REAL-TIME EDGE FOLLOW: A NEW PARADIGM TO REAL-TIME PATH SEARCH

Cagatay Undeger
Modeling and Simulation Section
Defense Technologies Enginering Inc.
35. Sok. No.28 Balgat
06520 Ankara,
Turkey
E-mail: cundeger@ceng.metu.edu.tr

Faruk Polat
Department of Computer Engineering
Middle East Technical University
06531 Ankara,
Turkey
E-mail: polat@ceng.metu.edu.tr

Ziya Ipekkan
Scientific Decision Support Center
Turkish General Staff (TGS)
Bakanliklar
06100 Ankara,
Turkey
E-mail: zipekkan@tsk.mil.tr

KEYWORDS
Computer-generated forces, real-time path search, maze problems.

ABSTRACT

Path searching and mission planning are challenging problems in many domains such as war games, robotics, military mission planning, computer-generated forces, etc. The objective of this study is to develop a real-time path-planning algorithm to accomplish specified missions on large landscapes. For that purpose, a real-time goal-directed path search algorithm, Real-Time Edge Follow (RTEF), which can work on fully known, partial known or completely unknown maze environments, is developed. RTEF aims to find a path from a staring point to a static or dynamic target point in real-time. The basic idea behind the RTEF is to let the agent eliminate closed directions (the directions that cannot reach the target point) by analyzing obstacle edges in order to decide on which way to go (open directions). For instance, if the agent has a chance to realize that moving to north and east won't let him reach the goal state (although the target is at north-east), then he will prefer going to south or west. RTEF finds out these open and closed directions, so decreasing the number of choices the agent has and significantly shortening the path. The method is tested on large mazes and compared with Real-Time A*. We observed that RTEF always performs much better than RTA* when solution quality is considered and usually better when total time spent to reach the goal state is considered (especially on complicated mazes). RTEF frequently gives high solution quality, which is in most cases near to optimal solution, and never needs to return to a previously visited cell while on the way.

INTRODUCTION

Multi agent systems can be used to model computer-generated environments where intelligent agents react suitably to various events. Many of the applications in this context need realistic environment generation, efficient search algorithms and heuristics suitable for real-time simulations. Multi agent systems are integrated into these simulations for supporting automatic and semi-automatic human and group behaviors to complete a given mission. Planning a mission usually means to plan a sequence of actions that lead to the goal-state.

The problem of path planning can be described as finding a sequence of state transitions from some initial state (starting point) to a goal state (target point), or finding out that no such sequence exists. Path-planning algorithms can be off-line or on-line. Off-line path planning algorithms like A* [Russell and Norving 1994] find the whole solution before starting execution. They plan paths in advance and usually find optimal solutions. Their efficiency is not considered to be crucial and the agent just follows the generated path. Although this is a good solution for a static environment, it is completely infeasible for dynamic environments, because if the environment or the cost function changes, the remaining path may need to be re-planned, which is not efficient for real-time applications. Real-time path planning algorithms such as Real-Time A* (RTA*), Learning Real-Time A* (LRTA*) [Knight 1993; Korf 1090], Moving Target Search [Ishida and Korf 1995], Bi-directional Real-Time Search [Ishida 1996], Real-Time Horizontal A* [Undeger 2001], D* [Stentz 1994], Focused D* [Stentz 1995] are on-line and offer more efficient solutions. Some of them produce optimal solutions for dynamic changes such as D* and Focused D*, and some only bring efficiency but not optimality such as Real-Time A*.

In this paper, we have proposed a real-time path search algorithm, "Real-Time Edge Follow" (RTEF), which provides both efficiency and solution quality on maze environments. The algorithm is capable of searching a path in real-time to reach a static or dynamic target on a fully known, a partial known or a completely unknown mazes. It can also be used in real-life applications uncluding robotics. RTEF is tested on randomly generated mazes and large terrain data, and compared with RTA*. The results showed that RTEF performed better in both solution quality and execution time.

In Section 2, a survey of related work on path planning is given. Our real-time path search algorithm, Real-Time Edge Follow is described in detail in Section 3. The

performance analysis of RTEF is given in Section 4. Finally, the conclusion is given in Section 5.

RELATED WORK

Multi-agent systems are used in many domains such as robotics, computer generated forces, games, training, RoboCup soccer and their simulators. In robotics [LaValle and Kuffner 1999] and RoboCup soccer [Jensen and Veloso 1998], intelligent planning aims to find out ways for interacting with the physical world, which makes the problem hard to solve. In contrast, intelligent planning for computer generated forces and games [Pew and Mavor 1998; Undeger et al. 2000; Undeger 2001; Baxter and Horn 1999; Gelenbe 1998; DeLoura 2000] aim to generate behaviors similar to the real world in virtual environments. Simulating real world actions in a virtual environment is basically used to test some conditions that are not possible or hard to generate in the real world. For example, intelligent agents that behave much like real world entities are frequently used for pilot trainings in flight simulators [Jones et al. 1993, 1994]. In such simulators, realistic modeling of agent behaviors is very important for the realism of training.

