Moving-Target Search: A Real-Time Search for Changing Goals

Toru Ishida, *Member, IEEE,* and Richard E. Korf

***Abstract*—We consider the case of heuristic search where the goal may change during the course of the search. For example, the goal may be a target that actively avoids 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 in a connected problem space.**

The original MTS algorithm was constructed with the minimum operations necessary to guarantee its *completeness*, and hence is not very efficient. To improve its efficiency, we introduce ideas from the area of *resource-bounded planning* into MTS, including

1) *commitment to goals*, and

2) *deliberation for selecting plans*.

Experimental results demonstrate that the improved MTS is 10 to 20 times more efficient than the original MTS in uncertain situations.

***Index Terms*—Search, real-time, problem solving, learning, moving target, deliberation, reactiveness, commitment.**

I. INTRODUCTION

ALMOST all existing heuristic search algorithms assume that the goal state is fixed and does not change during the course of the search. For example, in the problem of a robot navigating from its current location to a desired goal location, it is assumed that the goal location remains stationary. In this paper, we relax this assumption, and allow the goal to change during the search. In the robot example, instead of moving to a particular fixed location, the robot's task may be to reach another robot which is in fact moving as well. The target robot may be cooperatively trying to reach problem-solving robot, it may be actively avoiding the problem-solving robot, or it may be indifferent and simply going about other business. There is no assumption that the target robot will eventually stop, but the goal is achieved when the position of the problem-solving robot and the position of the target robot coincide. In order to guarantee success in this task, the problem solver must be able to move faster than the target. Otherwise, the target could evade the problem solver infinitely, 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* [11], compute an entire solution path before executing the first step in the path. Real-time algorithms, such as Real-Time-A* and Learning-Real-Time-A* [13], perform sufficient computation to determine a plausible next move of the problem solver, execute that move in the physical world, then perform further computation 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. Thus, we propose a real-time *moving-target search* (MTS) algorithm. The MTS algorithm is *complete* in the sense that it will eventually reach the target, assuming a finite connected problem space, and that the speed of the problem solver is faster than that of the target.

We first implement MTS with the minimum operations necessary to guarantee completeness. The resulting algorithm consists of a pair of steps, which are repeatedly performed in alternation. The first step is incremental learning of the estimated distance between the problem solver and the target, and the second step moves the problem solver toward the target. As a result, MTS is *reactive*, i.e., capable of performing each move in constant time, but it is not very efficient. Various experiments show that the efficiency of the algorithm decreases rapidly as *uncertainty* increases. This is because, in uncertain situations, the heuristic function does not return sufficiently accurate information, and the learning cost becomes prohibitive.

To improve the efficiency of MTS, we introduce ideas from the planning literature. In this area, researchers have recently focused on dynamically changing environments, where agents are *resource-bounded* in constructing plans. Georgeff and Lansky [8] have studied agent architectures to cope with environmental changes. Cohen and Levesque [3] have defined the notion of *commitment* as a persistent goal. Kinny and Georgeff [12] quantitatively evaluated how the *degree of commitment*

Manuscript received Jan. 31, 1994; revised Mar. 15, 1995. Recommended for acceptance by Linda Shapiro.

T. Ishida is with the Dept. of Information Science, Kyoto University, Kyoto 606-01, Japan, e-mail ishida@kuis.kyoto-u.ac.jp

R.E. Korf is with the Computer Science Dept., University of California at Los Angeles, Los Angeles, CA 90024, e-mail korf@cs.ucla.edu.

IEEECS Log No. P95088.

affects agents' performance. Durfee and Lesser [6] performed a similar evaluation in multiagent environments. The role of *deliberation* has been investigated by Bratman et al. [1]. Pollack and Ringuette [15] proposed an experimental environment called *Tileworld* and have quantitatively evaluated the *tradeoff between deliberation and reactiveness*, which has been discussed in the planning community [15], [16]. These notions are introduced into MTS to improve its efficiency. The primary motivation behind this research, in addition to the inherent interest in the problem itself, is as a case study of problem solving in a dynamic, uncertain, and unpredictable domain. We believe this general area is of great importance. One approach to research in this area is to introduce these elements to the classical problem-solving framework one at a time, and in a carefully controlled manner. Thus, allowing the goal to change during the course of a search represents an initial step in a research plan toward this more general and important research issues. Much of this work has previously appeared in [9], [10].

II. Previous Work

A. Finding a Moving Target

There is a considerable body of work in the operations research literature on the problem of searching for a moving target.[1] See, for example [2], [4], [7], [17], [18], and the literature cited therein. In that work, the location of the target is unknown to the problem solver, but it is assumed that the motion of the target is governed by a Markov process, the parameters of which are known to the problem solver. The task of the problem solver is to plan a sequence of sensory probes into the problem space in order to maximize the probability of detecting the target in a given number of probes, but not necessarily to move to the position of the target. In general, obstacles are not present in the problem space, and there can be no guarantees of finding the target, since its location is unknown and it keeps moving. In contrast, we assume that the location of the target is known to the problem solver, at least periodically, and that the task of the problem solver is to plan and execute a path to the target in the presence of obstacles.

B. LRTA*

Here we briefly review previous work on real-time search algorithms for the case of a fixed goal location, in particular Learning-Real-Time-A*(LRTA*) [13]. LRTA* 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 come from a heuristic evaluation function, or are set to zero if none is 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 converge to the exact distances to the goal.

