Improving the Learning Efficiencies of Realtime Search

Toru Ishida **Masashi Shimbo**
Department of Information Science
Kyoto University
Kyoto, 606-01, JAPAN
{ishida, shimbo}@kuis.kyoto-u.ac.jp

Abstract

The capability of learning is one of the salient features of realtime search algorithms such as LRTA*. The major impediment is, however, the instability of the solution quality during convergence: (1) they try to find all optimal solutions even after obtaining fairly good solutions, and (2) they tend to move towards unexplored areas thus failing to balance exploration and exploitation. We propose and analyze two new realtime search algorithms to stabilize the convergence process. *ε-search* (*weighted realtime search*) allows suboptimal solutions with ε error to reduce the total amount of learning performed. *δ-search* (*realtime search with upper bounds*) utilizes the upper bounds of estimated costs, which become available after the problem is solved once. Guided by the upper bounds, δ-search can better control the tradeoff between exploration and exploitation.

Introduction

Existing search algorithms can be divided into two classes: offline search such as A* [Hart *et al.*, 1986], and realtime search such as Real-Time-A* (RTA*) and Learning Real-Time-A* (LRTA*) [Korf, 1990]. Offline search completely examines every possible path to the goal state before executing that path, while realtime search makes each decision in a constant time, and commits its decision to the physical world. Realtime search cannot guarantee to find an optimal solution, but can interleave planning and execution. Various extensions of realtime search have been studied in recent years [Russell and Wefald, 1991; Chimura and Tokoro, 1994; Ishida and Korf, 1995; Ishida, 1996].

Another important capability of realtime search is learning, that is, as in LRTA*, the solution path converges to an optimal path by repeating problem solving trials.[1] In this paper, we will focus not on the performance of the first problem solving trial, but on the learning process to converge to an optimal solution.

[1]Barto clarified the relationship between LRTA* and Q-learning, by showing that both are based on dynamic programming techniques [Barto *et al.*, 1995].

This paper is the first to point out that the following problems are incurred when repeatedly applying LRTA* to solve a problem.

- Searching all optimal solutions:

  Even after obtaining a fairly good solution, the algorithm continues searching for an optimal solution. When more than one optimal solution exists, the algorithm does not stop until it finds all of them. Since only the lower bounds of actual costs are memorized, the algorithm cannot determine whether the obtained solution is optimal or not. In a realtime situation, though, it is seldom important to find a truly optimal solution (it is definitely not important to obtain all of them), but the algorithm is not satisfied with suboptimal solutions.

- Instability of solution quality:

  Every realtime search algorithm always moves toward a state with the smallest estimated cost. The initial estimated costs are given by a heuristic evaluation function. To guarantee convergence to optimal solutions, the algorithm requires the heuristic evaluation function be *admissible*, i.e., it always returns the lower bound of actual cost [Pearl, 1984]. As a result, the algorithm tends to explore unvisited states, and often moves along a more expensive path than the one obtained before.

In this paper, we propose two new realtime search algorithms to overcome the above deficits: *ε-search* (*weighted realtime search*) allows suboptimal solutions with ε error, and *δ-search* (*realtime search with upper bounds*) which balances the tradeoff between exploration and exploitation. ε-search limits the exploration of new territories of a search space, and δ-search restrains the search in the current trial from going too far away from the solution path found in the previous trial. The upper bounds of estimated costs become available after the problem is solved once, and gradually approach the actual costs by repeating a problem solving trial.

Previous Research

LRTA* Algorithm

We briefly review previous work on realtime search algorithms, in particular Learning-Real-Time-A* (LRTA*) [Korf, 1990]. LRTA* commits to individual moves in a constant time, and interleaves the computation of moves with their execution. It builds and updates a table containing heuristic estimates of the cost from each state in the problem space. Initially, the entries in the table come from a heuristic evaluation function, or are set to zero if no function is available, and are assumed to be lower bounds of actual costs. Through repeated exploration of the space, however, more accurate values are learned until they eventually converge to the actual costs to the goal.

The basic LRTA* algorithm repeats the following steps until the problem solver reaches the goal state. Let x be the current position of the problem solver.

