Technical Correspondence

Real-Time Moving Target Evaluation Search

Cagatay Undeger and Faruk Polat

***Abstract*—In this correspondence, we address the problem of real-time moving target search in dynamic and partially observable environments, and propose an algorithm called real-time moving target evaluation search (MTES). MTES is able to detect the closed directions around the agent and determines the estimated best direction to capture a moving target avoiding the obstacles nearby. We have also developed a new prey algorithm (Prey-A*) to test the existing and our predator algorithms in our experiments. We have obtained an impressive improvement over moving target search, real-time target evaluation search, and real-time edge follow with respect to path length. Furthermore, we have also tested our algorithm against A*.**

***Index Terms*—Moving target search, real-time path planning.**

I. Introduction

Pursuing a moving target is a challenging task in areas such as robotics, computer games, military simulations, etc. Offline and incremental path planning algorithms are not able to handle moving targets in real-time, and most of the on-line search algorithms are specifically designed for partially observable environments with static targets. The most well-known real-time algorithm for moving targets is moving target search (MTS) [1], which maintains a heuristic table that contains estimated costs of paths between every pair of coordinates. Convergence of the estimated costs takes considerable time making MTS a poor algorithm to be used in practice. Upon seeing such inefficiency, the authors developed two extensions, which are *Commitment to Goal* (MTS-c) and *Deliberation* (MTS-d).

In this correspondence, we propose a new moving target search algorithm, real-time moving target evaluation search (MTES), which is built on real-time edge follow (RTEF) [2] and real-time target evaluation search (RTTES) [3] developed for partially observable environments with static targets. MTES is able to estimate the distance to the target considering the intervening obstacles and discards some nonpromising alternative moving directions in real-time. First, it propagates virtual rays (in the mind of the agent) away from the agent location in four directions, and determines the obstacles that the rays hit. For each such obstacle, we extract its border, estimate the alternative paths to the target, which go around the obstacle, and determine the best direction considering the estimated paths. Then, by using these directions and a resolution mechanism that will be described later, a single moving direction is determined.

To show the performance of our algorithm, we compared MTES with RTTES, RTEF, MTS-c, MTS-d, and A*. We extended RTTES and RTEF to handle moving targets using the method used in MTES. For the experiments, we randomly generated grids of different types, and developed a successful prey algorithm (Prey-A*) in order to challenge the algorithms used in the experiments. The results showed that MTES produces near-optimal solutions, and outperforms RTTES, RTEF, MTS-c, and MTS-d significantly with the help of its new path estimation methodology.

Section II provides the related work on path planning. In Sections III and IV, MTES and Prey-A* algorithms are described in detail, respectively. Section V presents the performance analysis, and Section VI is the conclusion.

II. Related Work

Path planning can be described as finding a path from an initial point to a target point if there exists one. Path planning algorithms are either offline or on-line. Offline algorithms such as A* [4] find the whole solution in advance before starting execution, and they are hard to use for large dynamic environments because of their time requirements. One solution is to make offline algorithms to be incremental, which is a continual planning technique that makes use of information from previous searches to find solutions to the problems potentially faster than are possible by solving the problems from scratch. Focused D* [5], D* Lite [6] and MT-adaptive A* [7] are some of the well-known incremental heuristic search algorithms. Due to the efficiency problems of offline techniques, a number of on-line approaches such as tangent-bug [8], learning real-time A* (LRTA*) [9], real-time adaptive A* (RTAA*) [10], free flight 3-D [11], random trees [12], and probabilistic road maps [13] are also presented. However, most of such on-line search algorithms cannot be used against a moving target since they are usually developed for fixed goals. MTS [1] is one of the algorithms capable of pursuing a moving target. The algorithm maintains a table of heuristic values, representing the function $h(x, y)$ for all pairs of locations x and y in the environment, where x is the location of the agent and y is the location of the target. The original MTS is a poor algorithm in practice because when the target moves (i.e., y changes), the learning process has to start all over again causing a performance bottleneck. Therefore, two MTS extensions called MTS-c and MTS-d are proposed to improve the solution quality [1]. In order to use the learned table values more effectively, MTS-c ignores some of the target's moves while in a heuristic depression, and MTS-d performs an offline search (deliberation) to update the heuristic values if the agent enters a heuristic depression.

Very recently, two real-time search algorithms, RTEF [2] and RTTES [3] are proposed for partially observable environments, on which our algorithm is built. Although these algorithms are developed for static targets, they have the potential to handle moving targets with little modification, which is presented in this correspondence.

When we look at the prey algorithms, we usually see hybrid techniques, which mix a number of reactive strategies such as moving randomly in any possible direction that is not blocked by a predator [1], [14], [15]; escaping from the predators along a straight line or a circle [14], [16]; moving in the opposite direction of the predator if the prey sees only one predator, and otherwise moving in the direction that bisects the largest angle of its field of view in which there are no predators [14]; moving toward a direction that maximizes the distance from the predator's location [1], [15]; moving toward a direction that maximizes the mobility by preferring a move that leads to more move choices, and moving a position that is not in line of sight of the predators [15]. Since these reactive algorithms are not good enough to challenge our predator algorithm, we developed an offline strategy that is slow but more powerful.