In multi agent simulations, evaluating the environment information, learning and reacting in time is essential. Erol Gelenbe proposed modeling computer-generated forces with learning stochastic finite-state machines whose state transitions are controlled by state and signal dependent random neural networks [Gelenbe 1998]. In Knuffner's approach [Kuffner and Latombe 1999], rendering off-screen from the character's point of view and real-time path planning is used. His path-planning module aims to find a collision free path between a starting and ending point over the 3D terrain using the information gathered from vision based perceptions. In the study of Knuffner, the terrain is divided into embedded graph cells, which have vertical, horizontal and diagonal costs of walking through. Then, the suitable path is found using Dijkstra's algorithm, which is actually an optimal off-line path-planning algorithm integrated into a real-time application. Using an off-line path-planning algorithm in a real-time application is not suitable for large terrains. By the help of some guidance such as admissible heuristics can increase efficiency of off-line path planning algorithms. A* [Russell and Norving 1994] is one of best-known efficient path planning algorithms, which is guided by a heuristic function. A* always finds the optimal solution and uses linear distances between points for the heuristic function. Optimal path planning algorithms cannot be used for large and dynamic landscapes because of its complexity. To avoid this drawback, some partial path update algorithms are also proposed such as D* [Stentz 1994] and focused D* [Stentz 1995]. These algorithms plan an off-line path, let the agent follow the path, and if any new environment information is gathered, they partially re-plan the existing solution. But some times, a small change in the environment may cause to re-plan almost a complete path, which may take a long process time.

A number of algorithms exist for supporting real-time simulations such as Real-time A* (RTA*) and Learning Real-Time A* (LRTA*) [Knight 1993; Korf 1990]. RTA* uses a greedy search strategy and a heuristic together to guide the search. It guarantees to find a solution if one exists, but the solution may not be optimal. In this paper, we have proposed a new Real-Time path search algorithm for maze environments.

There are also some path-planning algorithms that use random search techniques such as genetic algorithms, random tree generators. In [Sugihara and Smith 1997], an adaptive path-planning algorithm based on genetic approach is proposed. In this study, they assumed that a valid path that is not optimal is initially found and they refine this given path by genetic algorithm. Considering this concept, our previously developed off-line path-planning algorithm, Linear Search Path Finder [Undeger 2001] seems to be applicable to this case successfully. In the study of LaValla and Knuffner [LaValle and Kuffner 1999], a randomized planning technique based on a version of random tree generation called rapidly exploring random tree is presented. They generated two random trees starting from the goal and the target points, and try to catch an intersection among the points of distinct trees to find a path.

REAL-TIME EDGE FOLLOW

RTEF is developed for maze-type problems and aims to find a path from a staring point to a target point in real-time. The basic idea behind the RTEF is to let the agent eliminate closed directions (the directions that cannot reach the target point) in order to decide on which way to go, (open directions). For instance, if the agent has a chance to realize that moving to north and east won't let him reach the goal state, than he will prefer going to south or west. RTEF finds out these open and closed directions, so decreasing the number of choices the agent has.

The environment is assumed to be a grid world with obstacles, where the cells having obstacles are marked with a non-zero value and the rest is filed with zero. Initially, the agent is at a starting point and the goal is to reach a static or dynamic target. At each move, RTEF algorithm is executed to eliminate closed directions from the current cell in order to select the next move from a set of open directions. The cost of moving to the next cell + the distance to the target is used for the heuristic function to select one of the open alternatives. After doing the move, the previous cell is marked as an obstacle; so loops are prevented. The set of previously marked cells are called the history of the agent. In exploration mode with an unknown maze, the history has to be cleared when a newly discovered obstacle blocks the way that was open before.

The algorithm is constructed on the idea that every obstacle has a boundary, which is actually formed from a set of connected edges shaping the object. The obstacles are

defined as a set of merged cells in a grid world. RTEF splits the moving directions from each other by a set of rays sent away from the agent, and analyze each region to discover whether it is closed or open.