The basic 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 x, where $h(x')$ is the current heuristic estimate of the distance from x' to the 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, since every path to the goal must pass through one of the neighbors.

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. See [13] for the formal statements and proofs of these results. LRTA*, however, requires that the position of the goal state be fixed.

III. Problem Formulation

In Section III 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 nonuniform but rational edge costs, we can convert it to a graph with unit edge costs, without changing the underlying topology of the graph, by choosing the unit to be the lowest common denominator of all the edge costs, and then inserting into any edge with a larger cost enough intermediate nodes of degree two so that each edge has unit cost.[2] Fig. 1 illustrates this procedure.

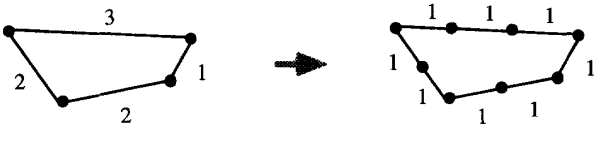


Fig. 1. Conversion of graph to unit edge costs.

There is an initial position of the problem solver and an initial position of the target. We assume that 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

1. Unfortunately, we were unaware of this research when we chose the same name for our own work.

2. This transformation may add an exponential number of edges and may have severe effects on time and space complexities. The unit cost assumption is adopted to simplify the definition of *speed* of the problem solver and the target. Further study is needed to allow nonunit cost edges.

the movements of the target, and no knowledge to allow it to predict, even probabilistically, the motion of the target. The task is accomplished when the problem solver and the target 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 periodically requiring it to skip a turn, making no move, and remaining at its current position. We will slightly relax this assumption in Section VII.

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 VII. We assume that the problem solver does not know the topology of the problem space, other than the locations of the immediate neighbors of its current location. 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 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 that it never overestimates the actual distance between any pair of points.[3] 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, but merely that it be admissible.

Perhaps the simplest solution to the problem as specified is what we refer to as the *tracking strategy*. In the tracking strategy, the problem solver first moves to the initial position of the target, keeping track of all moves made by the target in the meantime. Once the problem solver reaches the initial position of the target, it simply tracks the target by repeating all the moves made by the target since the beginning. Since the problem solver moves faster than the target, it will eventually catch up and solve the problem. While this strategy is not very clever,[4] it does solve the problem, and its performance will be compared to the performance of our more sophisticated algorithms.

IV. Moving-Target Search Algorithm

We now present the *moving-target search* (MTS) algorithm, which is a generalization of LRTA* to the case where the target can move. MTS must acquire heuristic information for each target 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, however, we only store those values that differ from their static values. Thus, even though the complete matrix may be very large, it is typically quite sparse, and sparse matrix techniques may be used for 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 positions of the problem solver and target coincide. 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 the neighbors x' of x, and chooses a neighbor x'_0 with the smallest f value to move to. Ties are broken randomly. If $h(x, y) < f(x'_0, y)$, then $h(x, y)$ is updated to $f(x'_0, y)$. 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 one of its neighbors 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 can be 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:

1) Calculate $h(x', y)$ for each neighbor x' of x.
2) Update the value of $h(x, y)$ as follows:

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

3) Move to the neighbor x' with the minimum $h(x', y)$, i.e., assign the value of x' to x. Ties are broken randomly.

When the target moves:

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

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

3) Reflect the target's new position as the new goal of the problem solver, i.e., assign the value of y' to y.

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.

3. Nonadmissible functions cannot guarantee the completeness of the moving-target search algorithm described in Section IV.

4. The tracking strategy can only be applied when the problem solver always knows the target's position, an assumption that will be relaxed in Section VII.

V. Completeness of Moving-Target Search

Here 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, meaning that they never overestimate actual cost. 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 1. *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 $h(x'_0, y) + 1$, where $h(x'_0, y)$ 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 $h(x'_0, y)$ is a lower bound on the cost from any neighbor x' to y, and hence $h(x'_0, y) + 1$ 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 1. *Given a finite problem space, in which a path exists between every pair of states, and nonnegative 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 point of the algorithm 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 a 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 distance.

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 distance 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 distance 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 at least one unit, and a move by the target can only increase it by at most one unit, in a pair of moves by the 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 distance 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 distance is non negative. Thus, heuristic disparity is non negative. Therefore, either the algorithm terminates successfully, or the heuristic disparity eventually reaches zero, meaning that both total heuristic error and heuristic distance 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. □

Note that the admissibility of heuristic functions is not required for the completeness of LRTA*. The only condition required for LRTA* is that heuristic functions return finite values. For MTS, however, since the target changes its location over the course of the search, the assumption of an admissible heuristic is needed to guarantee its completeness.

VI. Complexity of Moving-Target Search

A. Space Complexity

An upper bound on the space complexity of moving-target search is $O(N^2)$, where N is the number of states in the problem space, since 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 that 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, and multiple moves may update the same value. Thus, the overall space complexity is the smaller of N^2, and the total number of moves of the problem solver and target combined. This latter factor is a dynamic property of the algorithm and problem instance, and cannot be determined a priori. Thus, the amount of memory actually required at runtime depends on the particular application.