Manuscript received February 7, 2008; revised July 18, 2008 and November 10, 2008. Current version published April 15, 2009. This paper was recommended by Associate Editor L. Zhang.

The authors are with the Department of Computer Engineering, Middle East Technical University, Ankara 06531, Turkey (e-mail: cundeger@ceng.metu.edu.tr; polat@ceng.metu.edu.tr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.org.

Digital Object Identifier 10.1109/TSMCC.2009.2009545

1094-6977/$25.00 © 2009 IEEE

III. MTES ALGORITHM

In this section, we describe our moving target search algorithm namely *MTES*. The environment is a partially observable planar grid world, where any grid cell can either be free or obstacle. There is a single predator agent that aims to reach a static or moving prey through a short path avoiding obstacles in real-time. Prey and predator can move in the direction of north, south, east, or west in each step. Initially, they are randomly located far from each other in free grid cells. All agents use Euclidian distance metric for heuristic estimations. We assume that the predator knows the location of the prey all the time, but perceives the obstacles around him within a square region centered at the agent location. The size of the square is $(2v+1)\text{x}(2v+1)$, where v is the *vision range*. Unknown parts of the grid world are assumed to be free of obstacles, until they are explored. By an obstacle, we refer to known part of that obstacle. We used the term *infinite vision* to emphasize the setting where the agent has unlimited sensing capability and knows the entire grid world before the search starts. The prey has unlimited perception and knows all the grid world and the location of predator all the time. The predator and prey move alternately at each step, and the first step is always taken by the prey. To make the prey slower than the predator, the prey skips one move after each m moves, where m is chosen as 7 in our experiments. The search continues until the predator reaches the prey.

MTES makes use of an improved version of the heuristic in [3], RTTE evaluatio (RTTE-h), which analyzes obstacles and proposes a moving direction that avoids these obstacles and leads to the target by shorter paths. To do this, RTTE-h geometrically analyzes the obstacles nearby, tries to estimate the lengths of paths around the obstacles to reach the target, and proposes a moving direction. RTTE-h works in continuous space to determine the moving direction, which is then mapped to one of the actual moving directions (north, south, east, and west).

Algorithm 1 An Iteration of MTES Algorithm

```
Call RTTE-h to compute the proposed direction and the utilities of
  neighbor cells.
if a direction is proposed by RTTE-h then
    Select the neighbor cell with the highest utility from the set of non-
      obstacle neighbors with the smallest visit count.
    Move to the selected direction.
    Increment the visit count of previous cell by one.
    Insert the previous cell into the history.
else
    if History is not empty then
        Clear all the History.
        Jump to 1
    else
        Destination is unreachable, stop the search with failure.
    end if
end if
```

MTES repeats the steps in Algorithm 1 until reaching the target or detecting that the target is inaccessible. In the first step, MTES calls RTTE-h heuristic function, which returns a moving direction and the utilities of neighbor cells according to that proposed direction. Next, MTES selects one of the neighbor cells on open directions with the minimum *visit count*, which stores the number of visits to the cell. If there is more than one cell, which has the minimum visit count, the one with the maximum utility (see [3] for details) is selected. If utilities are also the same, then one of them is selected randomly. After the move is performed, the visit count of the previous cell is incremented by one and the cell is inserted into the *history*. The set of previously visited cells forms the history of the agent. History cells are treated as obstacles. Therefore, if the agent discovers a new obstacle during the exploration and realizes that the target has become inaccessible due to history cells, the agent clears the history to be able to backtrack.

Algorithm 2 RTTE-h Heuristic

```
1: Mark all the moving directions as open.
2: Propagate four diagonal rays.
3: for each ray hitting an obstacle do
4:     Extract the border of the obstacle by starting from the hit-point and
       tracing the edges towards the left side until making a complete tour
       around the obstacle.
5:     Detect closed directions.
6:     Analyze the border to extract geometric features of the obstacle.
7:     Evaluate the results and determine the best direction to avoid the
       obstacle.
8: end for
9: Merge individual results, propose a direction to move, and compute the
   utilities of the neighbor cells.
```

In moving target search problem, the prey may sometimes pass through the cells which the predator previously walked through. In such a case, there is a risk that the history blocks the agent to reach the target since history cells are assumed to be obstacles and may close some gateways that are required to return back. If such a case occurs at some point, the agent will surely be able to detect this at the end, and clear the history, opening all the closed gateways. Therefore, the algorithm is capable of searching moving targets without any additions. As a matter of fact, the only drawback of the history is not the possibility that it can block the way to the target entirely, but it can sometimes prevent the agent to reach the target through shorter paths just by closing some of the shortcuts. To reduce the performance problems of this side effect encountered in RTTES and RTEF, the following procedure is applied in MTES. Assuming that (x_1, y_1) and (x_2, y_2) are the previous and newly observed locations of the target, respectively, and that R is the set of cells the target could have visited in going from (x_1, y_1) to (x_2, y_2), the algorithm clears the history along with visit counts when any cell in set R appears in history or has a nonzero visit count. In the algorithm, R can be determined in several ways depending on the required accuracy. The smallest set has to contain at least the newly observed location of the target, (x_2, y_2). One can choose to ignore some of the set members and only use (x_2, y_2) to keep the algorithm simple, or one may compute a more accurate set, which has the cells fall into ellipse whose foci are (x_1, y_1) and (x_2, y_2), and the sum of radii from the foci to a point on the ellipse is constant m, where m is the maximum number of moves the target could have made in going from (x_1, y_1) to (x_2, y_2).

