

# Moving Target Search

**Toru Ishida**

NTT Communications and  
Information Processing Laboratories  
1-2356, Take, Yokosuka,  
238-03, JAPAN  
ishida%nttkb.ntt.jp@relay.cs.net

**Richard E. Korf**

Computer Science Department  
University of California, Los Angeles  
Los Angeles, Ca. 90024, U.S.A.  
korf@lanai.cs.ucla.edu

## Abstract

We consider the case of heuristic search where the location of the goal may change during the course of the search. For example, the goal may be a target that is actively avoiding the problem solver. We present a moving target search algorithm (MTS) to solve this problem. We prove that if the average speed of the target is slower than that of the problem solver, then the problem solver is guaranteed to eventually reach the target. An implementation with randomly positioned obstacles confirms that the MTS algorithm is highly effective in various situations.

## 1 Introduction

All existing heuristic search algorithms assume that the goal state is fixed and does not change over the course of the search. For example, in the problem of driving from the current location to a desired goal location along a network of roads, it is assumed that the goal location is fixed and does not change during the drive. In this paper, we relax the assumption of a fixed goal, and allow it to move over time. In the road navigation example, instead of driving to a particular address, the task may be to reach another vehicle which is in fact moving as well.

There is no assumption that the target will eventually stop, but the goal is achieved when the position of the problem solver and the position of the target coincide. In order to have any hope of success in this task, the problem solver must be able to move faster than the target. Otherwise, the target could evade the problem solver indefinitely, even in a finite problem space, merely by avoiding being trapped in a dead-end path.

Existing search algorithms can be divided into two classes: off-line and real-time. Off-line algorithms, such as A\*[1], compute an entire solution path before executing the first step in the path. Real-time algorithms, such as Real-Time-A\*[2], perform sufficient computation to determine a plausible next move of the problem solver, execute that move in the physical world, then perform further computations to determine the following move, etc., until the goal is reached. These algorithms do not find optimal solutions, but can commit to actions in constant time per move.

If one were to use an off-line algorithm such as A\* for the moving goal problem, by the time an optimal path was found to the current position of the goal, and the path was executed, the goal would have moved to a new position. If one were then to repeat the algorithm with the new current state and the new goal position, one could follow the target. In fact, this approach would work if the time to perform the search was negligible. However, we assume that search takes time, and thus even if the problem solver could move faster than the target, if it has to stop periodically to plan a new path, its speed advantage over the target may be lost.

Therefore, we must consider real-time algorithms that can rapidly react to each move of the target, and always make the best move toward the current location of the target. The next section briefly reviews previous work on such algorithms for the case of a fixed goal location.

## 2 Previous Work: LRTA\*

Learning-real-time-A\*(LRTA\*[2]) is a real-time search algorithm for fixed goal states. As such, it commits to individual moves in constant time, and interleaves the computation of moves with their execution. It builds and updates a table containing heuristic estimates of the distance from each state in the problem space to the fixed goal state. Initially, the entries in the table correspond to heuristic evaluations, or zero if none are available, and are assumed to be lower bounds on actual distance. Through repeated exploration of the space, however, more accurate values are learned until they eventually equal the exact distances to the goal.

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

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

The reason for updating the value of  $h(x)$  is that since

the  $f(x')$  values represent lower bounds on the actual distance to the goal through each of the neighbors, the actual distance 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 a goal, LRTA\* is *complete* in the sense that it will eventually reach a goal. Furthermore, if the initial heuristic values are lower bounds on the actual values, then over repeated problem-solving trials, the values learned by LRTA\* will eventually converge to their exact values along every optimal path to the goal. LRTA\*, however, requires that the position of the goal state be fixed.

### 3 Problem Formulation

In this section we give a precise characterization of the moving target search problem. We represent the problem space as a connected graph. The graph is undirected, allowing motion of either the problem solver or the target along any edge in either direction. To simplify the following discussions, we assume that all edges in the graph have unit cost. Given a graph with non-uniform but rational edge costs, we can convert it to a graph with unit edge costs, without changing the topology of the graph, by choosing the unit to be the lowest common denominator of the edge costs, and then inserting into any edge with a larger value enough intermediate nodes of degree two so that each edge has unit cost.