B. Time Complexity

The worst-case time complexity of MTS is $O(N^3)$, where N is the number of states in the problem space. This can be obtained from the maximum heuristic disparity. The total heuristic error is less than or equal to N^3, because there are N^2 heuristic values, one for every possible pair of states, and the maximum heuristic distance between any pair of states is N. On the other hand, if no updates of heuristic values occur, then the maximum number of moves of the problem solver to reach the target is N/S, where S is a constant representing the fraction of moves that are skipped by the target. This is because the maximum heuristic distance 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 $O(N^3)$ 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.

C. 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. For complete learning, however, N^2 space is required to store all heuristic values.

VII. Relaxing Some Constraints on MTS

A. 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 heuristic estimate of the distance from the problem solver by at least one unit. 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 to guarantee that the problem solver will eventually catch the target, we need only assume that the target periodically makes apparently suboptimal moves. This is because an apparently suboptimal move by the target is one that does not increase the heuristic distance. 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.

B. Available Information About the Target

We previously assumed that the problem solver always knows the exact 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 temporarily lose sight of the target, and thus may be more practical in real applications.

Suppose the problem solver and the target have each moved at most t times between the target being observed at y and then at z. During this time, updating of heuristic values with moves of the problem solver continues as usual, but with respect to the old position y of the target. At the end of t cycles, the updating of the heuristic values associated with the target move is generalized as follows.

1) Calculate $h(x, z)$ for the target's new position z, where x is the new position of the problem solver.

2) Update the value of $h(x, y)$ as follows:

$$h(x,y) \leftarrow max\begin{Bmatrix} h(x,y) \\ h(x,z)-t \end{Bmatrix}$$

This 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 V. If $h(x, y) \geq h(x, z) - t$, then there is no update and the heuristic distance and hence the heuristic disparity increases by at most t units. Conversely, if $h(x, y) < h(x, z) - t$, 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 distance 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 with respect to the old position y of the target, moves of the problem solver cause the heuristic disparity to decrease by t units in the period between when the target is observed at y and then at z. On the other hand, moves by the target from y to z can only increase the heuristic disparity 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.

VIII. Performance Bottleneck of MTS

A. Experience with a User-Controlled Target

We implemented MTS in a 100 × 100 rectangular grid problem space with randomly positioned obstacles. We allow motion along the horizontal and vertical dimensions, but not along the diagonals. To eliminate boundary effects, we form a torus by connecting the opposite boundaries. Note that the rectangular grid represents only one of many possible problem spaces. Though MTS is applicable to any type of problem space, we use the rectangular grid as an experimental environment simply because it is suitable for plotting on a workstation display, and it helps us to observe the operation of the algorithm. Each state has at most four neighbors, with obstacles restricting the number of neighboring states.

Manhattan distance along the grid is used for the initial heuristic value. The Manhattan distance represents the actual grid distance if there were no obstacles, but it becomes less accurate as the number of obstacles increases. The problem solver and the target are initially positioned as far apart as possible in the torus, i.e., 100 units in Manhattan distance. The speed of the target is set to 80% of the problem solver, meaning that the target skips one in every five moves.

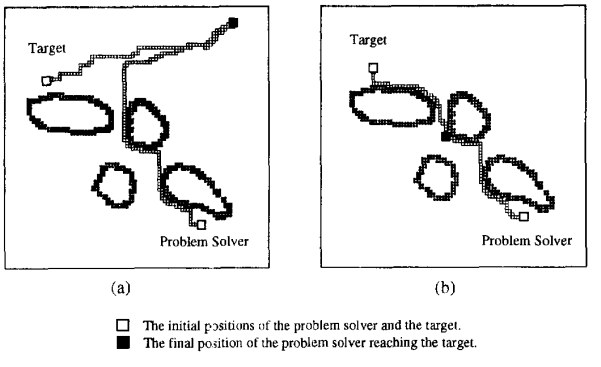


Fig. 2. Sample tracks of MTS.

Interesting target behavior is 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. Fig. 2 shows the experimental setup along with sample tracks of the target (controlled by a human user) and problem solver (controlled by MTS) with manually placed obstacles. The initial positions of the problem solver and the target are represented by white rectangles, while their final positions are denoted by black rectangles. In Fig. 2(a), the user's task is to avoid the problem solver, which is executing MTS, for as long as possible, while in Fig. 2(b), the user's task is to meet the problem solver as quickly as possible. We observed that if one is trying to avoid a faster pursuer as long as possible, the best strategy is not to run away, but to hide behind obstacles.

B. Experiments with an Automatically Controlled Target

We then examined the performance of MTS more systematically. In this evaluation, obstacles are randomly chosen grid positions. For example, an obstacle ratio of 20% means that 2,000 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. The complexity of the maze rapidly increases as the obstacle ratio increases from 25% to 35%. When the ratio reaches 40%, the obstacles tend to disconnect the problem space, separating the target from the problem solver. We eliminate cases in which the problem space is disconnected.

The motion of the target is automatically controlled by one of the following four response strategies.

Avoid: The target actively avoids the problem solver, by performing MTS, but moving toward a position as far from the problem solver as possible.

Meet: The target moves cooperatively to meet the problem solver, by performing MTS to decrease the estimated distance from the problem solver.

Random: The target moves randomly.

Stationary: The target remains stationary. In this case, MTS behaves exactly the same as LRTA*.