RTTE heuristic method (RTTE-h) given in Algorithm 2 propagates four diagonal virtual rays (propagated in the mind of agent) away from the agent location (line 2 in Algorithm 2) to split north, south, east, and west directions as shown in Fig. 1. The rays move outward from the agent until they hit an obstacle or maximum ray distance is achieved. Four rays split the area around the agent into four regions. A region is said to be closed if the target is inaccessible from any cell in that region. If all the regions are closed, the target is unreachable from the current location. To detect closed regions, the boundary of the obstacle is extracted (line 4) and analyzed (line 5). Next, the obstacle border is retraced from both left and right sides to determine geometric features of the obstacle (line 6). These features are evaluated and a moving direction to avoid the obstacle is identified (line 7). After all the obstacles have been evaluated, the results are merged in order to propose a final moving direction (line 9). Details of these steps are given in the following subsections.

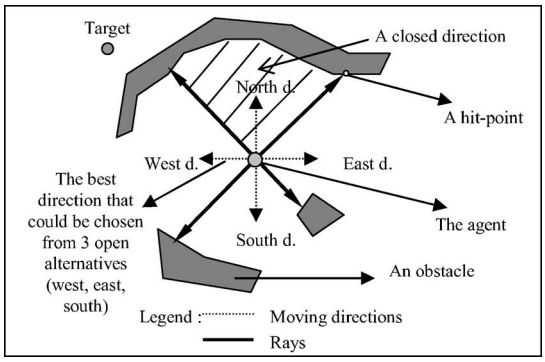

Fig. 1. Sending rays to split moving directions [2].

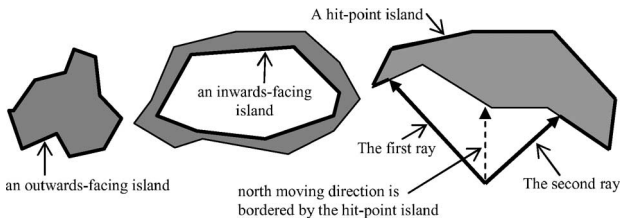

Fig. 2. (Left) Outward facing island, (middle) inward facing island, (right) hit-point island [2].

A. Detecting Closed Directions

The extracted border of an obstacle is represented as a polygonal area called *island*, which is stored as a list of vertices. As illustrated in Fig. 2, there are two kinds of islands: *outward-facing* and *inward-facing islands*. The target is unreachable from agent location if it is inside an outward-facing island or outside an inward-facing island. It is possible that more than one ray hit the same obstacle. As shown in Fig. 2, an augmented polygonal area called *hit-point-island* is formed when we reach the hit-point of another ray on the same obstacle while tracing the edges. A hit-point-island borders one or more agent moving directions. If the target point is not inside the hit-point-island, all the directions that are bordered by the hit-point-island are closed; otherwise all the directions not bordered by the hit-point-island are closed. The details and proof of completeness of the method (line 5 in Algorithm 2) can be found in [2].

B. Analyzing an Obstacle Border

After detecting the closed directions, the extracted border of the obstacle is analyzed furthermore in order to determine the geometric features of the obstacle. Border analysis (line 6 in Algorithm 2) is done by tracing the border of an obstacle from left and right. In left/right analysis, the known border of the obstacle is traced, starting from the hit point, edge by edge toward the left/right until a complete tour around the obstacle border is made. During the process, several geometric features of the obstacle, some of which are illustrated in Fig. 3, are extracted (see [3] for details).

C. Evaluating Individual Obstacle Features

In individual obstacle evaluation step (line 7 in Algorithm 2), if an obstacle blocks the line of sight from the agent to the target, we determine a direction to move that avoids the obstacle and reaches the target by a shorter path (see [3] for details and proof of correctness). This algorithm requires several path length estimations given in