Moving Directions and Sent-Rays

RTEF accepts the possible moving directions as north, south, east and west. Although there are many interval values of these directions, a set of basic directions is chosen to partition the area for the sake of efficiency. The idea is not to find an exact direction, but to choose a general direction having a left and right angle limit. The algorithm splits the visual environment into four regions by sending 4 rays away from the agent. The rays are sent along 4 diagonal directions (the angle between two adjacent rays is 90 degree). The rays go away from the agent until hitting an obstacle and hence split the area into 4 different regions. Some of these regions are on the way to the target point and some are not. This is illustrated in Figure 1.

Each ray hits the boundary of one of the obstacles. The cells on the way of a ray are followed until a blocked cell is detected. The boundary of the grid world is also marked as blocked in order to prevent infinitely going rays. The first ray is assumed to be on the northwest diagonal. This is illustrated in Figure 2. A ray is assumed to hit an obstacle, when any cell on the way is marked as blocked.

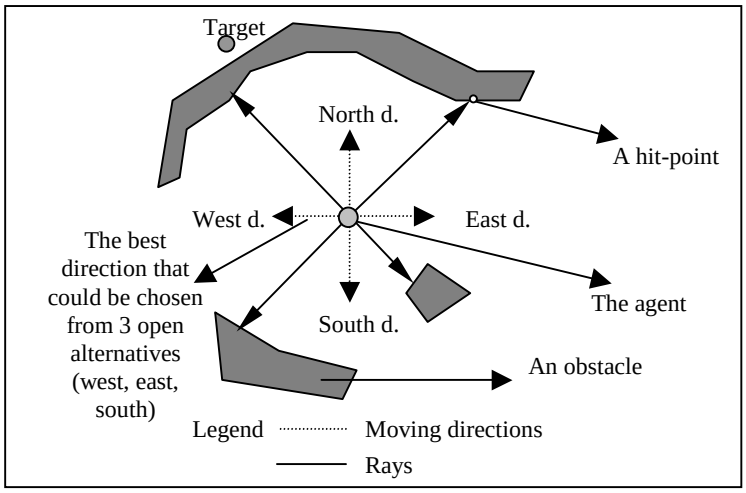

Figure 1: Sending rays until hitting an obstacle to split the basic directions

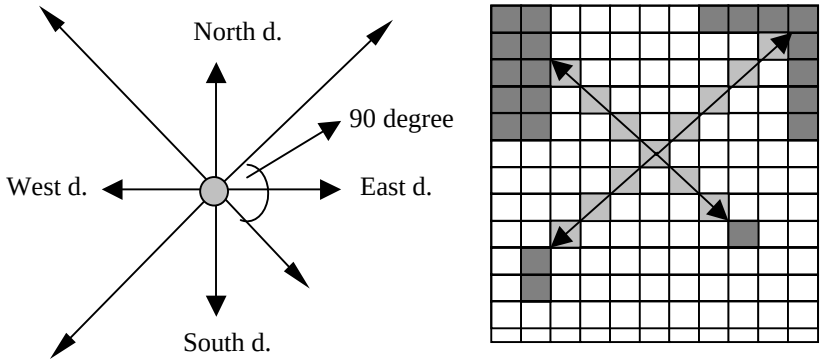

Figure 2: Rays, Directions and hitting: Rays and directions are numbered from 0 to 3 in clockwise direction on the left. The figure on the right shows the cells on the way of 4 rays.

Following Edges

Four rays split the area around the agent into 4 regions. Some of these limited regions are closed and target point is inaccessible from any point located inside these regions. If all are closed, that means the target is inaccessible from the current state or location. To detect which ones are closed, the boundaries of obstacles that the rays hit must be analyzed. The edges on the boundaries are followed and if edges are followed clockwise or counter-clockwise directions starting from a hit-point, we always return to the same point. By following edges and returning to the same starting point on the boundary, a polygonal area is formed (the boundary of the obstacle is detected). We call this polygonal area an "*island*". There are two kinds of islands: outwards facing and inwards facing, which are shown in Figure 3. If the target is inside an outwards facing island or

outside an inwards facing island, that means the target is inaccessible from the current location.

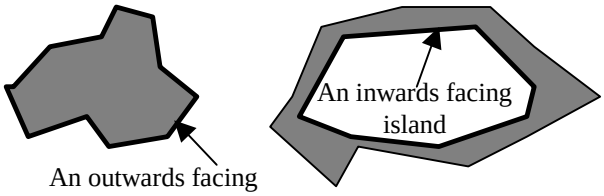

Figure 3: Island types: outwards facing (top), inwards facing (bottom)

If we reach the hit-point of another ray while following edges (only the first one reached is considered), we have an additional polygonal area (the two rays are also included) called *"hit-point-island"*. This polygonal area is illustrated in Figure 4.