Fig. 3 illustrates the search cost represented by the total number of moves of the problem solver for various response strategies of the target. The X-axis represents the obstacle ratio, and the Y-axis represents the number of moves taken by the problem solver to catch the target. Each data point in this graph are obtained by averaging 100 trials.

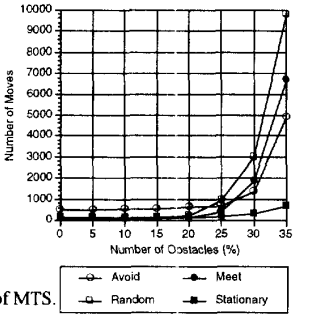


Fig. 3. Performance of MTS.

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. The experimental results show that the performance degrades as the number of obstacles increases, since the accuracy of the initial heuristic values decreases. Though this phenomena is observed in all behavior modes of the target, the performance decreases more rapidly when the target moves than when it remains *Stationary*. When the obstacles become more numerous, 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. At first, this result seems counterintuitive. Recall, however, that in order to avoid a faster pursuer, the best strategy is not to run away, but to hide behind obstacles. Both *Meet* and *Random* approximate this strategy better than *Avoid*. In these situations, the agents tend to reach opposite sides of the same wall, and move back and forth in confusion. A more coordinated approach that involves negotiation may eliminate these counterintuitive results. Such higher level coordination is beyond of the scope of this paper, however. The reason *Avoid* is more effective than remaining *Stationary* is that *Avoid* makes the problem solver spend more time learning new heuristic values.

The average performance of the tracking strategy, where the problem solver simply follows the path of the target after reaching its initial position, is easily calculated from the *Stationary* case. Let M_s and M_t be the numbers of moves of the problem solver in the *Stationary* and tracking cases, respectively. Let R be the fraction of moves skipped by the target. The following observations can be made.

1) In M_t time units, the target will make an average of $M_t(1 - R)$ moves.
2) In order to catch the target, the problem solver will have made M_s moves to reach the initial position of the target, plus $M_t(1 - R)$ moves to reach the target from its initial position, for a total of $M_s + M_t(1 - R)$ moves.

Since the problem solver makes exactly one move per time unit, the time to reach the target is the same as the number of moves by the problem solver, i.e., $M_t = M_s + M_t(1 - R)$. Solving this equation for M_t gives $M_t = M_s/R$.

In the above experiments, M_t is five times larger than M_s, since the target skips 20% of its moves. Thus, with this fraction, the tracking strategy performs better than MTS for moving goals when the obstacle ratio is more than 30%. However, the MTS algorithm will be improved in Sections IX and X, and the resulting MTS performance shown in Section XI is superior to that of the tracking strategy.

C. Heuristic Depression

Let us examine the behavior of MTS in more detail to figure out why MTS becomes inefficient in uncertain situations. Fig. 4 represents the behavior of the problem solver in a one-dimensional problem space. In this case, the target is assumed to be fixed at position q. The X-axis represents the position of the problem solver, while the Y-axis represents the estimated distance between the problem solver and the target. An example set of initial heuristic values are plotted with wide lines. Arrows indicate moves of the problem solver. Incremental updates of heuristic values performed by the problem solver are indicated by dark boxes. As shown in Fig. 4, the problem solver performing MTS repeatedly moves down the slope of heuristic values.

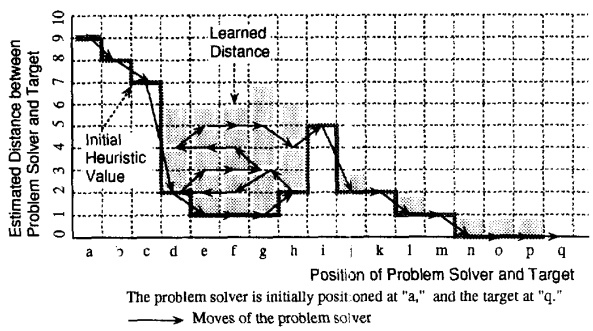


Fig. 4. Behavior of MTS.

To explain the problem solver's behavior, we define a *heuristic depression* with respect to a single goal state. A heuristic depression is a set of states with heuristic values less than or equal to those of a set of immediately and completely surrounding states. We say a heuristic depression is *locally maximal*, when no single neighboring state can be added to the set, so that all the values within the set are less than or equal to the boundary set. For example, in Fig. 4, positions d through h form a locally maximal heuristic depression. Note that no depressions can exist in actual distance. However, as the situation becomes uncertain, heuristic values differ significantly from the actual distances, and so heuristic depressions tend to appear frequently in the problem space.

When in a heuristic depression, the problem solver faces the situation that there is no way to decrease the heuristic distance, and recognizes that its heuristic values are inaccurate. The problem solver cannot reach the target without "filling" the depression by repeatedly updating the heuristic values. Furthermore, in general the target moves during this learning process. Since MTS maintains different heuristic values for each goal location, the problem solver has to start the learning process over again for the target's new position. This is why the performance of MTS rapidly decreases in uncertain situations.

To summarize, *the performance bottleneck of MTS results from its inefficiency in filling heuristic depressions.* Thus, in Sections IX and X, we introduce two ideas from the area of resource-bounded planning, namely *commitment to goals* and *deliberation for selecting plans*. The effect of introducing these ideas will be shown in Section XI using the same examples given in Fig. 3.