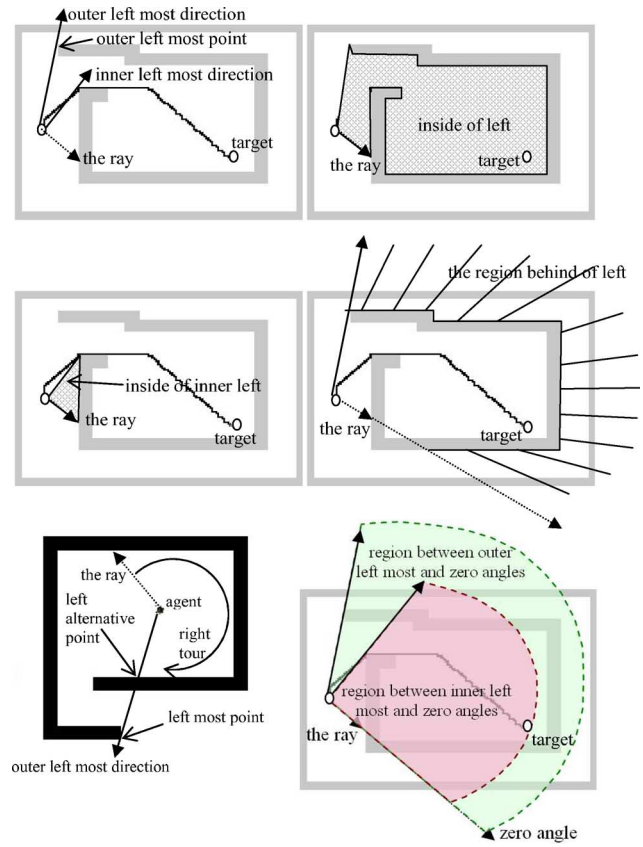

Fig. 3. Geometric features of an obstacle [3].

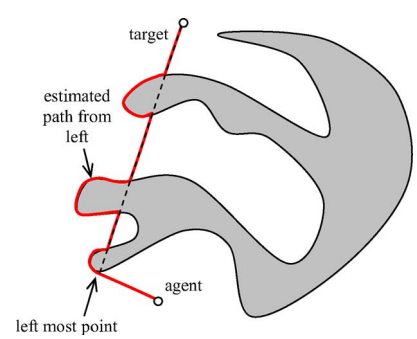

Fig. 4. Exemplified d_{left} estimation.

Definitions III.1–III.3 in addition to the acquired geometric features of the obstacle. The path length estimation algorithm used in MTES is different from that in RTTES, which makes MTES perform significantly better than RTTES.

Definition 3.1 (d_{left}): The approximated length of the path that starts from the agent location, jumps to the *outer leftmost point*, and then follows the path determined by Algorithm 3 (see Fig. 4).

Definition 3.2 ($d_{\text{left.alter}}$): The approximated length of the path that starts from the agent location, jumps to the *outer rightmost point*, and then to the *outer leftmost point*, and finally follows the path determined by Algorithm 3 (see Fig. 5).

Definition 3.3 ($d_{\text{left.inner}}$): The approximated length of the path passing through the agent location, the *inner leftmost point*, and the target (see Fig. 6).

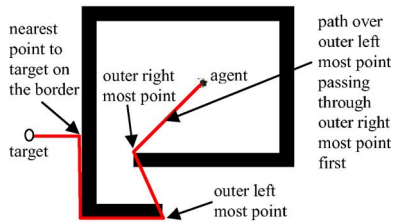

Fig. 5. Exemplified $d_{\text{left.alter}}$ estimation [3].

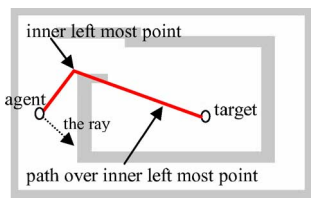

Fig. 6. Exemplified $d_{\text{left.inner}}$ estimation [3].

Algorithm 3 Path length estimation used for d_{left} and $d_{\text{left.alter}}$

Require: t : target point
Require: s : outer leftmost point
Require: n : the nearest point of border to the target
Require: $+$: next border point (left of)
Require: $-$: previous border point (right of)
Require: $insert(p)$: inserts a point to the estimated path
Require: $clasify(p_1, p_2, p_3)$: if the edge formed by the points p_1, p_2, p_3 does a left turn then returns true, else returns false
Require: $isoutwardsfacing(side, p_1, p_2)$: see Algorithm 4

```
let prev = s
let prevleft = true
insert(s)
for each border point v between s+ and n do
  if v = n then
    if isoutwardsfacing(prevleft, prev, t) then
      insert(all border points between prev+ and v)
    end if
    insert(t)
    return length of estimated path
  end if
  let vleft = not clasify(s, t, v)
  if prevleft ≠ vleft then
    let z = intersection point of lines (s, t) and (v−, v)
    if not isoutwardsfacing(prevleft, prev, z) and z is between prev and t then
      insert(t)
      return length of estimated path
    end if
    if isoutwardsfacing(prevleft, prev, z) then
      insert(all border points between prev+ and v)
    else
      insert(v)
    end if
    let prev = v
    let prevleft = vleft
  end if
end for
```

Algorithm 3 is internally used in computations of d_{left} and $d_{\text{left.alter}}$, and the subfunction *isoutwardsfacing* is called for detecting if a border segment, whose both ends touch the line passing through the *outer leftmost point* and the target point, is outward facing (see Fig. 7). The estimated target distances over the right side of the obstacle are similar to those over the left side, and computed symmetrically.

