Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 745219, 6 pages
doi:10.1155/2009/745219

Research Article

Performance Simulations of Moving Target Search Algorithms

Peter K. K. Loh[1] and Edmond C. Prakash[2]

[1] *Division of Computer Science, School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798*
[2] *Department of Computing and Mathematics, Manchester Metropolitan University, Chester Street, Manchester M1 5GD, UK*

Correspondence should be addressed to Peter K. K. Loh, askkloh@ntu.edu.sg

Received 9 April 2008; Revised 9 July 2008; Accepted 25 September 2008

Recommended by Kok Wai Wong

The design of appropriate moving target search (MTS) algorithms for computer-generated bots poses serious challenges as they have to satisfy stringent requirements that include computation and execution efficiency. In this paper, we investigate the performance and behaviour of existing moving target search algorithms when applied to search-and-capture gaming scenarios. As part of the investigation, we also introduce a novel algorithm known as abstraction MTS. We conduct performance simulations with a game bot and moving target within randomly generated mazes of increasing sizes and reveal that abstraction MTS exhibits competitive performance even with large problem spaces.

1. Introduction

In most RPG/adventure-based computer games, different types of bots act as adversaries to players. For example, in the recently launched Hellgate [1], players need to defend against and fight computer-generated demonic hordes. In such games, each generated bot is typically incorporated with suitable algorithms that enable it to locate and move towards a player. Each bot also has a "detection range or area" within which, it can detect a player. Unlike existing algorithms for static targets [2, 3], algorithmic designs for moving target search (MTS) algorithms are inherently more involved. The computational and memory requirements are significant. In some computer games, search algorithms can take up as much as 70% of CPU time [4–6]. This is due to the large number of objects (e.g., player, NPC, building, and walls) that need to be taken into consideration in the game environment [7]. The computational and memory requirements are also high when multiple bots communicate to find strategic paths as shown in our earlier work on Team AI [8]. Graphics also consume a significant proportion of computational resources leaving a limited amount for game AI [4]. Many contemporary graphics-intensive computer games are real-time, however, which means that bot responses to a player must be made as soon as possible. Such a scenario poses conflicting demands on the design of MTS algorithms.

This paper presents a study of the performance and behaviour of existing moving target search (MTS) algorithms in a maze search-and-capture scenario. As part of the study, we include a novel MTS algorithm called *abstraction MTS* and evaluate its performance and behaviour against the existing algorithms. Section 2 of this paper reviews three existing widely used MTS algorithms. Section 3 states the definitions and notations on which subsequent sections are based. The design of the abstraction MTS algorithm is detailed in Section 4. Section 5 describes the performance and behavioural analyses. The paper concludes with Section 6 followed by the Acknowledgments and References.

2. Survey

In a contemporary player-bot engagement-based computer game, the bot's response and behaviour are designed to be as realistic as possible. For example, a bot would be able to sense (detect) a player within its visibility region and not beyond. To make the game more engaging and playable, a typical bot should not be able to detect beyond some

finite region. Deep look-ahead search techniques that would be useful in certain games like chess would add an unfair advantage to a bot's capabilities and reduce the engagement and playability of the game [9]. In our work, therefore, we focus on algorithm designs that exploit "neighbourhood" information—information that can be determined within a finite detection region surrounding a bot. In this section, we review three well-known existing MTS algorithms for moving targets: basic moving-target search, weighted moving target search, and commitment and deliberation moving target search algorithms.

2.1. Basic Moving Target Search. The basic moving target search (BMTS) algorithm [10] is a generalisation of the learning real-time A* algorithm [11]. A matrix of heuristic values is maintained during the search process to improve their accuracy. The upper bounds on space and time complexities of B-MTS are N^2 and N^3, respectively, where N is the number of states in the problem space. Although MTS could converge to an optimum path (solution) eventually, it suffers from *heuristic depression* [10], which is a set of states with heuristic values not exceeding those of all neighbouring states. This may occur since heuristic value updates are localised leaving state inaccuracies over other areas in the problem space. In a heuristic depression, an agent repeatedly traverses the same subset of neighbouring states without visiting the rest. The agent may also continue to look for a shorter path even though a fairly good path to the target has been found. This would incur additional computational overheads and reduce bot performance during game play.