IX. Commitment to Goals

In the original MTS, the problem solver always knows the position of the target. However, in Section VII.B, we extended MTS so that the problem solver can ignore some of the target's moves. The extended MTS only requires 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. The idea of *commitment to goals* is to ignore some of the target's moves, while concentrating on filling the current heuristic depression. Surprisingly, this turns out to improve performance in some situations.

In particular, we propose that the problem solver *reflect the target's move in the problem solver's goal only when the heuristic distance decreases; otherwise, when it is in a heuristic depression, it commits to the target's previous position and does not change its goal even if the target moves.* As shown in Fig. 4, however, when filling a large heuristic depression, updating heuristic values creates small depressions, and thus the heuristic distance may temporarily decrease (i.e., arrows are sometimes directed down). In such a situation, even if the heuristic distance decreases, retargeting still causes the learning process to be restarted again. Therefore, we introduce a *degree of commitment (doc)*, which represents the strength of the problem solver's commitment to its goal.

Let D_c be the number of problem solver moves in sequence for which the heuristic distance decreased, ending with the last move made by the problem solver. The problem solver will reflect the target's move in its goal only when $D_c \geq doc$. Obviously, when $doc = 0$ the problem solver is most sensitive to the target's moves, and behaves exactly the same as the original MTS. On the other hand, when $doc = \infty$, the problem solver is least sensitive. When the problem solver reaches the committed goal (i.e., $x = y$), the problem solver always changes its goal to the target's new position. Thus, when $doc = \infty$, the problem solver changes its goal only when it reaches the last committed position of the target.

The MTS algorithm with commitment is as follows. D_c is set to 0 initially.

When the problem solver moves:

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

2) If $h(x, y) > min_{x'}\{h(x', y)\}$, $D_c \leftarrow D_c + 1$.

   If $h(x, y) \leq min_{x'}\{h(x', y)\}$, $D_c \leftarrow 0$.

3) Update the value of $h(x, y)$ as follows:

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

4) Move to the neighbor with the minimum $h(x', y)$, i.e., assign the value of x' to x. Ties are broken randomly.

When the target moves:

When $D_c \geq doc$ or $x = y$, perform the following:

1) Calculate $h(x, y')$ for the target's new position y'.

2) Update the value of $h(x, y)$ as follows:

$$h(x,y) \leftarrow max\left\{\begin{matrix} h(x,y) \\ h(x,y')-t \end{matrix}\right\}$$

   where t is the maximum number of moves the target could have made in going from y to y'

3) Reflect the target's move as the new goal of the problem solver, i.e., assign the value of y' to y.

Although it is not explored in this paper, it might be a good idea to take account of the distance between y and y', in deciding when to reflect the target's move in the problem solver's goal. In other words, if the target has moved far from the problem solver's goal, it might be reasonable to change the goal to the target's new position, even when the problem solver is in a heuristic depression.

X. Deliberation

The idea of *deliberation* is to perform an off-line search (or a *look-ahead search*), in which the problem-solver updates heuristic values without moving, to find a way out of a heuristic depression. When the problem solver moves, MTS always selects a neighboring position that offers the minimum estimated distance to the goal. However, real-time search is not always more efficient than off-line search. With inaccurate heuristic values, such a reactive decision can be inaccurate. From our experience, with an inaccurate heuristic function, off-line search often results in better performance. The reason is that, in this mode, the problem solver does not move back and forth, but extends its wave front step by step even with an inaccurate heuristic function. Thus, deliberative investigation using off-line search, though it decreases the speed of the problem solver, might improve overall performance in uncertain situations. In practice, introducing off-line search enables MTS to efficiently find the boundary of a locally-maximal heuristic depression, and thus overcomes the performance bottleneck. The question is when and to what extend should the problem solver perform off-line search?

Our idea is that *the problem solver performs real-time search while the heuristic distance decreases. Otherwise, when in a heuristic depression, the problem solver performs off-line search.* Fig. 5 illustrates the deliberation process of the above algorithm using the same situation given in Fig. 4. The problem solver starts with real-time search. When in a heuristic depression, the problem solver commits to the target's current position and starts off-line search. The problem solver performs off-line search until it finds a boundary of the depression. When the boundary is found, the problem solver updates the heuristic values of all states in the depression, moves out of the depression, and continues to perform real-time search while the heuristic distance decreases. Note that during deliberation, even though the target may be moving, the problem-solver remains committed to the position of the target when the current phase of deliberation began.

A heuristic depression may contain several small depressions, however, and thus the heuristic distance may temporarily decrease during the off-line search. In such a situation, even if the heuristic distance decreases, it might be reasonable to continue off-line search to find the boundary of the larger outside depression. Therefore, we introduce a *degree of deliberation (dod)*, which represents the problem solver's thoughtfulness in its environment.