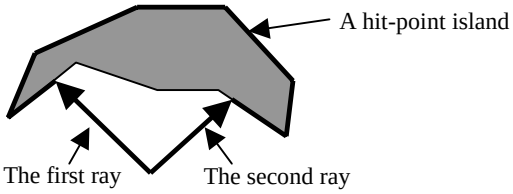

Figure 4: Two rays hitting the same obstacle at two different points

A hit-point-island borders one or more moving directions. If the target point is not inside the hit-point-island, it means that all the directions that are bordered by the hit-point-island are closed. Otherwise all the directions that are not bordered by the hit-point-island are closed. This is illustrated in Figure 5.

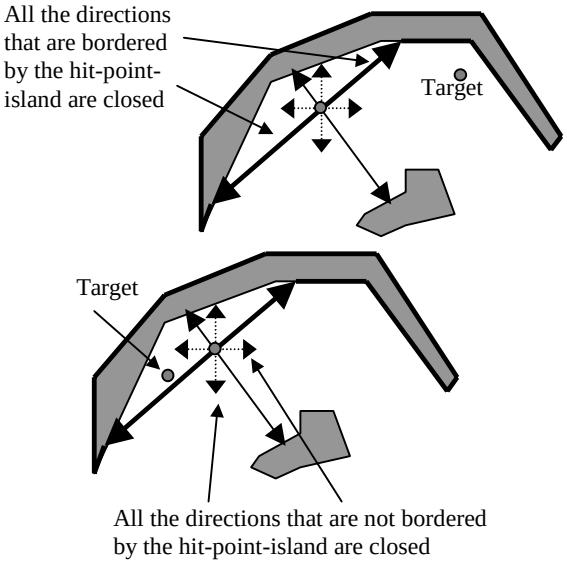

Figure 5: Analyzing hit-point islands and eliminating moving directions

The edge following algorithm works on grid worlds and it is the most time consuming part of RTEF. There are two edge follow directions. One follows the edges from the left side of the hit-point and one follows from the right side. If we follow edges from the left, we have to choose the next edge, which is connected to the left side of the current edge and similar process is done for the right side. This is depicted in Figure 6.

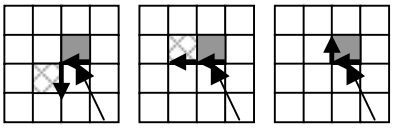

Figure 6: Edge detection: The figure illustrate next possible states when starting from south side of a cell. The possibilities are tested from smaller angles to larger angles.

After finding the island and the hit-point-island, the target must be checked if it is located inside these polygonal areas or not. Implementing an inside test is not so complicated. For testing whether the coordinate (*tx*,*ty*) is inside a polygon *P* or not, we just need to count the number of edges in *P* that are on the left side of *ty* and intersecting the horizontal line passing on *ty*. If the edge count is an odd number, the point is inside the polygon else it is not.

Detecting Closed/Open Regions

To detect closed and open regions, "edge following process" and "inside polygon tests" are used together. To follow edges from only one direction is enough for the detection, but one direction has to be chosen and used for all the rays. The C-like pseudo code for left side edge following is shown in Figure 7. When open and closed directions are detected, one of the open directions is chosen and one-step move is performed. The simplest heuristic is the cost of moving to the next cell + the distance of next cell to the target. The direction, which minimizes the cost, is selected for the next move.

Although the algorithm finds only the open directions, there is a high possibility of getting into loops, especially when there is more than one open direction at a time. For example, assume that the agent is at the same y coordinate as the target is, and there is one choice "moving north". So, the agent moves north. Consider the next state has two open regions: north and south. South is the previous cell. If the agent selects the south, it is obvious that it will go into a loop. The agent selects the south, because being at the same y coordinate as the target is, has less cost than being at two cells up from the y coordinate of the target (the distance to the target is shorter). This is only one of many possibilities. The simplest solution to avoid the loops is to mark the previous point as an obstacle. So the agent never has a chance to visit the same point again.

```
For each ray-sent
   Find hit-point of the ray;
   Follow the edges from left side of the hit-point;
   Find out the island and hit-point-island;
   if (an hit-point-island is found)
   {
      if (target_x,target_y) is inside the hit-point-
      island)
      {
        if ((target_x,target_y) is inside the island)
         {
            if ((agent_x,agent_y) is not inside the
            island)
                  close all the directions; // the
                                           target is
                                           inaccessible
                                           ;
         }
         else
         close the directions that are not bounded by
         hit-point-island.
      }
      else close the directions that are bounded by
      hit-point-island.
   }
   else
   {
    if ((the island is outwards facing and
    (target_x,target_y) is inside the hit-point-island) or
    (the island is inwards facing and (target_x,target_y)
    is not inside the hit-point-island))
       close all the directions; // the target is
                                 inaccessible;
   }
```