Algorithm 4 Function $isoutwardsfacing(side, p_1, p_2)$

Require: t : target point
Require: s : outer leftmost point
Require: $len(n_1, n_2)$: returns distance between points n_1 and n_2
Require: $positive(m)$: if $len(m,t) \leq len(s,t)$ or $len(m,t) \leq len(s,m)$ then returns true, else returns false
Require: $slen(m)$: if $positive(m)$ then returns $+len(s,m)$ else returns $-len(s,m)$

```
if (side and slen(p1) < slen(p2)) or (not side and slen(p1) > slen(p2)) then
  return true
else
  return false
end if
```

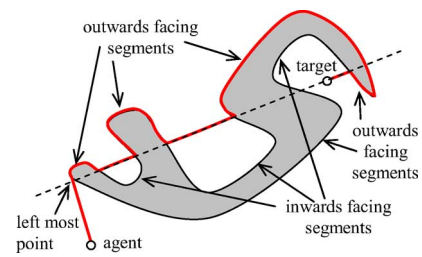

Fig. 7. Outward and inward facing segments.

D. Merging Entire Results

In the result merging step (line 9 in Algorithm 2), the evaluation results (moving direction and estimated distance pairs) for all obstacles are used to determine a final moving direction to reach the target. The proposed direction will be passed to the MTES algorithm (see Algorithm 1) for final decision. The most critical step of the merging phase is to compute the moving direction to get around *the most constraining obstacle* (the obstacle that is marked as blocking the target and which maximizes the distance to the target). The reason why we determine the moving direction based on the most constraining obstacle is the fact that it might be blocking the target the most. We aim to get around the most constraining obstacle and to do this we have to reach its border. In case there are some other obstacles on the way to the most constraining obstacle, we need to avoid them and determine an appropriate moving direction by sweeping the angle of direction (see [3] for details).

E. Analysis of the Algorithm

In each move, MTES performs steps similar to RTTES, but consumes a little more time in path length estimation, which does not change the theoretical complexity of the algorithm. The moving target extension does not also change the complexity since the algorithm only checks a few number of cells in each iteration, and rarely clears the visit counts and history. As a result, the worst case complexity of MTES is the same as that of RTTES, which is $O(w.h)$ per step, where w is the width and h is the height of the grid representing the length of the longest obstacle border that is possible in a grid world. Since increasing grid size decreases efficiency, a *search depth* (d) can be used similar to RTTES in order to limit the worst case complexity. A search depth is a rectangular area of size$(2d+1) \times (2d+1)$ centered at the agent location, which makes the algorithm treat the cells beyond the rectangle as nonobstacle. With this limitation, the complexity becomes $O(d^2)$ (see [2] for details). Proof of completeness [2] does not also change since it is based on the methodology used for detecting close directions. In this correspondence, we have improved the path length

Algorithm 5 An Iteration of Prey-A* Algorithm

1: Generate $costs_{predator}$, which is a grid, each cell of which maps to a cell of the environment and contains the number of moves required for the nearest predator to reach the cell.
2: Generate $costs_{prey}$, which is a grid, each cell of which maps to a cell of the environment and contains the number of moves required for prey to reach the cell.
3: Check all the nearby cells of the prey within a limited search window centered at the prey location, and find out the cell ($destination$), which maximizes the number of moves for the predators to reach (the cell maximizing the cost in $costs_{predator}$) meanwhile ensuring that the prey will not be caught by any predators during the travel to the $destination$ on the shortest path generated by A*. The safety of the shortest path is guaranteed by checking all the cells on the shortest path. If $time_{caught}$ for all the cells on the way is greater than $zero$, we are sure the predators will not catch the prey during its travel to the ($destination$). $time_{caught}$ for a cell is computed as:
$costs_{predator}[cell] - \alpha.costs_{prey}[cell]$
where α is the speed ratio of the predator and the prey found by the formula $speed_{predator}/speed_{prey}$.
4: The move to the first cell that is on the way of the shortest path to the $destination$ is taken as the next step. Note that prey may choose not to move if $destination$ is the cell the prey is on.

estimation, which only affects the performance of the algorithm, but not the completeness.

IV. Prey Algorithm

To challenge our predator algorithm, we developed a deliberative offline prey algorithm, Prey-A* (see Algorithm 5), which is powerful but not very efficient. To prevent the side effects caused by the efficiency difference, the predator and the prey algorithms are executed alternately in performance tests. The algorithm generates two grids, $\mathrm{costs_{predator}}$ and $\mathrm{costs_{prey}}$, whose sizes are the same as the size of the environment, and have one-to-one mapping to the cells of the grid world. Each cell of the $\mathrm{costs_{predator}}$ contains the length of the optimal path from the nearest predator to the cell, and similarly, each cell of the $\mathrm{costs_{prey}}$ stores the length of the optimal path from the prey to the cell. The objective is to find a cell such that the number of moves from the nearest predator to the cell (the cost in $\mathrm{costs_{predator}}$) is maximized, and the prey will not be caught by the predators during the travel to the cell by the optimal path. This is checked by ensuring that each cell on the optimal path satisfies $\mathrm{costs_{predator}[cell]} - \alpha.\mathrm{costs_{prey}[cell]} > 0$, where α is computed by the formula $\mathrm{speed_{predator}}/\mathrm{speed_{prey}}$. In order to find the best cell, the algorithm examines each nonobstacle cell within a limited search window centered at the prey location, and moves one step toward the selected one.