Let D_d be the length of a path (a sequence of states expanded during off-line search) along which the heuristic distance continuously decreased. Let the problem solver restart real-time search only when $D_d \geq dod$. Obviously, when $dod = 0$

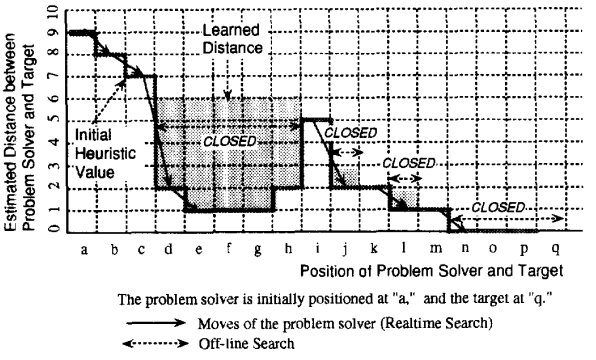


Fig. 5. Behavior of MTS with deliberation.

and $doc = 0$, the problem solver is most reactive, and behaves exactly the same as the original MTS. On the other hand, when $dod = \infty$, the problem solver is most deliberative. Note that when the wave front of off-line search reaches the committed goal, the problem solver always restarts real-time search.

The MTS algorithm with deliberation is as follows. When the target moves, the same algorithm in Section IX is applied.

When it is the problem solver's turn to move:

1) Calculate $h(x', y)$ for each neighbor x' of x, where x is the current position of the problem solver.

2) If $h(x, y) > min_{x'}\{h(x', y)\}$, $D_c \leftarrow D_c + 1$.
If $h(x, y) \leq min_{x'}\{h(x', y)\}$, $D_c \leftarrow 0$.

3) If $h(x, y) \leq min_{x'}\{h(x', y)\}$ and $dod \neq 0$, perform an off-line search as follows:
   a) Perform best-first search with the cost function $f(w) = h(w, y)$, until finding a state w that satisfies one of the following conditions:
      i) w is on the boundary of the current heuristic depression, i.e., $h(w, y) > min_{w'}\{h(w', y)\}$, where w' is an unexpanded neighbor of w. Furthermore, there must exist a path that starts at w, leaves the heuristic depression, and is of length greater than or equal to dod, in which the heuristic distance to y continuously decreases. In other words, $D_d \geq dod$. Note that once the best-first search reaches the boundary node w, it will continue to expand nodes along this path.
      ii) w is the currently committed goal state, i.e., $w = y$.
   b) For each state v expanded during the best-first search, update the value of $h(v, y)$ as follows:

$$h(v,y) \leftarrow max\begin{Bmatrix} h(v,y) \\ h(w,y) \end{Bmatrix}$$

   c) Move from x to w on the path determined during the best-first search, and assign the value of w to x.

4) If $h(x, y) > min_{x'}\{h(x', y)\}$ or $dod = 0$, perform real-time search:
   a) Update the value of $h(x, y)$ as follows:

$$h(x,y) \leftarrow max\begin{Bmatrix} h(x,y) \\ min_{x'}\{h(x',y)+1\} \end{Bmatrix}$$

   b) Move to the neighbor with the minimum $h(x', y)$, i.e., assign the value of x' to x. Ties are broken randomly.

This algorithm does not consider the cost of deliberation, and thus there is no guarantee that the problem solver can complete the above steps in constant time. During off-line search, it is reasonable to assume that the target can get further and further away from the problem solver. To include the cost of off-line search, we thus assume that in each turn, the problem solver can expand a constant number of states (n) in off-line search, or travel one edge in real-time search. This allows the target to move while the problem solver expands states in off-line search. In the following evaluation, we will examine the cases when $n = 1$ and $n = 10$ to investigate how the relative speed of thinking and moving affects its overall performance.

XI. Experimental Results

We evaluated the effectiveness of the improved MTS in the same situation given in Fig. 3. Note that the problem solver performs the improved MTS, while the target performs the original MTS in *Avoid*, and the improved MTS in *Meet*. This is to more clearly show the performance improvement possible with the improved MTS algorithms: In *Avoid*, we compare the case of the improved MTS pursuing the original MTS with the case of the original MTS pursuing the original MTS (as in Fig. 3), and in *Meet*, the cooperative behavior of the improved MTS agents is compared with that of the original MTS agents (as in Fig. 3). The major results are as follows:

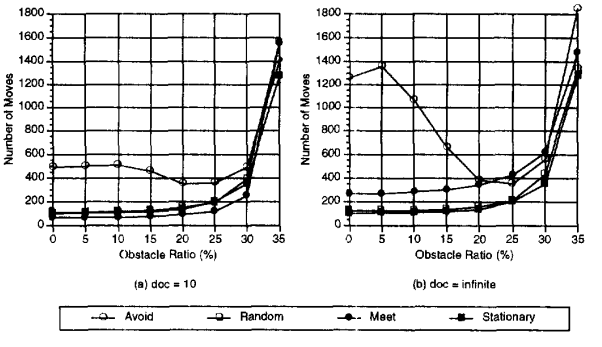


Fig. 6. Performance of MTS with commitment

Commitment to goals: Fig. 6 plots MTS performance under the conditions of $doc = 10$ and $doc = \infty$, the latter case meaning that the problem solver only changes its goal when it reaches the last committed position of the target. Since Fig. 3 can be seen as the result of $doc = 0$, the effects of the degree of commitment to the overall performance can be evaluated by comparing these three figures. $doc = 10$ produced the best performance for the set of values we tried, including 1, 5, 10, 15, 20, 30, and ∞. With $doc = 10$, the performance improvement, in terms of number of moves, is

roughly seven times against *Random*, five times against *Meet*, and four times against *Avoid*. The reason why the largest effect is obtained in *Random* is that the target does not deviate significantly from the initial position because of the random moves, and thus the problem solver can safely ignore the target's moves.