Figure 7: The C-like pseudo code for detecting closed and open regions

Exploring and Adaptation to Dynamic Environments

If the maze is not given in advance or just a part of the maze is known, the agent has to explore and learn the environment in real-time. The algorithm is applicable for real-time exploration. It will immediately adapt itself to the new condition, but there can be a problem because of marking the cells traveled with obstacle. Because the agent assumes that the rest of the way is opened, so it marks the previous cells as obstacle, but when it explores the area and notices that the region is closed, it may be stuck in a completely closed region. All we need to do is to clear the history when all the directions are closed.

In real life, the targets, the obstacles and the threats usually have a dynamic nature. For RTEF having a dynamic environment is not a big problem in fact. The algorithm can also be adapted to a change in the cost function rapidly. The only problem that may occur is blocking. For example, while the target is moving, it may enter to a previously blocked region, but if the target is blocked, the agent rapidly detects that all the directions are closed and the history is cleared.

One disadvantage of clearing history is forgetting previous experience and returning back to same regions again. While marking traveled cells as obstacles, one good thing is also done. The noise (small obstacles) on the maze is cleared by connecting them with history lines. So clearing the history causes noise to increase again.

Complexity of RTEF

The worst case complexity of RTEF for each move is O(w x h), where w is the width of maze and h is the height of maze. But almost never we have this complexity. In general, it is much more smaller than this. The efficiency of a move can be explained with total number of edges followed. Other operations are very efficient compared to edge following process. The observed worst-case frame rates for different sized mazes are shown in Table 1. As seen, the seconds per move increases almost proportional to the number of cells.

Table 1: Worst-case performances on sample mazes: Sec/M = seconds per move, M/sec = moves per second, Increase = Increase compared to previous column

	100x100	200x200	400x400	800x800	1600x1600
Sec/m	0.005	0.024	0.110	0.462	1.935
M/sec	200.000	41.666	9.090	2.164	0.516
Increase	-	4.800	4.583	4.200	4.188

PERFORMANCE ANALYSIS

The test environment is implemented in Borland Delphi programming language under Windows platform. The tests are run on an Intel Celeron-466 with 128 MB memory. Both RTEF and RTA* (described in subsection “Real-Time A*”) are implemented. We observed that the solution quality of RTEF is always better than RTA* in both simple and complicated mazes. The efficiency (seconds per move) of RTEF is satisfactory in most cases, but not better than RTA*. The efficiency of RTEF depends on the environment complexity, while the efficiency of RTA* is nearly constant.

Three different versions of RTEF algorithm are realized and tested using randomly generated mazes, and compared to RTA*. The first RTEF algorithm knows the entire maze in advance, so has a chance of better planning. The second and third ones don’t know the maze. The agents can see the neighboring cells to a certain depth. They completely see a square region with size d x d, where d is twice of their visual depth plus one. For example, if the observation depth is 5, the agent knows a square of 11x11 cells (5 left, 5 right, 5 up and 5 down). The agent can build the map with real data in time.

Real-Time A* (RTA*)

RTA*, which RTEF is compared to, is a greedy search algorithm that uses heuristics to direct the search. It evaluates the costs of the neighbor voxels at the current position and jumps to the voxel having minimum cost. While jumping to the next voxel, the algorithm writes 1 plus the cost of the second best neighbor to the previous voxel. This is illustrated in Figure 8.

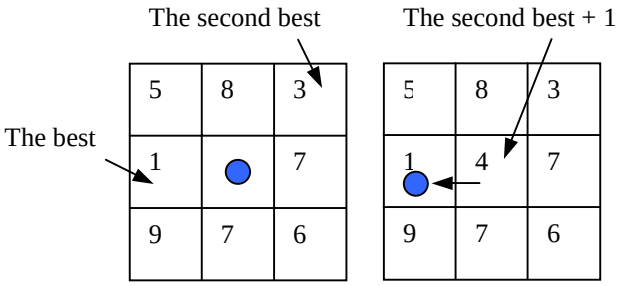

Figure 8: State transition of Real-Time A*

The algorithm is effective and complete for maze environments, but if the terrain is large and there are many semi-closed regions having large open areas inside, the agent may be stuck in the regions for a long time, because the search strategy is too local, only the neighbor voxels are evaluated. In addition RTA* uses much memory than RTEF, because it needs an additional array to store cost updates. And RTA is also not applicable to dynamic environments in practice.