1. Lookahead:
   Calculate $f(x') = k(x, x') + h(x')$ for each neighbor x' of the current state x, where $h(x')$ is the current lower bound of the actual cost from x' to the goal state, and $k(x, x')$ is the edge cost from x to x'.
2. Consistency maintenance:
   Update the lower bound of the state x as follows.
   $$h(x) \leftarrow \min_{x'} f(x')$$
3. Action selection:
   Move to neighbor x' that has the minimum $f(x')$ value. Ties are broken randomly.

The reason for updating the value of $h(x)$ is as follows. Every path to the goal state must pass through one of the neighbors. Since the $f(x')$ values represent lower bounds of actual costs to the goal state through each of the neighbors, the actual cost from the given state must be at least as large as the smallest of these estimates.

In a finite problem space with positive edge costs, in which there exists a path from every node to the goal state, LRTA* is *complete* in the sense that it will eventually reach the goal state. Furthermore, if the initial heuristic values are admissible, then over repeated problem solving trials, the values learned by LRTA* will eventually converge to their actual costs along every optimal path to the goal state [Korf, 1990].

LRTA* Performance

To evaluate the efficiency of LRTA*, we use a rectangle problem space containing 10,000 states. In this evaluation, obstacles are randomly chosen grid positions. For example, an obstacle ratio of 35% means that 3,500 randomly chosen locations in the 100×100 grid are replaced by obstacles. With high obstacle ratios (more than 20%), obstacles tend to join up and form walls with various shapes. Figure 1 represents the problem space with 35% obstacles. Manhattan distance is used as the heuristic evaluation function. In the problem space, the initial and goal states are respectively denoted by s and t (see Figure 1), and positioned 100 units apart in terms of Manhattan distance. The actual solution length from s to t in this case is 122 units. Note that LRTA* can be applied to any kind of problem spaces. Mazes are used in this evaluation, simply because the search behavior is easy to observe.

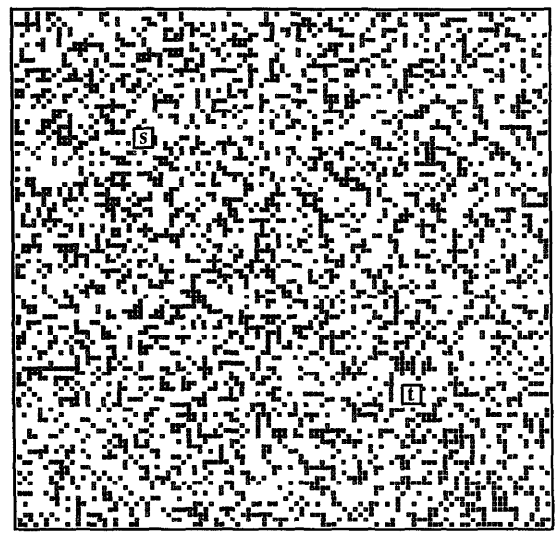

Figure 1: Example maze (35% obstacles)

Figure 2(a) shows the learning efficiency of LRTA* when repeatedly solving the maze in Figure 1. In this figure, the y-axis represents the solution length (the number of moves of the problem solver) at each trial, and the x-axis represents the number of problem solving trials. From Figure 2(a), we can observe the following characteristics:

- The solution length does not monotonically decrease during the convergence process.
- The algorithm continues to search for alternative solutions even after a fairly good solution is obtained, and this often makes the solution length increase again.

In contrast, as shown in Figure 2(b), the learning process of $\varepsilon\delta$-search ($\varepsilon = 0.2$ and $\delta = 2$) is far more stable. The algorithm will be detailed in the next three sections.

Introducing ε-Lower and Upper Bounds

This section introduces two new kinds of estimated costs (*ε-lower bounds* and *upper bounds*), and a method for maintaining their *local consistency*, which means the consistency of estimated costs in explored areas.

