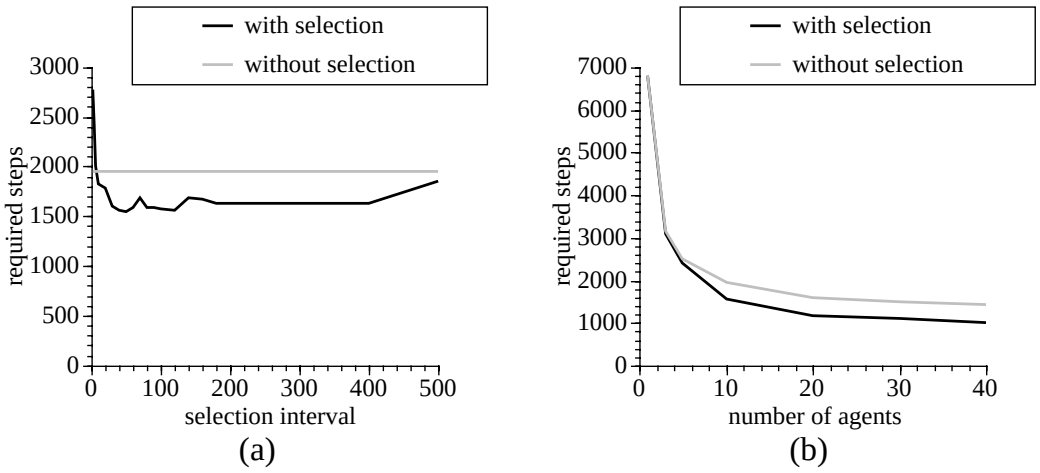


Figure 3: Evaluation in grid state-space search (effect of selection interval/number of agents)

Table 2: Evaluation in 24-puzzle (selection interval=5)

algorithm	steps	solution length
1 agent	66901.7	23538.6
5 agents, without selection	14815.9	7648.6
5 agents, with selection	2942.0	1154.7

search performance. Otherwise, the selection mechanism is beneficial.

N-puzzle

We show the required steps for the 24-puzzle (the averages of 100 trials with different randomly generated initial states) in Table 2. An initial value in the table of heuristic estimates is the sum of Manhattan distances of wrongly placed tiles. We show the case that only one agent solves the problem, the case that five agents solve the problem without the selection mechanism, and the case that five agents solve the problem with the selection mechanism (in which the selection interval is 5). To illustrate the effect of the selection interval, we show results from the five agents' case, varying the selection interval (Figure 4).

Furthermore, we show the required steps for the 35-puzzle and 48-puzzle (the averages of 100 trials with different randomly generated initial states) in Table 3.

In order to terminate the experiments in a reasonable amount of time, the total number of steps over agents is limited to one million (each agent can take at most 200,000 steps in 5 agents cases), and we interrupt any trial that exceeds this limit. We show the average of successfully terminated trials only, and show the ratio of successfully terminated trials within the limit.

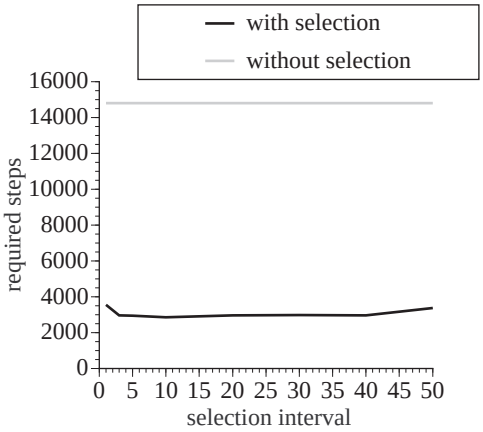


Figure 4: Evaluation in 24-puzzle (effect of selection interval)

As these results illustrate, the effect of introducing the selection mechanism is very impressive for n-puzzles. As far as the authors' knowledge, there has been no unidirectional heuristic search algorithm that can solve the 35- or 48-puzzle consistently, using the Manhattan distance as the only guide.