Test Results of RTEF

The mazes with size 100x100 and 500x500 are randomly generated. The agents and the targets are positioned on upper-left, lower-right or upper-central, lower- central side of the mazes. Total time passed, average moves/sec and the number of moves to reach the target point are used to evaluate the results. We evaluated the efficiency and solution quality of RTEF. The average values of test results are shown in Table 2 and 3.

As seen from the results, RTEF always gives better solution quality than RTA* on the average. The difference significantly increases if complicated mazes are used. The RTEF again has a good solution quality when unknown mazes are explored. The best solution quality over RTA* is 995.741 times better than RTA* on the average. The worst solution quality of RTEF is again better than RTA*, which is 1.010 times better than RTA*. The average moves/sec (moves per seconds) of RTA* is 2395 (almost constant), where average moves/sec of RTEF is 371 (varies from 29 moves/sec to 991 moves/sec). This moves/sec is much more smaller than RTA*, but acceptable for soft real-time systems. In robotics, doing unneeded physical actions is much more time consuming than thinking and doing good actions, so the moves/sec is not a disadvantage at all. In real world, the time to walk to the next state will usually take more time than thinking phase, and the next move can be calculated while walking to the next state. If we examine total time spent to reach the goal state, an interesting result is observed. In very simple mazes the efficiency ratio of RTA* is better than solution quality ratio of RTEF. But in simple mazes, they go head to head. And after simple mazes, the solution quality and efficiency generally seems to be better than RTA* and after average mazes, the RTEF is better both in efficiency and solution quality. Some sample snapshots are shown in Figure 9, 10.

CONCLUSION

In this paper, we have studied the concept of computer-generated forces in order to construct a real-time simulation system. We have generated complicated landscapes similar to real ones and tried to solve real-time path planning problem for mission planning purposes. A test environment is generated and a set of scenarios is performed to evaluate performance of developed algorithm. We observed that RTEF performs better than RTA* in most cases. The algorithm is also applicable to robotics and real-time observation, and it frequently gives high solution quality, which is in most cases near to optimal solution.

By introducing new heuristics, the solution quality can be increased, but these methods may reduce the efficiency of the algorithm. For example, some evaluation methods can be applied using edge follow information.

As a result of our observations, we state that the real-time path planning techniques can be improved by increasing the visual search depth, which helps a lot to escape earlier from the local semi-closed regions. But we have also noticed that there is much to do for better intelligent search strategies. More human-like evaluation techniques are needed to go one step forward.

Figure 9: A Maze with 5 cell corridors: RTA* performs 226772 moves in 102.034 seconds (top), RTEF performs 1664 moves in 46.474 seconds (bottom). Black regions are traveled cells. RTA* may travel a cell more than once.