V. Experimental Results

In this section, we present our experimental results on MTS-c, MTS-d, RTEF, RTTES, MTES, and A* (predators) against a moving target (prey). As being an offline algorithm, we executed A* in each step from scratch. Euclidian distance heuristic function is employed for all the compared algorithms. For the test runs, we used nine randomly generated sample grids of size 150×150. Six of them were the *maze* grids, and three of them were the *U-type* grids (see [3] for definitions). For each grid, we produced 15 different randomly generated predator–prey location pairs, and made all the algorithms use the same pairs for fairness. For predators, we used 10, 20, 40, and infinite *vision ranges* and *search depths*. To test the algorithms against a moving target, we used the deliberative offline prey algorithm, Prey-A* with 161×161 sized search window.

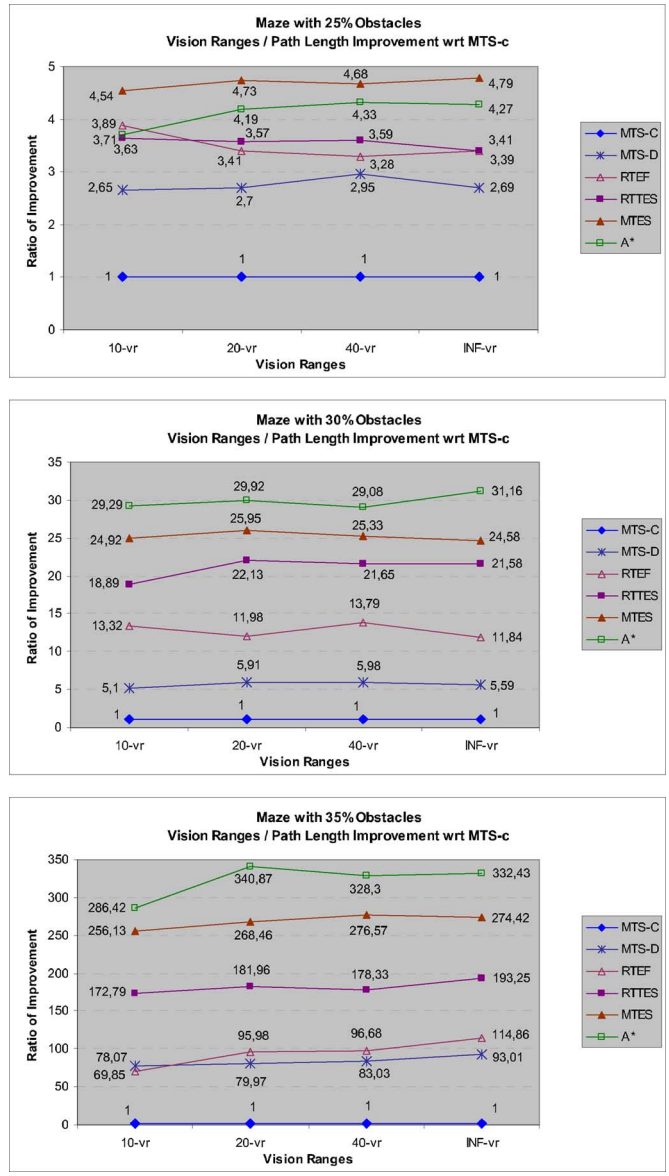

Fig. 8. Average of path length results of maze grids [(top) 25% obstacles, (middle) 30% obstacles, (bottom) 35% obstacles] for increasing vision ranges against a moving target.

First, we present the path length performance of the predator algorithms. With respect to vision ranges and search depths, the averages of path lengths on maze grids are given in Figs. 8 and 9, and the averages of path lengths on U-type grids are given in Figs. 10 and 11, respectively. In the charts, the vertical axis contains the ratio of improvement with respect to MTS-c (the path length of MTS-c divided by that of the compared algorithm). The results showed that MTES performs significantly better than RTTES, RTEF, MTS-d, and MTS-c, and usually offers near-optimal solutions that are almost as good as those produced by A*, and even better in some cases. Next, RTTES, RTEF, MTS-d, and MTS-c follow MTES, respectively. In U-type grids, MTES mostly outperforms A*. When we have examined this interesting result, we have observed that they behave differently in sparse parts of the grid. MTES prefers performing diagonally shaped manoeuvers for approaching targets located in diagonal directions, whereas A* prefers performing L-shaped manoeuvres in such cases. Since agents are only permitted