There is an initial position of the problem solver and an initial position of the target. We assume the problem solver and the target move alternately, and can traverse at most one edge in a single move. The problem solver has no control over the movements of the target. The task is accomplished when the problem solver and the goal occupy the same node. The problem is further characterized by the following constraints.

*Speed of the problem solver and the target:*

Given unit edge costs, we reduce the speed of the target by assuming that periodically the target will make no move, and remain at its current position. Thus, the target can move at the same speed as the problem solver for only so long before losing a step by skipping a turn to move. We will slightly relax this assumption in Section 7.

*Available information about the target and problem space:*

We start with the assumption that the problem solver always knows the target's position, but will generalize this later in Section 7. We do not assume that the problem solver knows the topology of the problem space in the vicinity of the target position, nor the topology in the vicinity of the problem solver, other than the locations of the immediate neighbors of the current location of the problem solver. In other words, the problem solver does not have a map of the problem space.

*A heuristic static evaluation function:*

Another assumption is that there also exists a heuristic static evaluation function that returns an

estimate of the distance between any pair of states. The only constraint placed on the static evaluation function is that it be admissible, meaning it never overestimates the actual distance between any pair of points. For example, a function returning the Euclidean distance in the plane is admissible, but so is a heuristic function that returns zero for every pair of points. Thus, we do not assume that the heuristic static evaluation function provides any useful information.

### 4 Moving Target Search Algorithm

In this section, we present the algorithm, called *moving-target search* (MTS). It is a generalization of LRTA\* to the case where the goal can move. MTS must acquire heuristic information for each goal location. Thus, MTS maintains a matrix of heuristic values, representing the function  $h(x, y)$  for all pairs of states  $x$  and  $y$ . Conceptually, all heuristic values are read from this matrix, which is initialized to the values returned by the static evaluation function. Over the course of the search, these heuristic values are updated to improve their accuracy. In practice, we only store those values that differ from their static values. Thus, even though the complete matrix may be very large, it may be quite sparse, and sparse matrix techniques may be used for time and space efficiency.

There are two different events that occur in the algorithm: a move of the problem solver, and a move of the target, each of which may be accompanied by the updating of a heuristic value. The problem solver and target alternate moves, and heuristic values are updated as necessary, until the position of the problem solver and the target are equal. In the description below,  $x$  is the current position of the problem solver, and  $y$  is the current position of the target.

*When it is the problem solver's turn to move,* it calculates  $f(x', y) = h(x', y) + 1$  for each of its neighbors  $x'$  of  $x$ , and chooses a neighbor  $w$  with the smallest  $f$  value to move to. Ties are broken randomly. If  $f(w, y) > h(x, y)$ , then  $h(x, y)$  is updated to  $f(w, y)$ . The reason is that since any path from  $x$  to  $y$  must go through one of its neighbors  $x'$ , the cost of the minimum path from  $x$  to  $y$  must be as large as the minimum cost path through any of its neighbors  $x'$ .

*When it is the target's turn to move,* the problem solver observes its move from  $y$  to its neighbor  $y'$ , calculates  $h(x, y')$ , and compares it to  $h(x, y)$ , where  $x$  is the current state of the problem solver. If  $h(x, y') > h(x, y) + 1$ , then  $h(x, y)$  is updated to  $h(x, y') - 1$ . The reason is that since the old and new positions of the target are at most one unit apart, the distance to the old position must be at least as large as the distance to the new position minus one unit.

The algorithm is described more succinctly below.

*When the problem solver moves from  $x$ :*

1. Calculate  $f(x', y) = h(x', y) + 1$  for each neighbor  $x'$  of  $x$ .

2. Move to the neighbor with the minimum  $f(x', y)$ . Ties are broken randomly.
3. Update the value of  $h(x, y)$  as follows:

$$h(x, y) \leftarrow \max \left\{ \begin{array}{l} h(x, y) \\ \min_{x'} \{h(x', y) + 1\} \end{array} \right\}$$

When the target moves from  $y$  to  $y'$ :

1. Calculate  $h(x, y')$  for the target's new position  $y'$ .
2. Update the value of  $h(x, y)$  as follows:

$$h(x, y) \leftarrow \max \left\{ \begin{array}{l} h(x, y) \\ h(x, y') - 1 \end{array} \right\}$$

It should be emphasized that this is only one algorithm to solve this problem. In particular, it was designed to perform the minimum amount of computation necessary to guarantee success. Performance of the algorithm could be considerably improved by updating more heuristic values with each move. For example, the update of  $h(x, y)$  performed with each move of the problem solver could also be similarly applied to  $h(x, z)$  for values of  $z$  other than the target position  $y$ . The drawback of this is that such computation requires time and memory, and may not be of any use if the target never visits position  $z$ .