2.2. Weighted Moving Target Search. In certain scenarios, an optimal solution may not be needed and suboptimal paths may be found in a shorter time. The weighted moving target search (WMTS) algorithm [12] reduces the amount of exploration in MTS and accelerates convergence by producing a suboptimal solution. It allows a suboptimal solution with ε-error and δ-search (real-time search with upper bound) to achieve a balance in solution path quality and exploration cost. During the search, heuristic values are brought as close as possible to, but not reaching, the actual values. So, there is no guarantee that the search will eventually converge to an optimal solution. It is also important to determine a value of δ such that it can restrain exploration and find better solutions. The amount of memory space increases as δ increases.

2.3. Commitment and Deliberation Moving Target Search. With the commitment and deliberation moving target search (CDMTS) algorithm [10], the agent may ignore some of the target's moves. The agent only updates the target's moves when the agent is not in a heuristic depression. The *commitment* to the current target state increases if the agent moves in a direction where the heuristic value is reducing. If the agent is in a depression, it ignores the target's moves and commitment is set to 0. During *deliberation*, real-time search is performed when heuristic difference decreases, and offline search is performed when the agent is in heuristic depression. The offline search is used to determine the boundary of the heuristic depression. The CDMTS algorithm improves upon the efficiency of BMTS since the agent can exit from the heuristic depression faster.

3. Preliminaries

To simplify the problem, we prohibit movements in the third dimension (e.g., jumping, climbing) by either the bot or the player. The problem space is then reduced to that of a two-dimensional (2D) region, whereby movements of both bot and player are restricted to *left*, *right*, *forwards,* and *backwards*. We also require that the size of the problem space can be varied with obstacles generated and placed randomly. The unobstructed locations (no obstacles) in the maze are defined as a set of *states* and all traversals between a state and neighbouring states are defined by a set of edges with edge cost = 1.

In the following, we will use the terms "agent" or "bot" and "target" or "player" interchangeably. From each state, the agent or target can move to any of a maximum of four neighbouring states (representing locations to the *left*, *right*, *forward,* and *backward* directions) if unobstructed. The target moves randomly and slower than the agent so that the target will be acquired in a finite time. The goal for the agent is then to find a path from starting state s to the current target state g, if there is at least one path from s to g. The goal is accomplished if both agent and target occupy the same state. We define the following:

s = current state;

g = goal state;

s' = other state (not s or g);

$succ(s)$ = the set of successor states of s (neighbour states of state s);

$j(a, b)$ = total edge cost from state a to state b;

$h(a, b)$ = heuristic value from state a to state b;

$h^*(a, b)$ = minimal heuristic value from state a to state b considering all alternative paths;

$f(s, g) = j(s, s') + h(s', g)$, where f is the computed cost of a path from s to g.

To guarantee the completeness of the algorithm, we assume that the minimum heuristic value is never overestimated, that is, $h(a, b) \leq h^*(a, b)$ [10]. This is the case in the previous 3 algorithms surveyed. Information that includes the maze configuration, target position, and target movement pattern are not available initially. As shown in Figure 1, the agent can only detect the target's position if the target is within detection range r, regardless of whether there is an obstacle between them.

To simplify the calculation, the detection area is represented as a square and the value r is greater than one to emulate a bot equipped with above average human player's sensory-detection capabilities. For comparison purpose, the target (human) may be assumed to have a detection area with $r = 1$. This is typical in the more challenging