To confirm this fact, we are going to show the evaluation results of two well-known off-line search algorithms. One is the weighted-A* (WA*) algorithm(Korf 1993), where $f(x) = h(x)$. This algorithm tends to exhaust available memory very rapidly since it needs to store all expanded nodes in the open or close lists. Another one is the recursive best-first search (RBFS) algorithm(Korf 1993), where $f(x) = g(x) + W \times h(x)$. This algorithm can run in linear space. The parameter W represents the weight on h. If $W = 1$, this algorithm finds the optimal solution. Korf (1993) reports

Table 3: Evaluation in 35-/48-puzzle (selection interval=5)

problem	algorithm	steps	solution length	ratio
35-puzzle	1 agent	420489.4	219553.3	0.42
	5 agents, without selection	83346.7	46827.4	0.43
	5 agents, with selection	8685.5	3198.0	1.0
48-puzzle	1 agent	393439.0	228849.0	0.02
	5 agents, without selection	125723.0	14527.0	0.01
	5 agents, with selection	35452.7	11045.0	1.0

Table 4: Evaluation in 35-/45-puzzle (WA* and RBFS)

problem	algorithm	steps	solution length	ratio
35-puzzle	WA*	110397.3	1968.2	0.6
	RBFS	1058833.0	297.7	0.06
48-puzzle	WA*	189241.9	3151.0	0.1
	RBFS	—	—	0

that required time for finding a solution is optimized when W is set to 3 in the 15-puzzle. This is because RBFS tends to explore all possible paths to a given node, and the number of duplicate nodes explodes as the search depth increases. Since increasing the weight on h increases the search depth, small weights can work better than large weights.

We solved the problem instances used in Table 3 by these algorithms. The results are summarized in Table 4. We set the memory limit of the WA* to one million, and the limit of expanded nodes in the RBFS ($W = 3$) to 10 million. We show the average steps (the number of expanding a node) of the solved instances, and the ratio of successfully solved instances. We can see neither of these algorithms can solve the 35- or 48-puzzle consistently, although these algorithms can produce shorter solutions if they can find them.

Discussions

Serializable Subgoals

Why is the selection mechanism very effective for n-puzzles? A careful examination of the algorithm execution traces reveals that this algorithm utilizes the existence of serializable subgoals (Korf 1988; Newell & Simon 1972).

A problem has serializable subgoals iff the goal can be divided into a set of subgoals, and there exists an ordering among the subgoals such that the subgoals can always be solved sequentially without ever violating a previously solved subgoal in the order (Korf 1988). For example, if we solve the bottom row of the 15-puzzle as a subgoal, then we can always solve the rest of the problem without disturbing the bottom row. Furthermore, if we solve the right column as the next subgoal, the problem is reduced to the 8-puzzle (Figure 5).

Let us examine how these subgoals are achieved in the Multiagent Real-Time-A*. Let us call the states in which the first subgoal (either the bottom row or the right column is solved) is achieved *level 1* states, and the states in which the first and second subgoals (both the bottom row and the right column are solved) are achieved *level 2* states, and so on. Figure 6 shows traces of maximal levels of agent states when five agents are solving the 24-puzzle with and without the selection mechanism (the selection interval is 5).

As shown in Figure 6, when using the selection mechanism, once a certain subgoal is achieved, this subgoal is rarely violated by the agents as a whole, and the next subgoal is achieved eventually. On the other hand, when the selection mechanism is not used, achieved subgoals are fragile. When using the selection mechanism, the heuristic estimation of the state that achieves a subgoal is relatively good, thus the agent in that state has a higher probability of making many offspring in the next generation. As a result, the achieved subgoal remains stable, i.e., at least one agent keeps the subgoal, and the next subgoal is likely to be achieved.

It must be emphasized that agents do not have any knowledge about the serializable subgoals. The agents use only the heuristic estimations, where these estimations reflect serializable subgoals very weakly, i.e., the states that achieve the subgoals are relatively preferable to other states.

To reconfirm the fact that the selection mechanism can utilize serializable subgoals, we perform evaluations in the Tower of Hanoi problem (Pearl 1984), which is a typical problem that can be divided into serializable subgoals. The problem is described as fol-