## 5 Completeness of Moving Target Search

In this section, we prove that a problem solver executing MTS is guaranteed to eventually reach the target, as long as the target periodically skips a move. We begin by showing that the updates preserve admissibility of the heuristic values. Then, we define a positive quantity called *heuristic disparity*, which is a combination of the difference between the current heuristic values and their exact values, and the current estimated heuristic distance from the problem solver to the target. We show that in any pair of moves of the problem solver and the target, this quantity can never increase, and decreases whenever the target does not move. Since it cannot be negative, and if it ever reaches zero the problem is solved, the algorithm must eventually terminate successfully.

Lemma:

Updating in MTS preserves admissibility of heuristic values.

Proof:

Assume that the current values are admissible. There are two types of updates, those occurring with moves of the problem solver, and those accompanying moves of the target.

In the case of problem solver moves, if  $h(x, y)$  is updated, it is set equal to  $h(w, y) + 1$ , where  $h(w, y)$  is the minimum of  $h(x', y)$  for all neighbors  $x'$  of  $x$ . Since any path from  $x$  to  $y$  must go through one of its neighbors  $x'$ , and each neighbor is one unit away, the cost of the minimum path from  $x$  to  $y$  must be one greater than the minimum cost path from any

neighbor  $x'$  to  $y$ . If  $h(x', y)$  is a lower bound on the cost from  $x'$  to  $y$ , then  $h(w, y)$  is a lower bound on the cost from any neighbor  $x'$  to  $y$ , and hence  $h(w, y) + 1$  must be a lower bound on the cost from  $x$  to  $y$ . Thus, this update preserves admissibility.

In the case of moves of the target, if  $h(x, y)$  is updated, it is set equal to  $h(x, y') - 1$ , where  $y'$  is the new position of the target. Since  $y$  is at most one unit away from  $y'$ , the distance from  $x$  to  $y$  can only differ from the distance from  $x$  to  $y'$  by at most one unit. Thus if  $y'$  is at least  $h(x, y')$  units from  $x$ , then  $y$  must be at least  $h(x, y') - 1$  units from  $x$ . Thus, this update preserves admissibility as well.

By induction on the total number of updates, updating preserves admissibility.

Theorem:

In a finite problem space, in which a path exists between every pair of nodes, starting with non-negative admissible initial heuristic values, a problem solver executing MTS will eventually reach the target, if the target periodically skips moves.

Proof:

Define the *heuristic error* at a given state of the computation as the sum over all pairs of states  $a$  and  $b$  of  $h^*(a, b) - h(a, b)$  where  $h^*(a, b)$  is the length of the shortest path between  $a$  and  $b$ , and  $h(a, b)$  is the current heuristic value. Define the *heuristic distance* as  $h(x, y)$ , the current heuristic value between the current state of the problem solver,  $x$ , and the current state of the target,  $y$ . Define the *heuristic disparity* as the sum of the heuristic error and the heuristic difference.

First, consider a move of the problem solver from  $x$  to  $x'$ . If  $h(x', y) < h(x, y)$ , then no update occurs, and the heuristic distance from problem solver to target after the move,  $h(x', y)$ , is at least one unit less than the heuristic distance between them before the move,  $h(x, y)$ . Thus, the move causes the heuristic distance and hence the heuristic disparity to decrease by at least one unit. Conversely, if  $h(x', y) \geq h(x, y)$ , then  $h(x, y)$  is increased to  $h(x', y) + 1$ . As a result, the total heuristic error decreases by  $h(x', y) + 1 - h(x, y)$ , while the heuristic distance increases by  $h(x', y) - h(x, y)$ , for a net decrease of one unit in the heuristic disparity. Thus, in either case, the heuristic disparity decreases by at least one unit.

Now consider a move of the target from  $y$  to  $y'$ . If  $h(x, y') \leq h(x, y) + 1$ , then there is no update and the heuristic difference and the heuristic disparity increase by at most one unit. Conversely, if  $h(x, y') > h(x, y) + 1$ , then  $h(x, y)$  is increased to  $h(x, y') - 1$ . Thus, the total heuristic error decreases by  $h(x, y') - 1 - h(x, y)$ , but the heuristic difference increases by  $h(x, y') - h(x, y)$ , for a total increase in the heuristic disparity of one unit. In either case, the heuristic disparity increases by at most one unit.