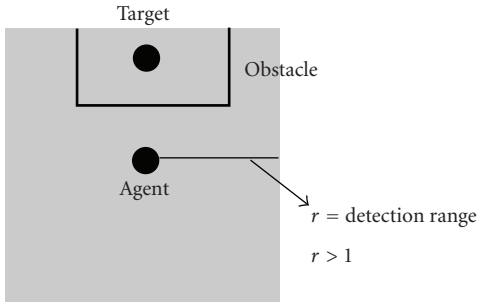
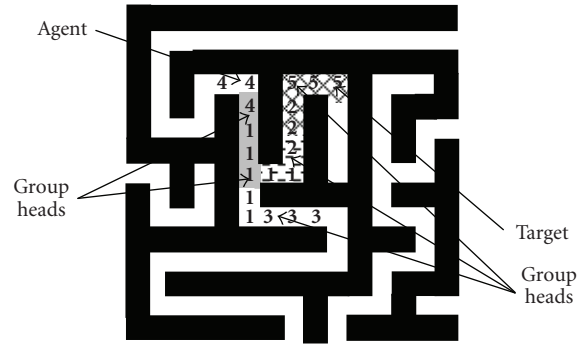


Figure 1: Target is detected within range.



Real move list 1
Real move list 2
Real move list 3

Figure 2: Abstraction maze example.

contemporary games. In the case of BMTS, WMTS, and CDMTS, the agent does not know the value of $h^*(a, b)$ until it finds an optimal solution. A path $(s_0, s_1, \ldots, s_n)$ is optimal if and only if $h(s_i) = h^*(s_i)$ for $0 \leq i \leq n$, where $h^*(s_i)$ equals the actual cost from s to the goal and $h(s_i)$ equals the heuristic value. Heuristic value is computed with the Manhattan distance method.

4. Abstraction Moving Target Search Algorithm Design

Each of the existing MTS algorithms employs a heuristic array table in its learning process. For each state s, we need to store the heuristic value with all states. That is, we need to store the state pairs: $h(s, k)$, where $k \in S$ (problem state space) except $h(s, s) = 0$. Group values are stored in a 2D-array, *abstract*. Also, since $h(x, y) = h(y, x)$, the total memory needed is upper-bounded by $(n^2 - n)/2$, where n is the number of states in S. To solve this problem, we apply the *abstraction maze* approach. Our approach is to have a 2-level search as illustrated by an example of an abstraction maze in Figure 2.

Figure 2 shows that when the agent detects the target, it checks if the target position belongs to a group. If so, the agent will determine the best *abstraction move list AL* and required *real move list(s) RL* to acquire the target. In this example, AL = $\{4, 1, 2, 5\}$. Using group numbers to simplify representation, three real move lists would be generated as follows: real move list 1 = $\{4, 1, 1, 1\}$, real move list 2 = $\{1, 1, 2\}$, and real move list 3 = $\{2, 2, 5, 5, 5\}$. Each *real move list* contains a movement path up to the next group head and the last leading up to the target.

Specifically, each node x in the abstraction maze may be labeled with a number which indicates an associated group, gr_x. These group values are stored in a 2D-array labeled *abstract*. A group p also has a *group head*, $hd(p)$. The group head is used as a base to measure the cost (distance) to all nodes in the same group. The distance from a node s to its group head, gr_s, must be less than some constant *abstractDistance*. That is, $j(s, hd(gr_s)) < abstractDistance$. In the example in Figure 2, *abstractDistance* is equal to 3. Each group also maintains a list of its neighbours. For example, the neighbours of group 1 are groups 2, 3, and 4.

In the initialisation step of the algorithm, the starting point of the agent is set at the location of the head of group 1 and *number AbstractNode* is set to 1. The variable *number AbstractNode* is defined as the number of groups that has been created. During exploration, after the agent has moved to a new state (position) s', it will check if this node has a group. If it has not, the agent will check if the nearest head distance is less than *abstractDistance*. If it is, then apply *setAbstract* method. In *setAbstract* method, the current node will be grouped with the nearest group head. If the nearest head distance is not less than *abstractDistance*, then *number AbstractNode* is increased by one and the current node will become a new group head. Formally, this is expressed as follows.

For each $s' \notin gr_i$, (where $gr_i \in Abstract$ and $1 \leq i \leq |Abstract|$):

$j(s', hd(gr_i)) < abstractDistance) \Rightarrow setAbstract\ (s', gr_i)$;

$j(s', hd(gr_i)) \geq abstractDistance) \Rightarrow (gr_k = number\ AbstractNode + 1 \wedge hd(gr_k) = s')$.