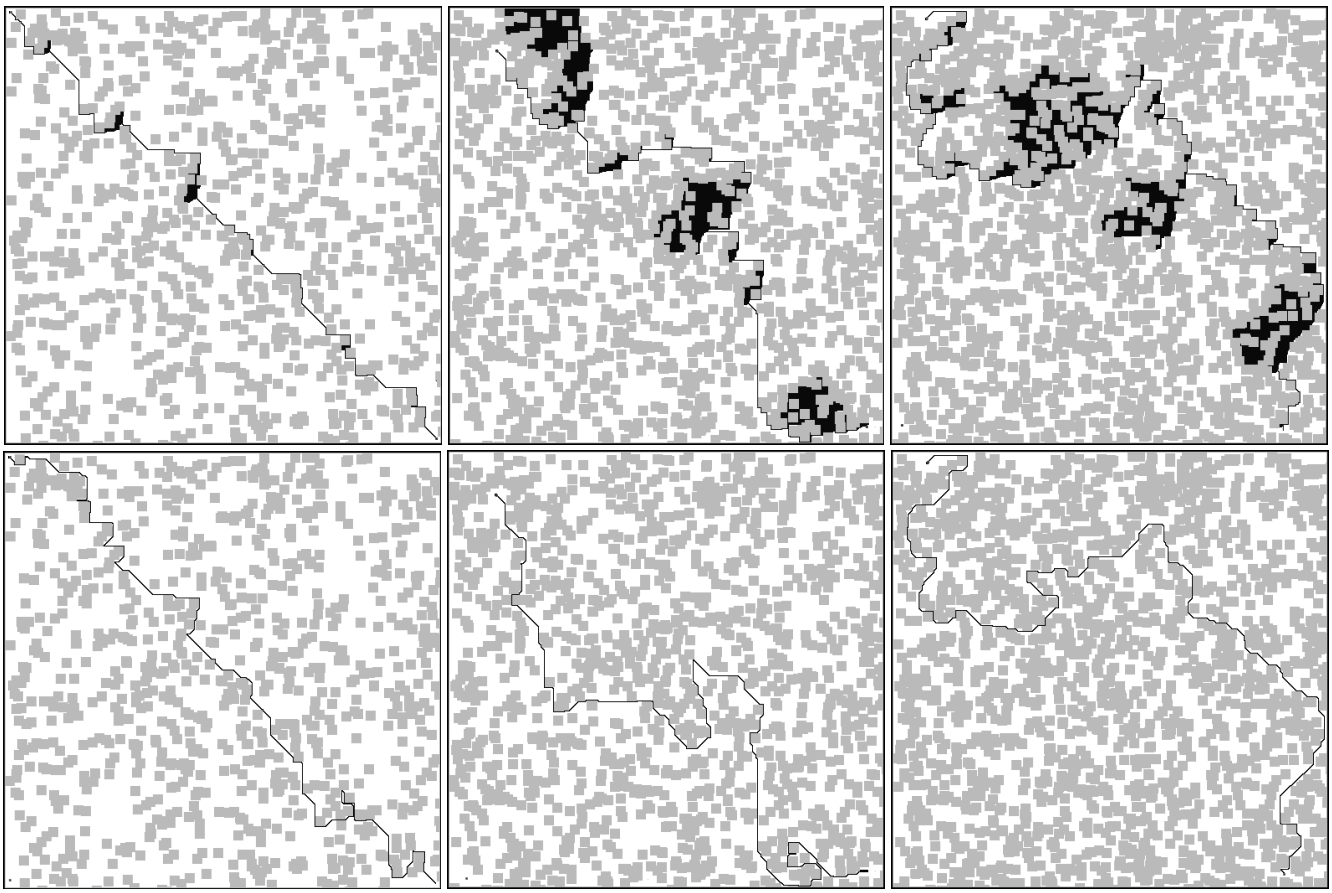

Figure 10: A Simple Maze (left): RTA* performs 2163 moves in 0.890 seconds (top-left), RTEF performs 1325 moves in 2.840 seconds (bottom-left). A Complicated Maze (Middle): RTA* performs 6157 moves in 27.024 seconds (top-middle), RTEF performs 1487 moves in 13.680 seconds (bottom-middle). A Very Complicated Maze (right): RTA* performs 76387 moves in 34.105 seconds (top-right), RTEF performs 1670 moves in 20.870 seconds (bottom-right).

Table 2: Test results (mazes filled with randomly distributed noise)

Average of Simple Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	1.665	2408	3950	1.000	1.000
RTEF (maze is fully known)	2.850	449	1282	0.584	3.081
RTEF (observation depth 5)	3.490	507	1771	0.477	2.230
RTEF (observation depth 10)	3.889	389	1513	0.428	2.610

Average of Average Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	4.058	2367	9718	1.000	1.000
RTEF (maze is fully known)	4.695	218	1026	0.864	9.471
RTEF (observation depth 5)	2.653	660	1753	1.529	5.543
RTEF (observation depth 10)	3.018	404	1221	1.344	7.959

Average of Complicated Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	57.685	2468	94365	1.000	1.000
RTEF (maze is fully known)	19.138	74	1428	3.014	66.081
RTEF (observation depth 5)	29.094	285	8317	1.982	11.346
RTEF (observation depth 10)	18.802	300	5652	3.068	16.695

Average of Very Complicated Mazes: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	58.766	2415	142089	1.000	1.000
RTEF (maze is fully known)	21.932	62	1373	2.679	103.487
RTEF (observation depth 5)	21.897	422	9245	2.683	15.369
RTEF (observation depth 10)	21.960	369	8123	2.676	17.492

Table 3: Test results (5 and 1 cell corridors with sizes 500x500 and 100x100)

Average of mazes with 5 cell corridors: 500x500 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	843.917	2495	2176692	1.000	1.000
RTEF (maze is fully known)	73.257	29	2186	11.519	995.741
RTEF (observation depth 5)	581.190	122	71377	1.452	30.495
RTEF (observation depth 10)	167.415	119	19983	5.040	108.927

Average of mazes with 1 cell corridors: 100x100 cells	Time in Seconds	Average Moves/Sec	Number of Moves	Efficiency of RTEF, RTA* Time / Time	S. Quality of RTEF, RTA* Moves / Moves
RTA*	3.861	2434	9201	1.000	1.000
RTEF (maze is fully known)	2.801	117	329	1.378	27.966
RTEF (observation depth 5)	2.808	434	1220	1.375	7.541
RTEF (observation depth 10)	2.385	295	704	1.618	13.069

REFERENCES

Baxter, J.W. and Horn, G.S. 1999. "A Model for Co-Ordination and Co-Operation Between CGF Agents." In *Proceedings of 8th conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida, 101-111.