Since a move by the problem solver always decreases the heuristic disparity by one unit, and a move by the target can only increase it by at most one unit,

in a pair of moves by problem solver and target the heuristic disparity cannot increase. When the target eventually skips a move, the heuristic disparity will decrease by at least one unit. With repeated skipped moves by the target, heuristic disparity must continue to decrease over time.

Since there is a path between every pair of states,  $h^*(a, b)$  is finite for all  $a$  and  $b$ , and since there are a finite number of states, total heuristic error is finite. Similarly, since all heuristic values are admissible and hence finite, heuristic difference is also finite. Therefore, heuristic disparity is finite. Since all heuristic values are admissible, total heuristic error is non-negative, and since heuristic values are non-negative, heuristic difference is non-negative. Thus, heuristic disparity is non-negative. Either the algorithm terminates successfully, or the heuristic disparity reaches zero, meaning that both total heuristic error and heuristic difference are zero. At that point, all heuristic values are exact, and the heuristic distance between problem solver and target is zero, meaning they are in the same location. This constitutes a successful termination.

## 6 Efficiency of Moving Target Search

### 6.1 Space Complexity

An upper bound on the space complexity of moving target search is  $N^2$ , where  $N$  is the number of states in the problem space. The reason is that in the worst case, heuristic values between every possible pair of states would have to be stored. In practice, however, the space complexity will usually be much lower. Since there exists a function which computes the static heuristic value between any pair of points, it is only necessary to store in memory those values that differ from the static values. When a heuristic value is needed, if it is not in the table, then it is computed by the function. Since at most only one update occurs per move of the problem solver or target, we never need more memory than the total number of moves of problem solver and target combined. In fact, we may need considerably less since an update does not necessarily occur with every move. Thus, the overall space complexity is the minimum of  $N^2$  and the total number of moves of the problem solver and target.

### 6.2 Time Complexity

The worst-case time complexity of MTS is  $N^3$ , where  $N$  is the number of states in the problem space. This can be obtained by investigating the maximum heuristic disparity. The total heuristic error is upper bounded by  $N^3$ , because there are  $N^2$  heuristic values between every possible pair of states, and the maximum heuristic value of each state is  $N$ . On the other hand, if no updates of heuristic values occur, then the maximum number moves of the problem solver to reach the target is  $N/S$ , where  $S$  is the fraction of moves that are skipped by the target. This is because the maximum heuristic difference is  $N$ , and it must decrease by at least one every time the target skips a move. Thus the overall time complexity, which is the sum of these two terms, is still  $N^3$  in the

worst case. The worst case assumes initial heuristic values of zero everywhere. In this case, the learning cost for correcting heuristic error is the major component of the time complexity of MTS.

### 6.3 Learning Over Multiple Trials

Over the course of a single problem solving episode, MTS gradually learns more accurate heuristic values, until the target is reached. If we choose new initial states for both the problem solver and the target, but start with the set of heuristic values left over from the previous problem solving episode, the knowledge learned from previous trials is immediately transferable to the new episode, with the result being better performance than if we started with the initial heuristic values. If we continue to preserve heuristic values from one episode to another, eventually the heuristic values will converge to their exact values for every pair of states. At that point, the problem solver will always make the best possible moves in pursuing the target, and the time complexity of an individual trial reduces to  $N/S$ . Of course, for complete learning,  $N^2$  space is required to store all heuristic values.

## 7 Relaxing Some Constraints on MTS

### 7.1 Speed of the Problem Solver and the Target

In previous sections, we assumed that the problem solver can move faster than the target, but this is not strictly necessary. A weaker condition is that the target can move as fast as the problem solver, but occasionally makes errors in avoiding the problem solver. For this purpose, we define an apparently optimal move of the target to be one that increases the distance from the problem solver by at least one unit, according to the heuristic values. If the heuristic values are exact, then an apparently optimal move is in fact an optimal move. However, if there is error in the heuristic values, then an apparently optimal move may or may not be in fact optimal, and an optimal move may or may not appear to be optimal. In order for it to be possible for the problem solver to catch the target, we assume that the target periodically makes apparently suboptimal moves.