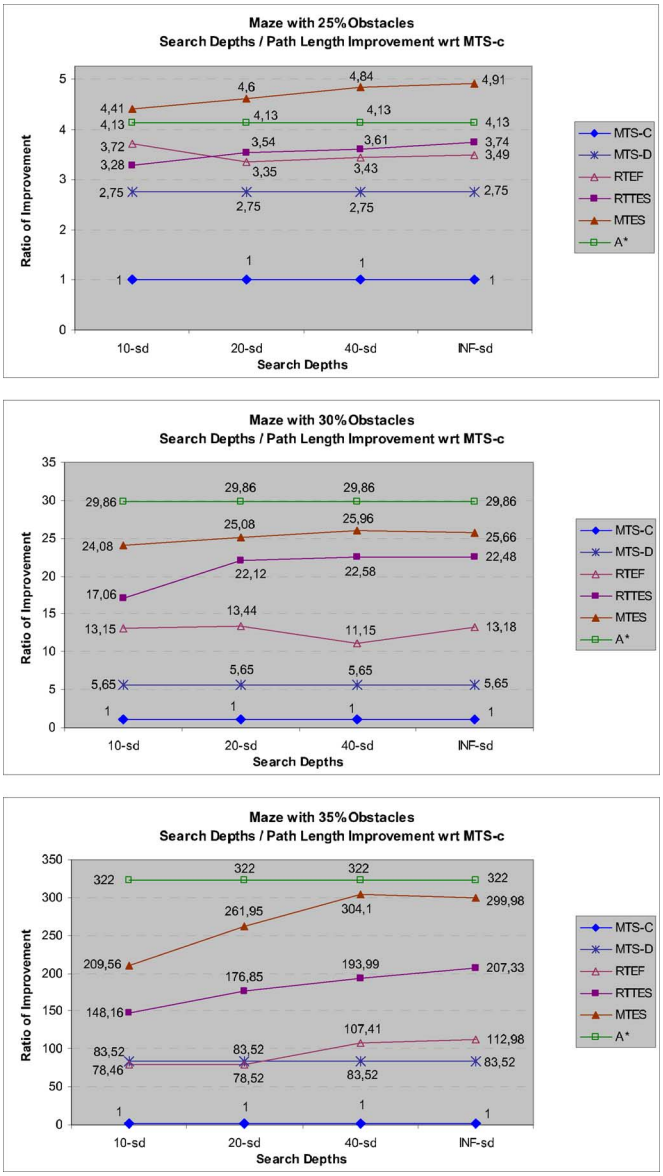

Fig. 9. Average of path length results of maze grids [(top) 25% obstacles, (middle) 30% obstacles, (bottom) 35% obstacles] for increasing search depths against a moving target.

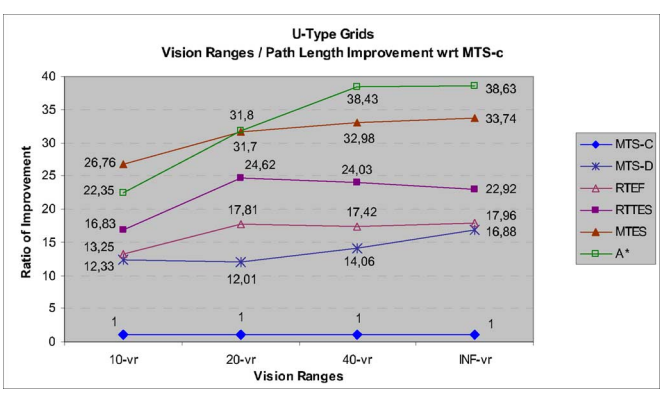

Fig. 10. Average of path length results of U-type grids for increasing vision ranges against a moving target.

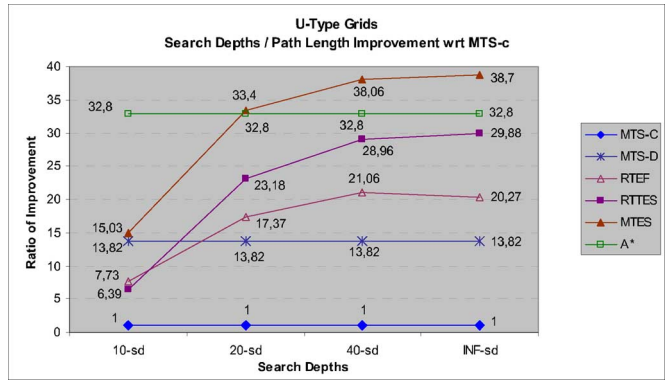

Fig. 11. Average of path length results of U-type grids for increasing search depths against a moving target.

TABLE I
MINIMUM AND AVERAGE NUMBER OF MOVES PER SECOND FOR INCREASING SEARCH DEPTHS

maze grids with 25% obstacles

S.Depth	10 Depth		20 Depth		40 Depth		INF Depth	
Algorithm	Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.
MTS-c	1612	2456	1612	2456	1612	2456	1612	2456
MTS-d	1562	2324	1562	2324	1562	2324	1562	2324
RTEF	1096	1834	1063	1550	537	1282	110	1040
RTTES	1063	1301	641	1001	267	727	84	582
MTES	1063	1324	641	984	292	709	84	585
A*	17	203	17	203	17	203	17	203

maze grids with 30% obstacles

S.Depth	10 Depth		20 Depth		40 Depth		INF Depth	
Algorithm	Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.
MTS-c	1562	2698	1562	2698	1562	2698	1562	2698
MTS-d	1483	2445	1483	2445	1483	2445	1483	2445
RTEF	1483	1687	641	1212	188	679	71	377
RTTES	632	1070	354	671	138	361	42	214
MTES	793	1073	400	649	152	358	50	211
A*	13	198	13	198	13	198	13	198

maze grids with 35% obstacles

S.Depth	10 Depth		20 Depth		40 Depth		INF Depth	
Algorithm	Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.
MTS-c	1063	2874	1063	2874	1063	2874	1063	2874
MTS-d	937	2469	937	2469	937	2469	937	2469
RTEF	1000	1625	400	936	118	309	32	81
RTTES	641	933	212	464	80	174	22	49
MTES	531	907	212	444	82	174	23	55
A*	20	168	20	168	20	168	20	168