However, increasing the degree of commitment beyond 10 does not always improve performance. When $doc = \infty$, the performance against *Avoid* is poorer when the obstacle ratio is low. In other words, when the heuristic function returns fairly good estimates, the problem solver should be sensitive to the target's moves. In *Meet*, the performance decreases for all obstacle ratios. This decrease is assumed to be due to ignoring the target's cooperative behavior.

Deliberation for selecting plans: Fig. 7 represents the performance of MTS with deliberation and commitment. The degree of deliberation (*dod*) and the degree of commitment (*doc*) are set to 10. Fig. 7(a) represents the case when the problem solver can expand one state in each turn ($n = 1$). Though this is an extremely conservative assumption, compared with Fig. 6(a), the performance is doubled in uncertain situations. These effects are observed in all target behavior modes including *Stationary*. This shows that deliberation is effective not only in moving-target problems, but also in real-time search for fixed goals.

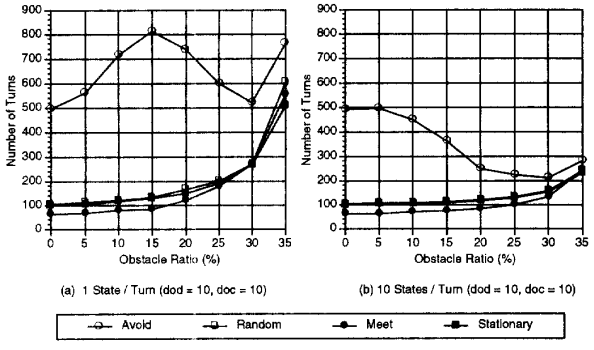


Fig. 7. Performance of MTS with deliberation.

In practice, we would expect that more states could be expanded in the time required to make one move. Fig. 7(b) represents the case when the problem solver can expand 10 states in each turn ($n = 10$). The performance is further improved, especially when the number of obstacles increases. In *Avoid*, when the obstacle ratio is high, it is observed that the problem solver can reach the target much easier than when no obstacles exist. This is because, with the higher obstacle ratios, the accuracy of the heuristic values becomes more important, and the improved heuristic values from the off-line search convey a greater advantage to the problem-solver relative to the target, which is performing the original MTS algorithm.

However, increasing the degree of deliberation does not always improve performance. For example, as *dod* increases, the performance against *Avoid* rapidly degrades as the obstacle ratio increases from 0% to 15%, as shown in Fig. 7(a). This is because, in such relatively clear situations, the target can move effectively while the problem solver performs deliberation.

To summarize, introducing commitment and deliberation dramatically improves the efficiency of MTS. The evaluation results clearly show that

1) MTS performance is improved by 10 to 20 times in uncertain situations depending on the behavior of the target, and
2) controlling the degrees of commitment and deliberation is essential to produce the best performance.

XII. Conclusions and Discussion

We have presented several algorithms for reaching a target that changes position over time. The main property necessary to guarantee success is that the target move 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. Our empirical results show that in pursuing a fleeing target, better performance is obtained by ignoring some moves of the target and only sampling its position periodically. Similarly, performance also improves by keeping the problem solver stationary and performing off-line computation, rather than always moving toward the target.

Both of these results are counterintuitive to some extent. In one case we do better by sometimes ignoring information, namely the exact position of the target, and in the other case, we do better by stopping the active pursuit of the target and just thinking. How can we explain these results? One way is to separate the motion of the problem solver and target from the updating of the heuristic values. There are two types of updates that occur. In one case, usually associated with moves of the problem solver, we have available all the neighbors of a given state, and can update the value of that state to the minimum f value of its neighbors. In the other case, usually associated with moves of the target, we simply know that two states are neighbors, and update their values to ensure that they do not differ by more than one unit. While in the algorithms we have presented there is a close coupling between the heuristic values that are updated and the actual locations of the problem solver and target, this need not be the case. In fact we are free to update any heuristic values for which we have sufficient information, including previous positions of the problem solver and target.

One common feature of both commitment and deliberation, which may explain their success, is that when performing updates of heuristic values $h(x, y)$, both techniques hold constant the position of the target. In addition, deliberation holds constant the position of the problem solver. If we view the heuristic function as a two-dimensional matrix, then these techniques correspond to spreading the information in the heuristic matrix in only one dimension at a time.[5] Thus, one can view the effectiveness of both commitment and deliberation as evidence for the hypothesis that information in a heuristic function should be spread

5. Note that in our experiments, the heuristic function actually exists in four dimensions, since both the position of the problem solver and the position of the target can range over a two-dimensional space.

in some restricted manner. We do not claim that the algorithms we have presented are optimal for solving this problem. In further work, one could develop more clever strategies for determining moves of the problem solver, and perhaps more importantly, better strategies for updating the information in the heuristic function. It is important, however, to charge for the updating computations. In this work, we have made the conservative assumption that only a small number of updates are allowed for each move cycle of the problem solver or target. In general, one would expect to be able to perform more computations per move, but the number will still be limited.

More generally, this work can be viewed as a case study of problem solving in a dynamic environment. In this case, the change in the environment stems from the motion of the goal. Uncertainty could also be added 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.