Let $h^*(x)$ be the actual cost from state x to the goal, and $h(x)$ be its lower bound. We introduce ε-lower

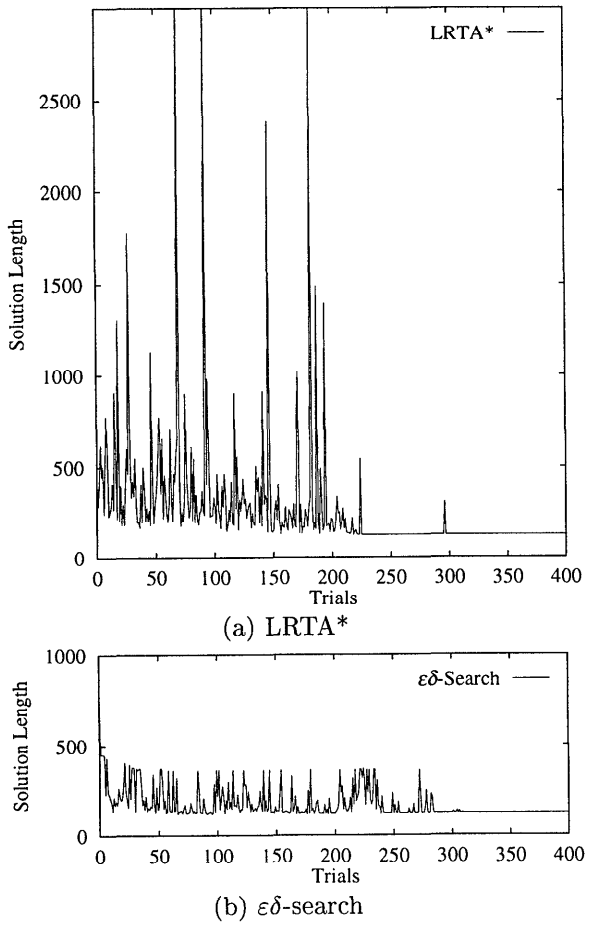

Figure 2: Learning efficiency of LRTA* and $\varepsilon\delta$-search

bound denoted by $h_\varepsilon(x)$, and upper bound denoted by $h_u(x)$. As described in Section 2, LRTA* utilizes an admissible heuristic evaluation function (Manhattan distance for example) to calculate the initial lower bounds. The initial ε-lower bound at each state is set to $(1+\varepsilon)$ times the initial lower bound value given by the heuristic evaluation function. On the other hand, the initial upper bound is set to infinity, i.e., at each state x, $h_u(x) \leftarrow \infty$, while at the goal state t, $h_u(t) \leftarrow 0$.

The actual cost from the state x to the goal (denoted by $h^*(x)$) can be represented by the actual cost from its neighboring state x' to the goal (denoted by $h^*(x')$), that is,

$$\begin{aligned} h^*(x) &= \min_{x'} f^*(x') \\ &= \min_{x'}\{k(x,x') + h^*(x')\}. \end{aligned} \quad (1)$$

From this observation, the following operations are obtained to maintain the local consistency of estimated costs.

$$\begin{aligned} h(x) &\leftarrow \min_{x'} f(x') \\ &= \min_{x'}\{k(x,x') + h(x')\} \end{aligned} \quad (2)$$

$$\begin{aligned} h_\varepsilon(x) &\leftarrow \max\left\{ \begin{array}{l} \min_{x'} f_\varepsilon(x') \\ h_\varepsilon(x) \end{array} \right\} \\ &= \max\left\{ \begin{array}{l} \min_{x'}\{k(x,x') + h_\varepsilon(x')\} \\ h_\varepsilon(x) \end{array} \right\} \end{aligned} \quad (3)$$

$$\begin{aligned} h_u(x) &\leftarrow \min\left\{ \begin{array}{l} \min_{x'} f_u(x') \\ h_u(x) \end{array} \right\} \\ &= \min\left\{ \begin{array}{l} \min_{x'}\{k(x,x') + h_u(x')\} \\ h_u(x) \end{array} \right\} \end{aligned} \quad (4)$$

The lower bound values start from the initial values given by the heuristic evaluation function. Assuming that the heuristic evaluation function is *monotonic*[2], operation (2) preserves monotonicity during the convergence process. Therefore, $h(x)$ never decreases by executing (2), and monotonically increases. Thus the lower bounds can be updated to $\min_{x'} f(x')$, if all neighboring states have been created.