U-type grids

S.Depth	10 Depth		20 Depth		40 Depth		INF Depth	
Algorithm	Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.
MTS-c	1063	2855	1063	2855	1063	2855	1063	2855
MTS-d	1063	2498	1063	2498	1063	2498	1063	2498
RTEF	1063	1955	641	1319	188	665	78	406
RTTES	793	1272	400	730	122	353	45	234
MTES	793	1257	400	747	133	348	57	233
A*	8	104	8	104	8	104	8	104

to move in horizontal and vertical directions, these two manoeuver patterns have equal path distances to a fixed location. Although there is nothing wrong with these for fixed targets, this is not the case for moving targets since the strategy difference significantly affects the behavior of the prey in U-type grids. To be more effective, diagonal moves are required, and to continuously move diagonally, agents should take horizontal and vertical moves alternately. That is what MTES does, but A* does not. When we examine the results furthermore, we see that vision range does not affect the solutions significantly, except in U-type grids since the obstacle sizes in these grids are the largest. Similarly, search depth does not also affect the results much except in maze grids with 35% obstacles and U-type grids. Another important fact we observed about the search depths is that even with very small depths,

MTES always performs better than MTS, and almost always performs better than RTTES and RTEF.

We also examined the step execution times of the algorithms running on an AMD Athlon 2500+. In Table I, the minimum and average number of moves/second in maze and U-type grids are shown. According to the results, we conclude that MTS-c and MTS-d have low and almost constant step execution times whereas the efficiency of MTES, RTTES, and RTEF is tied to the search depth and obstacle ratio, and hence appropriate depth should be chosen according to the required efficiency and grid type. With respect to worse case performance, A* seems to be the worst as expected.

VI. Conclusion

In this correspondence, we have focused on real-time moving target search in partially observable grid worlds, and proposed a predator algorithm and a prey algorithm. The experiments have shown that MTES is able to make use of environmental information acquired during the search successfully, and brings significant performance improvement over RTTES, RTEF, and MTS, and competes with A* with respect to path length. During the test runs, we have also observed that the two MTS versions are significantly different from each other. Although, MTS-d performs acceptably good, MTS-c almost never offers good solutions. In terms of step execution times, MTS-c and MTS-d seem to be the most efficient algorithms, and almost spend constant time in each move. However, their solution path lengths are usually unacceptably long. RTEF, MTES, and RTTES follow MTS, respectively, and their efficiency is inversely proportional to the increase in obstacle density. Finally, A* is almost always the worst.

References

[1] T. Ishida and R. Korf, "Moving target search: A real-time search for changing goals," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 6, pp. 97–109, Jun. 1995.

[2] C. Undeger and F. Polat, "Real-time edge follow: A real-time path search approach," *IEEE Trans. Syst., Man Cybern., C*, vol. 37, no. 5, pp. 860–872, Sep. 2007.

[3] C. Undeger and F. Polat, "RTTES: Real-time search in dynamic environments," *Appl. Intell.*, vol. 27, pp. 113–129, 2007.

[4] S. Russell and P. Norving, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[5] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Proc. Int. Joint Conf. Artif. Intell.*, Aug. 1995, pp. 1652–1659.

[6] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, Jun. 2005.

[7] S. Koenig, M. Likhachev, and X. Sun, "Speeding up moving-target search*," in *Proc. 6th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2007, pp. 188–188.

[8] I. Kamon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 1996, vol. 1, pp. 429–435.

[9] R. Korf, "Real-time heuristic search," *Artif. Intell.*, vol. 42, no. 2–3, pp. 189–211, 1990.

[10] S. Koenig and M. Likhachev, "Real-time adaptive a*," in *Proc. 5th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2006, pp. 281–288.

[11] Y. Kim, D. Gub, and I. Postlethwaite, "Real-time path planning with limited information for autonomous unmanned air vehicles," *Automatica*, vol. 44, pp. 696–712, 2008.

[12] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proc. Int. Conf. Intell. Robots Syst.*, 2002, vol. 3, pp. 2383–2388.

[13] D. Hsu, R. Kindel, J. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[14] Y. Ishiwaka, T. Sato, and Y. Kakazu, "An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning," *Elsevier J. Robot. Auton. Syst.*, vol. 43, no. 4, pp. 245–256, 2003.

[15] M. Goldenberg, A. Kovarsky, X. Wu, and J. Schaeffer, "Multiple agents moving target search," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI 2003)*, pp. 1536–1538.

[16] R. Kota, S. Braynov, and J. Llinas, "Multi-agent moving target search in a hazy environment," in *Proc. IEEE Int. Conf. Integr. Knowl. Intensive Multi-Agent Syst.*, Oct. 2003, pp. 275–278.