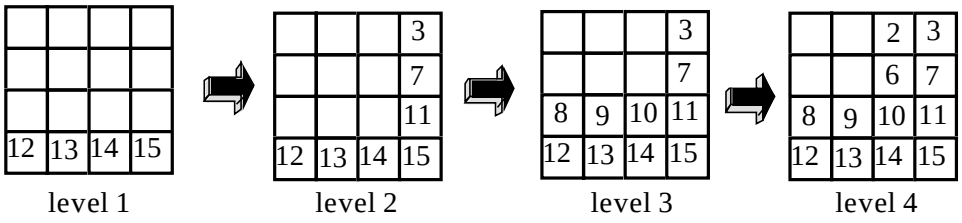
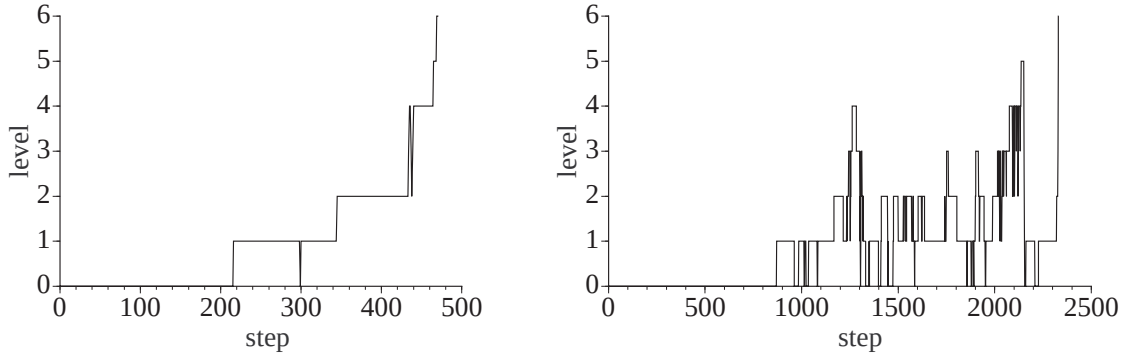


Figure 5: Serializable subgoals in n-puzzle



(a) with selection (b) without selection

Figure 6: Traces of subgoal achievement in 24-puzzle

lows:

> There are m disks $D_1, D_2, \ldots, D_m$, of graduated sizes and three pegs 1, 2, and 3. Initially all the disks are stacked on peg 1, with D_1, the smallest, on top and D_m, the largest, at the bottom. The problem is to transfer the stack to peg 3 given that only one disk can be moved at a time and that no disk may be placed on top of a smaller one.

This problem can be solved efficiently by decomposing the problem into subgoals, and the optimal number of required steps is given by $2^m - 1$. The obvious serializable subgoals are {(stack D_m on peg 3), (stack D_{m-1} on peg 3), ..., (stack D_1 on peg 3)}.

An initial value in the table of heuristic estimates is given as follows.

- For each disk D_i of the state, calculate h_{D_i} and use the summation as the heuristic estimate of the state.
- h_{D_i} is defined as follows:
  - If D_i is on peg 1 or peg 2, and there is no disk under D_i: 1
  - If D_i is on peg 1 or peg 2, and there is some disk under D_i: 2
  - If D_i is on peg 3, and the disks under D_i are $D_{i+1}, D_{i+2}, \ldots, D_m$ (i.e. identical to the goal state): 0
  - If D_i is on peg 3, and the disks under D_i are not identical to the goal state: 2

This heuristic estimation is identical to the optimal number of required steps in the case that there exist m pegs rather than 3 pegs. Therefore, this heuristic estimation is admissible.

We show the average of required steps for 50 trials in which 20 agents solve a 10 disk problem (the selection interval is 5) in Table 5. We can see a three-fold speed-up using the selection mechanism. Furthermore, we show traces of maximal levels of agent states when 10 agents are solving an eight disk problem[7] in Figure 7. As the figure shows, we can see that the achieved subgoals are rarely violated by agents as a whole when using the selection mechanism.

Does an algorithm that can utilize serializable subgoals have any practical advantages? Why aren't agents explicitly given knowledge of the serializable subgoals? For example, the Tower of Hanoi can be solved optimally without performing any search if the agents know the subgoals.

As discussed in Korf (1988), finding serializable subgoals is very difficult, i.e., proving that a set of subgoals is serializable is as difficult as proving that a given

[7] As in the n-puzzle, we call the states in which the first subgoal (stack D_8 on peg 3) is achieved level 1 states, and so on.