We have to be careful about ε-lower bounds, which do not preserve monotonicity, even though the heuristic evaluation function is monotonic. That means, in general, $h_\varepsilon(x) \leq k(x,x') + h_\varepsilon(x')$ is not always true. This is understandable from the fact that $k(x,x')$ can be ignored when ε is large, while $h_\varepsilon(x) \leq h_\varepsilon(x')$ is not always true. Therefore, to guarantee the monotonic increase of ε-lower bounds, updating should occur only when it increases the ε-lower bounds (see operation (3)).

Similarly, the upper bounds do not preserve monotonicity. The upper bound values start from infinity and monotonically decrease as represented by (4). Therefore, $h_u(x)$ can be updated to $\min_{x'} f_u(x')$ at any time, even when all neighbors have not been created. The value can be updated by (4) with neighboring states that have been created so far.

To maintain the local consistency of ε-lower bounds and upper bounds, we modify the lookahead and consistency maintenance steps of LRTA* as follows.

1. Lookahead:
   For all neighboring states x' of x, calculate $f(x') = k(x,x') + h(x')$, $f_\varepsilon(x') = k(x,x') + h_\varepsilon(x')$ and $f_u(x') = k(x,x') + h_u(x')$ for each neighbor x' of the current state x, where $h(x')$, $h_\varepsilon(x')$, $h_u(x')$ are the current lower, ε-lower and upper bounds of the actual cost from x' to the goal state, and $k(x,x')$ is the edge cost from x to x'.
2. Consistency maintenance:
   Update $h(x)$, $h_\varepsilon(x)$ and $h_u(x)$ based on operations (2), (3) and (4).

[2]This means $h(x) \leq k(x,x') + h(x')$ is satisfied for all x and its neighboring state x' when the first trial starts.

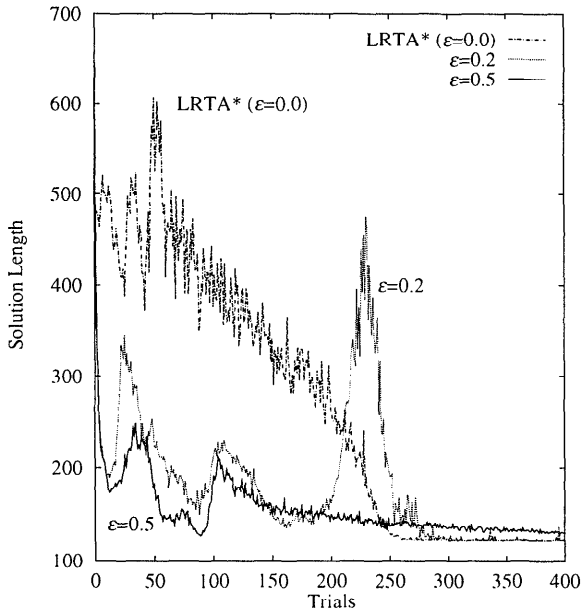

Figure 3: Solution length in ε-search

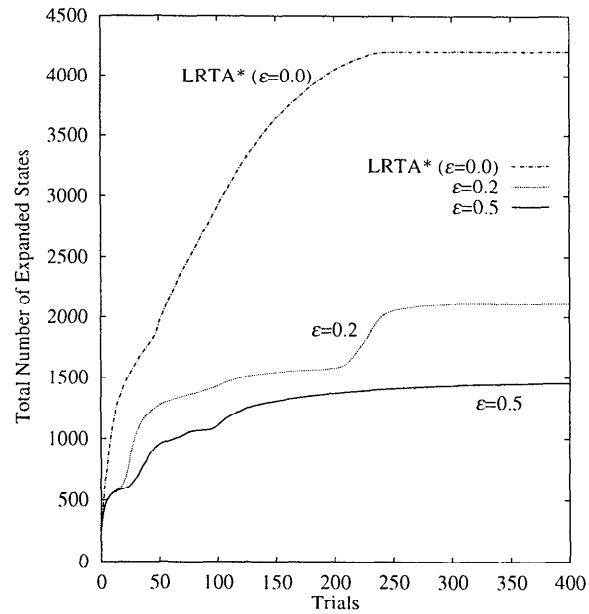

Figure 4: Expanded states in ε-search

Weighted Realtime Search

ε-Search Algorithm

As LRTA* maintains lower bounds, ε-search requires us to maintain ε-lower bounds as described in the previous section. The ε-search algorithm also modifies the action selection step of LRTA* as follows.