DeLoura, M.A. 2000. *Game Programming Gems.* Charles River Media.

Gelenbe, E. 1999. "Modelling CGF with Learning Stochastic Finite-State Machines." In *Proceedings of* 8th *conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida, 113-115.

Ishida, T. and Korf, R.E. 1995. "A Moving Terget Search: A Real-Time Search for Changing Goals." *IEEE Trans Pattern Analysis and Machine Intelligence*, Vol.17, No.6, 97-109.

Ishida, T. 1996. "Real-Time Bidirectional Search: Coordinated Problem Solving in Uncertain Situations." *IEEE Trans Pattern Analysis and Machine Intelligence*, Vol.18, No.6.

Jensen, R.M. and Veloso, M.M. 1998. "Interleaving Deliberative and Reactive Planning in Dynamic Multi-Agent Domains." In *Proceedings of the AAAI Fall Symposium on Integrated Planning for Autonomous Agent Architectures*, AAAI Press.

Jones, R.M., Tambe, M., Laird, J.E. and Rosenbloom, P.S. 1993. "Intelligent Automated Agents for Flight Training Simulator." In *Proceedings of* 3th *conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida, 33-42.

Jones, R.M., Laird, J.E., Tambe, M. and Rosenbloom, P.S. 1994. "Generating Behavior in Response to Interacting Goals." In *Proceedings of* 4th *conference on Computer Gererated Forces and Behavioral Representation*, Orlando, Florida.

Knight, K. 1993. "Are Many Reactive Agents Better Than A Few Deliberative Ones?" In *Proceedings of The International Joint Conference on Artificial Intelligence*, 432-437.

Kuffner, J.J. and Latombe, J.C. 1999. "Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans." In *Proceedings of Computer Animation*, IEEE, 118-127.

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

LaValle, S.M. and Kuffner, J.J. 1999. "Randomized Kinodynamic Planning." In *Proceedings of IEEE International Conference on Robotics and Automation*, ICRA'99, Detroit, MI.

Pew, R.W. and Mavor, A.S. 1998. *Modeling Human and Organizational Behavior: Application to Military Simulations*, National Academy Press.

Russell, S. and Norving, P. 1994. *Artificial Intelligence: a Modern Approach*, Prentice Hall.

Stentz, A. 1994. "Optimal and Efficient Path Planning for Partially-Known Environments." In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA '94*, Vol.4, 3310–3317.

Stentz A. 1995. "The Focussed D* Algorithm for Real-Time Replanning." In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI'95.

Sugihara K. and Smith, J.K. 1997. "Genetic Algorithms for Adaptive Planning of Path and Trajectory of a Mobile Robot in 2D Terrains." Technical Report, number ICS-TR-97-04, University of Hawaii, Department of Information and Computer Sciences.

Undeger, C., Isler, V. and Ipekkan, Z. 2000. "An Intelligent Action Algorithm for Virtual Human Agents." In *Proceedings of the* 9th *Conference on Computer Generated Forces and Behavioral Representation*, Orlando, Florida, 25-33.

Undeger, C. 2001. *Real-Time Mission Planning For Virtual Human Agents.* M.S. Thesis in Computer Engineering Department of Middle East Technical University, Ankara, Turkey.

AUTHOR BIOGRAPHY

CAGATAY UNDEGER received his B.Sc. degree from Kocaeli University, Turkey in 1998 and went to the Department of Computer Engineering, Middle East Technical University, where he has worked as a research assistant and obtained his M.S. degree in 2001. He is currently doing his Ph.D. in the same university and studing on Modeling and Simulation within Defense Technologies Engineering Inc.

FARUK POLAT is an associate professor in the Department of Computer Engineering of Middle East Technical University, Ankara, Turkey. He received his B.Sc. in computer engineering from the Middle East Technical University, Ankara, in 1987 and his M.S. and Ph.D. degrees in computer engineering and information science from Bilkent University, Ankara, in 1989 and 1993 respectively. He conducted research as a visiting NATO science scholar at Computer Science Department of University of Minnesota, Minneapolis in 1992-93. His research interests include artificial intelligence, multi-agent systems and object oriented data models.

ZIYA IPEKKAN has been serving as an operations research analyst at TGS for more than twelve years. Lt.Col.Ziya İpekkan is currently the leader for Force Structure Analyses Team at the position since August 2000. He is responsible for conduct of studies and analysis of joint concepts, the warfighting capabilities and plans, and force structures.