The above process then repeats with the new group gr_k.

The abstraction moving target search (AMTS) algorithm is shown in Figure 3. The *abstraction move list* guides the agent's movement sequence in the abstraction maze. The *real move list* guides the agent's movement sequence in the original (unabstracted) maze. Variable *detectTarget* denotes if the agent currently detects the target and *exploreLocation* denotes the nearest node location which does not belong to any group. If the agent does not detect the target, it will move according to the last generated *real move list* and complete the moves in this list. If the target has been acquired at the completion of moves in the real move list, *withinRange* is set to false and the run ends. Otherwise, the algorithm is repeated with the next detection of the target by the agent. If the real move list is empty, the algorithm attempts to generate

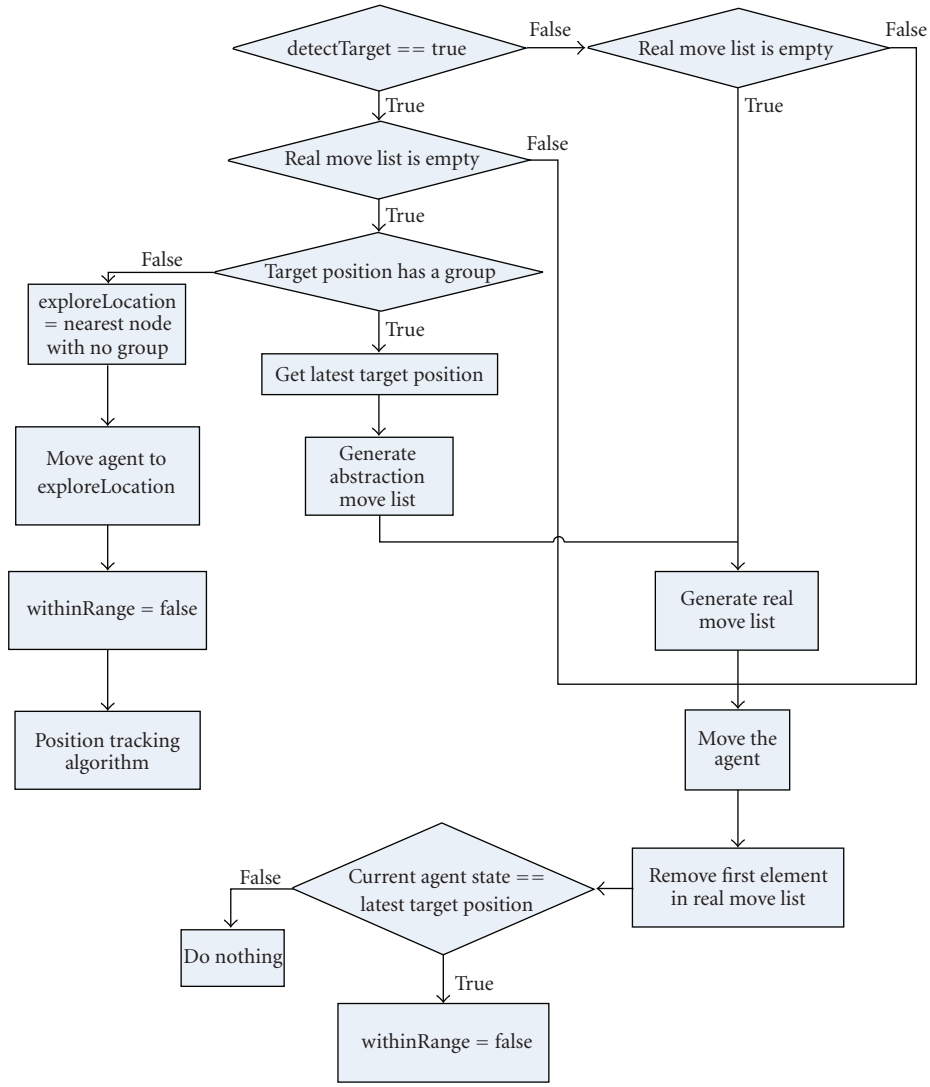


Figure 3: Abstraction MTS algorithm.

one, based on the last known abstraction maze position of the agent. The agent then follows the moves in this generated *real move list*.