3. Action selection:
   Move to neighbor x' with minimum $f_\varepsilon(x')$ value. Ties are broken randomly.

When $\varepsilon = 0$, ε-search is exactly same as LRTA*. The rest of this section shows that, by repeatedly applying ε-search to the same problem, the algorithm converges to a suboptimal solution with $\varepsilon \cdot h^*(s)$ error.

Definition: 1 The path P $(x = x_0, x_1, \ldots, x_{n-1}, x_n = t)$ from state x to goal state t is called *ε-optimal*, when the following condition is satisfied.

$$\sum_{i=0}^{n-1} k(x_i, x_{i+1}) \leq (1+\varepsilon)h^*(x)$$

Theorem: 1 In a finite problem space, with an admissible heuristic evaluation function, through over repeated problem solving trials of ε-search, a path from the initial state s to the goal state t along the minimum value of ε-lower bounds will eventually converge to ε-optimal.

Let us compare ε-search (weighted realtime search) to offline weighted search, which utilizes $f(x) = g(x) + w \cdot h(x)$ as an estimated cost [Pohl, 1970]. In both offline and realtime weighted search, when $w = (1+\varepsilon)$, suboptimal solutions with $\varepsilon \cdot h^*(s)$ error are obtained with reduced search effort. However, the runtime behavior of ε-search is much different from that of offline weighted search: the explored area is not equal to that of offline search, even after infinitely solving the same problem; the completeness in an infinite problem space cannot be guaranteed, even if local consistency is maintained.

ε-Search Performance

Figures 3 and 4 show the evaluation results of repeatedly applying ε-search to the sample maze illustrated in Figure 1. Figure 3 represents the solution length (the number of moves taken by the problem solver to reach the goal state), and Figure 4 displays the number of expanded states (the amount of memory space required to solve the problem). Figures 3 and 4 were created by averaging 300 charts, each of which records a different convergence process. Compared to Figure 2, these figures look smoother and it is easy to grasp search behavior.

Every figure represents the cases of $\varepsilon = 0, 0.2, 0.5$. As previously described, the case of $\varepsilon = 0$ is exactly the same as LRTA*. From these figures, we obtain the following results.

- Figure 4 shows that the number of expanded states decreases as ε increases. The computational cost of convergence is considerably reduced by introducing ε-lower bounds.

- On the other hand, as ε increases, optimal solutions become hard to get. The converged solution length increases up to the factor of $(1+\varepsilon)$. Figure 3 clearly shows that, when $\varepsilon = 0.5$ for example, the algorithm does not converge to an optimal solution.
- Through repeated problem solving trials, the solution length decreases more rapidly as ε increases. However, in the case of $\varepsilon = 0.2$ for example, when the algorithm starts searching for an alternative solution, the solution length irregularly increases (see Figure 3). This shows that, ε-search fails, by itself, to stabilize the solution quality during convergence.

The above evaluation shows that, the learning efficiency of realtime search can be improved significantly by allowing suboptimal solutions. The remaining problem is how to stabilize the solution quality during the convergence process.

Realtime Search with Upper Bounds

δ-Search Algorithm

As LRTA* maintains lower bounds, δ-search maintains upper bounds as well, which can be useful for balancing exploration and exploitation. Our idea is to guarantee the worst case cost by using the upper bounds.

The action selection step of δ-search is described below. Let $c(j)$ be the total cost of j moves performed so far by δ-search, i.e., when the problem solver moves along a path $(s = x_0, x_1, \ldots, x_j = x)$,

$$c(j) = \sum_{i=0}^{j-1} k(x_i, x_{i+1}).$$

Let h_0 be the upper bound value of the initial state s at the time when the current trial starts.