Even in this case, a problem solver executing MTS will still eventually reach the target. This is because an apparently suboptimal move by the target is one that does not increase the heuristic difference. Thus, there is no update and the heuristic disparity does not increase. An apparently suboptimal move of the target, coupled with a move of the problem solver, decreases the heuristic disparity by at least one unit. Thus, the heuristic disparity continues to decrease over time.

### 7.2 Available Information about the Target

We previously assumed that the problem solver always knows the position of the target. This assumption can also be relaxed by only requiring that the problem solver know the position of the target at some point before the problem solver reaches the last known position of the target. This generalization allows the problem solver to

Figure 1: Sample Tracks of MTS

temporarily lose sight of the target, and thus may be more practical in real applications.

Suppose the problem solver and the target have moved at most  $t$  times between the target being observed at  $y$  and then at  $z$ . In this case, the updating of the heuristic values is generalized as follows.

1. Calculate  $h(x, z)$  for the target's new position  $z$ .
2. Update the value of  $h(x, y)$  as follows:

$$h(x, y) \leftarrow \max \left\{ \begin{array}{l} h(x, y) \\ h(x, z) - t \end{array} \right\}$$

The update is admissible, because  $y$  is at most  $t$  units away from  $z$ , and the distance between  $x$  and  $y$  can only differ from the distance between  $x$  and  $z$  by at most  $t$  units. Thus if  $z$  is at least  $h(x, z)$  units from  $x$ , then  $y$  must be at least  $h(x, z) - t$  units from  $x$ .

To prove the completeness of the generalized MTS, we must slightly modify the proof in Section 5. If  $h(x, z) \leq h(x, y) + t$ , then there is no update and the heuristic difference and the heuristic disparity increase by at most  $t$  units. Conversely, if  $h(x, z) > h(x, y) + t$ , then  $h(x, y)$  is increased to  $h(x, z) - t$ . Thus, the total heuristic error decreases by  $h(x, z) - t - h(x, y)$ , but the heuristic difference increases by  $h(x, z) - h(x, y)$ , for a total increase in the heuristic disparity of  $t$  units. In either case, the heuristic disparity increases by at most  $t$  units. Since a move by the problem solver always decreases the heuristic disparity by at least one unit, the heuristic disparity decreases by  $t$  units in the period between when the target is observed at  $y$  and then at  $z$ . On the other hand, a move by the target from  $y$  to  $z$  can only increase it by at most  $t$  units. Thus, in the combined sequence of  $t$  moves by the problem solver and the target, the heuristic disparity cannot increase.

## 8 Experimental Evaluation

We have implemented MTS in a rectangular grid problem space ( $100 \times 100$ ) with randomly positioned obstacles. We allow motion along the horizontal and vertical dimensions, but not along the diagonal. Interesting target behavior was obtained by allowing a human user to indirectly control the motion of the target. The user moves a cursor around the screen using a mouse, and the target always moves toward the current position of the cursor, using static heuristic values for guidance. Figure 1 shows the experimental setup along with sample tracks of the target and problem solver with manually placed obstacles. In Figure 1(a), the user's task is to avoid the problem solver, which is executing MTS, for as long as possible, while in Figure 1(b), the task is to meet the problem solver as quickly as possible.

### 8.1 Experiments on Different Target Response Modes

Figure 2 illustrates the search cost represented by the total number of moves of the problem solver for various response strategies of the target. The response modes are: 1) the target actively avoids the problem solver (*Avoid*), 2) the target moves randomly (*Random*), 3) the target moves cooperatively to try to meet the problem solver (*Meet*), and 4) the target remains stationary (*Stationary*). In *Meet* and *Avoid*, the target also performs MTS: in *Meet*, the target moves to decrease the heuristic distance to the problem solver, while in *Avoid*, the target moves to increase the heuristic distance.

The speed of the target is set to 80% of the problem solver. In other words, it skips one of every five moves. To erase problem space boundaries, we formed a torus by connecting the opposite boundaries. The problem solver and the target are initially positioned as far apart

Figure 2: Performance with Different Target Response Modes

as possible in the torus, i.e., 100 units in Manhattan distance. Obstacles are randomly positioned. For example, an obstacle ratio of 20% means 2000 junctions in the  $100 \times 100$  grid are randomly replaced by obstacles. With high obstacle ratios (more than 20%), obstacles join up and form walls with various shapes. When the ratio reaches 40%, the obstacles tend to disconnect the problem space, separating the target from and the problem solver. Euclidean distance is used as the heuristic static evaluation function. The number of moves in the figures are obtained by averaging 100 trials.