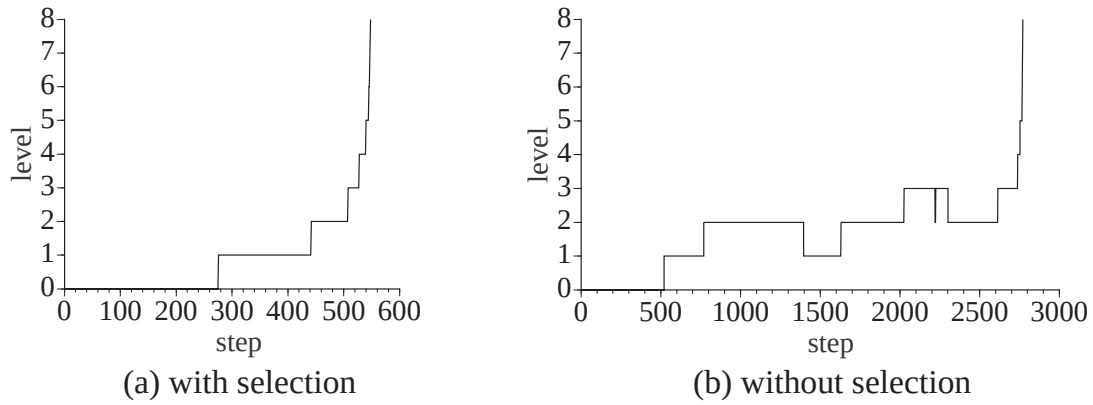


Figure 7: Traces of subgoal achievement in the Tower of Hanoi

Table 5: Evaluation in the Tower of Hanoi (10 disks, selection interval=5)

algorithm	steps
20 agents, without selection	23796.8
20 agents, with selection	7402.0

problem is solvable from all initial states. Actually, the authors did not notice the existence of serializable subgoals in n-puzzles when they first implemented this algorithm. Therefore, this algorithm has great practical advantages, since it can utilize serializable subgoals even if neither agents nor the designer of the algorithm has knowledge about them.

Relation with Genetic Algorithms

The selection mechanism in this algorithm is inspired by GA studies (Goldberg 1989). However, the algorithm as a whole is quite different from GAs in the following points.

- In GAs, an individual or *gene* represents a candidate of a solution. On the other hand, in this algorithm, a solution is a path from the initial state to the goal state (i.e., the history of transformations of individuals). Therefore, applying operators like *mutation* or *crossover* to the states does not make sense.
- Instead of applying operators like mutation or crossover, a state is changed by a local search procedure that uses heuristic estimations.
- The local search procedure (the Real-Time-A*) can escape from locally optimal states. Therefore, this algorithm can perform well with very small populations compared with GAs.

Conclusions

This paper presented the Multiagent Real-Time-A* algorithm that introduces a GA-like selection mechanism. By introducing the selection mechanism, this algorithm can concentrate agents on promising states while maintaining the diversity of decisions. Experimental evaluations have shown that this algorithm is very effective if the problems have serializable subgoals, even if neither agents nor the designer of the algorithm has any knowledge about these subgoals. In particular, this algorithm can solve the 48-puzzle, which can not be solved by existing heuristic search algorithms consistently within a reasonable amount of time unless the knowledge about the subgoals is explicitly given.

Our future works include confirming the efficiency of this algorithm in real-life application problems, and examining the applicability of this framework in other search domains such as constraint satisfaction problems.

Acknowledgments

The initial idea of this research emerged during the discussions at a workshop of Multiagent Research community in Kansai (MARK). The authors wish to thank members of MARK for their discussions, and Keihanna Interaction Plaza Inc. for supporting MARK. We also thank Koichi Matsuda and Nobuyasu Osato for their support in this work.

References

Clearwater, S. H.; Huberman, B. A.; and Hogg, T. 1991. Cooperative solution of constraint satisfaction problems. *Science* 254:1181–1183.

Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Hogg, T., and Williams, C. P. 1993. Solving the really hard problems with cooperative search. In *Proceed-*

ings of the Eleventh National Conference on Artificial Intelligence, 231–236.

Kitamura, Y.; Teranishi, K.; and Tatsumi, S. 1996a. An organizational approach to multi-agent real-time search and its evaluation. *Journal of Japanese Society for Artificial Intelligence* 11(3):470–477.

Kitamura, Y.; Teranishi, K.; and Tatsumi, S. 1996b. Organizational strategies for multiagent real-time search. In *Proceedings of the Second International Conference on Multi-Agent Systems*. MIT Press.

Knight, K. 1993. Are many reactive agents better than a few deliberative ones? In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 432–437.

Korf, R. E. 1988. Optimal path finding algorithms. In Kanal, L., and Kumar, V., eds., *Search in Artificial Intelligence*. Springer-Verlag. 223–267.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2–3):189–211.

Korf, R. E. 1992. Search. In Shapiro, S. C., ed., *Encyclopedia of Artificial Intelligence*. New York: Wiley-Interscience Publication. 1460–1467. second edition.

Korf, R. E. 1993. Linear-space best-first search. *Artificial Intelligence* 62(1):41–78.

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

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.