3. Action selection:

   For each neighboring state x' of x, update the upper bound as follows.

$$h_u(x') \leftarrow \min \left\{ \begin{array}{l} k(x', x) + h_u(x) \\ h_u(x') \end{array} \right\} \quad (5)$$

   Move to the neighbor x' that has the minimum $f(x')$ value among the states that satisfy the following condition.

$$c(j) + f_u(x') \leq (1+\delta)h_0 \quad (6)$$

   Ties are broken randomly.

Operation (5) partially performs operation (4) for the neighbor x'. This is needed because the monotonicity of upper bounds cannot be guaranteed even by the consistency maintenance steps. Operation (5) enables the problem solver to move towards unexplored areas.

When $\delta = \infty$, since condition (6) is always satisfied, δ-search is exactly the same as LRTA*. When $\delta \neq \infty$, δ-search ensures that the solution length will be less than $(1+\delta)$ times the upper bound value of the initial state at the time when the current trial starts. The following theorem confirms the contribution of δ-search in stabilizing the solution quality.

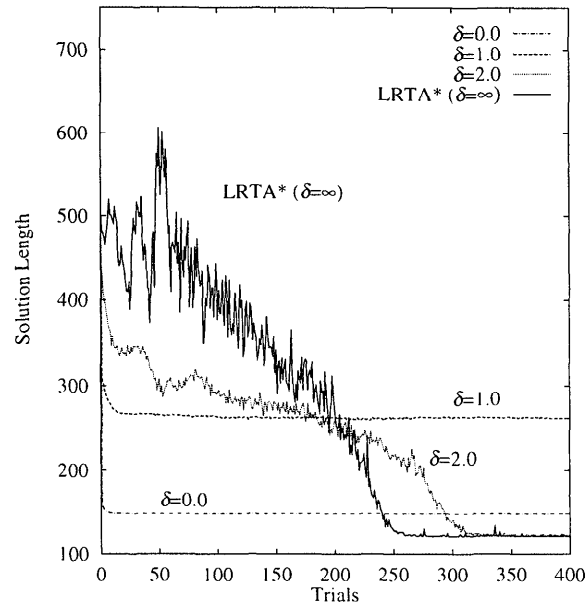

Figure 5: Solution length in δ-search

Theorem: 2 When $\delta \geq 0$, the solution length of δ-search cannot be greater than $(1+\delta)h_0$, where h_0 is the upper bound value of the initial state s at the time when the current trial starts.

δ-Search Performance

δ-search can guarantee the worst case solution length. Note that, however, to take this advantage, the upper bound value of the initial state s must be updated reflecting the results of the previous trial. Therefore, in the following evaluation, each time the problem is solved, we back-propagate the upper bound value from the goal state t to the initial state s along the solution path.

Figures 5 and 6 show the evaluation results of repeatedly applying δ-search to the sample maze illustrated in Figure 1. Figure 5 represents the solution length (the number of moves of the problem solver to reach the goal state), and Figure 6 represents the number of expanded states (the amount of memory space required to solve the problem). Figures 5 and 6 were created by averaging 50 charts, each of which records a different process of convergence.

Every figure shows the cases of $\delta = 0, 1.0, 2.0, \infty$. In the case of $\delta = 0$, the algorithm is satisfied with the solution yielded by the first trial, and thus exploration is not encouraged afterward. In the case of $\delta \geq 2$, the solution path converges to the optimal path. When