## 8.2 Experiments on Different Target Response Modes

As shown in Figure 2, with relatively few obstacles, the target that is easiest to catch is one that is trying to *Meet* the problem solver, and the most difficult target to catch is one that is trying to *Avoid* the problem solver, as one would expect. When the obstacles become more numerous, however, it becomes harder to catch a target making *Random* moves and one that is trying to *Meet* the problem solver, than a target trying to *Avoid* the problem solver or a stationary target. At first, this result seems counterintuitive. If one is trying to avoid a faster pursuer as long as possible, however, the best strategy is not to run away, but to hide behind obstacles. Both *Meet* and *Random* approximate this strategy better than

Figure 3: Performance with Various Problem Solver Sensitivities

*Avoid*. The reason *Avoid* is more effective than remaining stationary is that *Avoid* makes the problem solver spend more time learning new heuristic values.

## 8.3 Experiments on the Problem Solver's Sensitivity

As discussed in Section 7.2, the generalized MTS algorithm allows the problem solver to temporarily lose sight of the target. In other words, even if the problem solver always knows the target's location, it can ignore some of this information. Can this improve the performance of the problem solver?

One possibility is for the problem solver to always keep track of the current position of the target, or (*High*) sensitivity. Another possibility is that the problem solver only observes the target position when it reaches the previous position of the target, or (*Low*) sensitivity. We also implemented a approach called *Tracking*, in which MTS is not utilized: the problem solver first moves to the initial position of the target, keeping track of all moves made by the target in the meantime, and once the initial state of the target is reached, the problem solver simply follows the target by repeating each of its moves. While this approach can be applied only when the problem solver always knows the target's position, it is worth comparing it to MTS.

Figure 3 shows the empirical results, in which the tar-

get behaves in the *Avoid* mode. Experiments were done in the same setting as shown in Figure 2. The *Tracking* approach works relatively well when the obstacle ratio is low, while as the ratio increases, this approach becomes the worst.

When the obstacle ratio is low, *High* sensitivity is efficient, while as the ratio increases, *Low* sensitivity performs better. It says that in the presence of obstacles, pursuit behavior is improved by only sampling the position of the target periodically, and ignoring the intermediate moves. One possible explanation for this is that when the distance between the target and problem solver is large, trying to react to each individual move of the target is counterproductive, and it is better to simply move toward the general vicinity of the target.

## 9 Conclusions and Further Work

We have presented an algorithm for reaching a goal that changes position over time. The main property necessary to guarantee success is that the target moves slower than the problem solver. We have proven that under this condition, the problem solver will eventually reach any target. The algorithm has been implemented and tested in the cases of pursuing a fleeing target, reaching a randomly moving target, and meeting a cooperatively moving target. An empirical result shows that in pursuing a fleeing target, better performance is obtained by ignoring some moves of the target and only sampling its position periodically.

More generally, this work can be viewed as a case study of problem solving in a dynamic and unpredictable environment. In this case, the unpredictability stems from the motion of the goal. However, additional uncertainty could be introduced without any modifications of the algorithm. For example, new obstacles could be dynamically added to the space during the course of the search. While this will cause performance to degrade initially, the existence of these new obstacles will eventually be reflected in the learned heuristic values.

### Acknowledgment

The authors wish to thank Tsukasa Kawaoka and Ryohei Nakano for their support during this work at NTT Laboratories. This research benefitted from discussions with Kazuhiro Kuwabara, Yoshiyasu Nishibe, and Makoto Yokoo. The bulk of this research was performed while the second author was a visitor at, and supported by, NTT Laboratories. Additional support to the second author was provided by an NSF Presidential Young Investigator Award, NSF grant IRI-8801939, and a grant from Rockwell International.

### References

- [1] P. E. Hart, N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, SSC-4, No. 2, pp.100-107, 1968.
- [2] R. E. Korf, "Real-Time Heuristic Search", *Artificial Intelligence*, Vol. 42, No. 2-3, March 1990, pp. 189-211. 1990.
- [3] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, Mass., 1984.