Throughout this work, we have tried to bridge the studies on conceptual modeling and algorithms concerned with resource-bounded planning. In fact, the performance of MTS has been improved by introducing ideas from the area of resource-bounded planning. Since only a few steps are added to the original MTS, the improved MTS has not lost its simplicity. However, the behaviors of the two algorithms, as observed on a workstation display, are substantially different. The improved MTS behaves like a predator: In some situations, the problem solver is always sensitive to the target's moves and reactively moves toward the target current position, while in other situations, the problem solver ignores the target's moves, commits to its current goal, and deliberates to find a promising direction to reach that goal. The results suggest that combining innovative notions and computationally sound algorithms will provide robust and efficient methodologies for problem solving in dynamically changing environments.

ACKNOWLEDGMENTS

This research benefited from discussions with Kazuhiro Kuwabara, Jun-ichi Akahani, and Makoto Yokoo. Part of this research was performed while Richard Korf was a visitor at, and supported by, NTT Laboratories. Additional support to Dr. Korf was provided by a U.S. National Science Foundation Presidential Young Investigator Award, NSF grant IRI-9119825, and a grant from Rockwell International.

REFERENCES

[1] M.E. Bratman, D.J. Israel, and M. Pollack, "Plans and resource bounded practical reasoning," Computational Intelligence, vol. 4, no. 4, pp. 349-355, 1988.

[2] S.S. Brown, "Optimal search for a moving target in discrete time and space," Operations Research, vol. 28, no. 6, pp. 1,275-1,289, 1980.

[3] P.R. Cohen and H.J. Levesque, "Intention is choice with commitment," Artificial Intelligence, vol. 42, no. 3, 1990.

[4] J.M. Dobbie, "A two-cel model of search for a moving target," Operations Research, vol. 22, pp. 79-92, 1974.

[5] T. Dean and M. Boddy, "An analysis of time-dependent planning," AAAI 88, pp. 49-54, 1988.

[6] E.H. Durfee and V.R. Lesser, "Predictability versus responsiveness: Coordinating problem solvers in dynamic domains," AAAI 88, pp. 66-71, 1988.

[7] J.N. Eagle and J.R. Yee, "An optimal branch-and-bound procedure for the constrained path, moving target search problem," Operations Research, vol. 38, no. 1, pp. 110-114, 1990.

[8] M.P. Georgeff and A.L. Lansky, "Reactive reasoning and planning," AAAI 87, pp. 677-682, 1987.

[9] T. Ishida and R.E. Korf, "Moving target search," IJCAI 91, pp. 204-210, 1991.

[10] T. Ishida, "Moving target search with intelligence," AAAI 92, pp. 525-532, 1992.

[11] P.E. Hart, N.J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Trans. Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, 1968.

[12] D.N. Kinny and M.P. Georgeff, "Commitment and effectiveness of situated agents," IJCAI 91, pp. 82-88, 1991.

[13] R.E. Korf, "Real-time heuristic search," Artificial Intelligence, vol. 42, nos. 2-3, pp. 189-211, Mar. 1990.

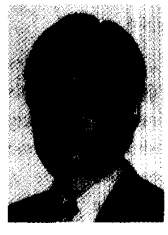
[14] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, Mass., 1984.

[15] M.E. Pollack and M. Ringuette, "Introducing the Tileworld: Experimentally evaluating agent architectures," AAAI 90, pp. 183-189, 1990.

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

[17] L. Tierney and J.B. Kadane, "Surveillance search for a moving target," Operations Research, vol. 31, pp. 720-783, 1983.

[18] A.R. Washburn, "Search for a moving target: The FAB algorithm," Operations Research, vol. 31, pp. 739-751, 1983.



Toru Ishida received his BE, MEng, and DEng degrees from Kyoto University in 1976, 1978, and 1989, respectively. From 1978 to 1993, he was a research scientist with NTT Laboratories, and from 1983 to 1984, he was a visiting research scientist in the Department of Computer Science at Columbia University. He has been working in the area of problem solving, search and distributed artificial intelligence since 1985, and is now a professor in the Department of Information Science at Kyoto University, Kyoto, Japan.

Dr. Ishida is the author of *Parallel, Distributed and Multiagent Production Systems* (Springer-Verlag, 1994). He received the JSAI Best Paper Award in 1993.

Richard E. Korf received his BS from the Massachusetts Institute of Technology in 1977 and his MS and PhD degrees from Carnegie Mellon University in 1980 and 1983, respectively, all in computer science. From 1983 to 1985, he served as Herbert M. Singer Assistant Professor of Computer Science at Columbia University. He is now an associate professor of computer science at the University of California, Los Angeles. His research interests are in the areas of problem solving, heuristic search, and planning in artificial intelligence.

Dr. Korf is the author of *Learning to Solve Problems by Searching for Macro-Operators* (Pitman, 1985). He is an associate editor of the *Journal of Artificial Intelligence Research*, and serves on the editorial board of the *Journal of Applied Intelligence*. Dr. Korf received the 1985 IBM Faculty Development Award, the 1986 National Science Foundation Presidential Young Investigator Award, and the 1989 UCLA Computer Science Department Distinguished Teaching Award. He was elected a fellow of the American Association for Artificial Intelligence in 1994.