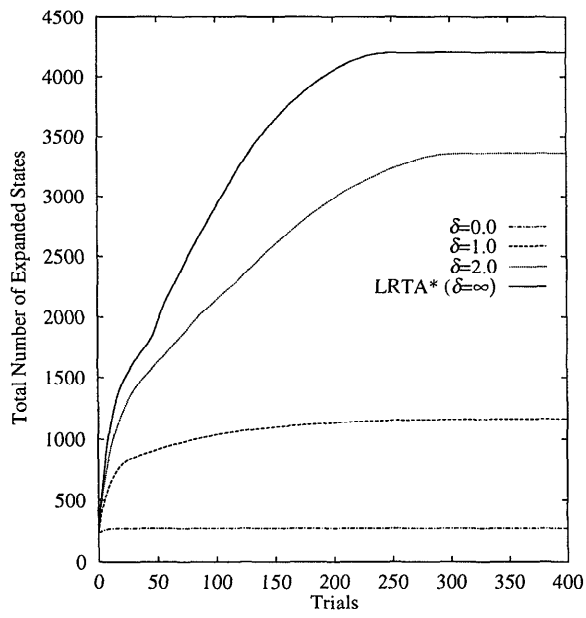

Figure 6: Expanded states in δ-search

$\delta = \infty$, the algorithm behaves just as LRTA*. From these figures, we obtain the following results.

- Figure 6 shows that the number of expanded states decreases as δ decreases. This happens because the explored area is effectively restricted by balancing exploration and exploitation. However, the decrease of learning amount does not always mean that convergence speed is increased. Figure 5 shows that, when $\delta = 2$, the convergence is slower than that when $\delta = \infty$. This is because δ-search restricts the amount of learning in each trial.
- Figure 5 shows that, as δ decreases, the solution length is dramatically stabilized. On the other hand, as δ decreases, it becomes hard to obtain optimal solutions. For example, when $\delta = 1.0$, the algorithm does not converge to the optimal solution, and the solution quality is worse than the case of $\delta = 0$. This is because the δ value is not small enough to inhibit exploration, and not large enough to find a better solution.

Unlike ε-search, δ-search eventually converges to an optimal solution when an appropriate δ value is selected. To find a better solution than those already obtained, however, δ-search requires the round trip cost of the current best solution, i.e., δ should not be less than 2.

We now merge the ideas of ε- and δ-search. The combined algorithm is called *$\varepsilon\delta$-search*, which modifies δ-search to move not along the minimum lower bounds but along the minimum ε-lower bounds. $\varepsilon\delta$-search is intended to reduce the amount of learning by ε-search, and to stabilize the solution quality by δ-search. The results are displayed in Figure 2(b).

Conclusion

Previous work on realtime search focused on the problem solving performance of the first trial, and did not pay much attention to the learning process. This paper reveals the convergence issues in the learning process of realtime search, and overcomes the deficits.

By introducing ε-lower bounds and upper bounds, we showed that the solution quality of realtime search can be stabilized during convergence. We introduced two new realtime search algorithms: ε-search (weighted realtime search) is intended to avoid consuming time and memory for meaningless solution refinement; δ-search (realtime search with upper bounds) provides a way to balance the solution quality and the exploration cost. The ε- and δ-search algorithms can be combined easily. The effectiveness of ε- and δ-search was demonstrated by solving randomly generated mazes.

References

[Barto *et al.*, 1995] A. G. Barto, S. J. Bradtke and S. P. Singh, "Learning to Act Using Real-Time Dynamic Programming," *Artificial Intelligence*, Vol. 72, pp. 81-138, 1995.

[Chimura and Tokoro, 1994] F. Chimura and M. Tokoro, "The Trailblazer Search: A New Method for Searching and Capturing Moving Targets," *AAAI-94*, pp. 1347-1352, 1994.

[Hart *et al.*, 1986] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Trans. SMC*, Vol. 4, No. 2, pp.100-107, 1968.

[Ishida and Korf, 1995] T. Ishida and R. E. Korf, "A Moving Target Search: A Real-Time Search for Changing Goals," *IEEE Trans. PAMI*, Vol. 17, No. 6, pp. 609-619, 1995.

[Ishida, 1996] "Real-Time Bidirectional Search: Coordinated Problem Solving in Uncertain Situations," *IEEE Trans. PAMI*, Vol. 18, 1996.

[Korf, 1990] R. E. Korf, "Real-Time Heuristic Search", *Artificial Intelligence*, Vol. 42, No. 2-3, pp. 189-211. 1990.

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

[Pohl, 1970] I. Pohl, "First Results on the Effect of Error in Heuristic Search," *Machine Intelligence*, Vol. 5, pp. 219-236, Edinburgh University Press, 1970.

[Russell and Wefald, 1991] S. Russell and E. Wefald, *Do the Right Thing,* The MIT Press, 1991.