However, if the agent detects the target, *withinRange* is set to true and the target position t is checked for association with a group. If it is, the agent will generate an *abstraction move list* (with the latest target position) and *real move list*. Each *real move list* contains a movement path up to the next group head and the last *real move list* with a path up to the target. Both abstraction and real move lists are then used to guide the agent's movement to acquire the target. When agent state is the same as the target position, withinRange is set to false so that the next search-acquisition cycle can begin. If, on the other hand, the target position is not associated with a group, the agent finds the nearest node with no group and continues with exploration from this location using the position tracking algorithm. Formally, this is expressed as follows.

While (*withinRange* = TRUE):

for all $gr_{s'}$, where $gr_{s'} \in succ(gr_s)$, find $f_{\min}(gr_{s'}, gr_t) = j(gr_{s'}, gr_t) + h(gr_{s'}, gr_t)$;

for $f_{\min}(gr_{s'}, gr_t) \Rightarrow \mathrm{AL} = \{gr_s, gr_{s'}, \ldots, gr_t\} \wedge \mathrm{RL} = \{s, s_1, s_2, \ldots hd(gr_{s'})\} \Rightarrow$ repeat generating RL for next group till $s_k = t$, where $s_k \in gr_t$.

Abstraction MTS does not employ heuristic values and, therefore, does not learn. To reduce first move decision latency, each *real move list* is generated after the previous one has been traversed. This process continues until the agent arrives at the target location. When the agent traverses a *real move list*, it will ignore the target's move. It only updates the target's position when it generates a new *real move list*. In this way, it will be faster to search for the target with lower memory requirements.

5. Performance Simulation and Analysis

The simulation was conducted for 6 different maze sizes: 50×50, 100×100, 150×150, 200×200, 250×250, and 300×300. All algorithms had the same starting points for their agent and target in the same maze. The starting points of agent and target are random and the heuristic distance between their starting points is at least half of the diagonal length of the maze. For each maze, every algorithm was executed 100 times and the average of the results was recorded.

5.1. Results. The results show that the degree of learning required in the algorithm depends on the target game application. For turn-based games like chess and weiqi, compute-intensive learning algorithms based on heuristics are effective propositions. However, for real-time action games where expected responses are in seconds or milliseconds, compute-intensive learning approaches significantly degrade playability. For real-time MTS gaming scenarios, in specific, an adaptive algorithm, with minimal or no learning but favouring faster acquisition, proves a more viable solution.

Figure 4 shows an example of a maze used in our experiments. The black portions represent the pathways in the maze. In this particular maze, there are two entry/exit points (top-left and bottom-right). White portions indicate walls in the maze. Each maze is essentially a square $n\times n$ grid.

Since all algorithms incorporate the same position tracking routine, the number of exploration moves is similar for all algorithms. In position tracking routine, the agent only needs to check the value of its neighbour. So, the other four algorithms should have a constant time, independent of maze size. Figure 5 shows the number of agent steps taken by the moving target search algorithm to acquire the target. The number of moves required for the learning process depends on the difference between the heuristic value and the actual value. As this value difference increases, the agent takes more steps to update the heuristic value. Weighted MTS incurs additional moves to find alternate path to the target. Because of this, weighted MTS has the worst performance in a perfect maze. Commitment MTS has the second best performance. It can be explained by the behaviour of the target that moves randomly. Because of that, the target will not move far away from initial position. So, it will be better for the agent to ignore some of target move to reduce learning process.

Each point on the graph represents an average of 100 runs of the AMTS algorithm. Although the agent and target start at the same locations for each run, the target moves randomly. In some of the runs, it is possible that the target approaches the agent more closely leading to an acquisition with less moves. The target's movement can also be influenced to an extent by the maze topology generated. The maze in Figure 4, for example, has a number of linear pathways without many junctions with the result that the target may have a net effect of moving along one dimension without deviation, leading it closer to the agent despite starting further away. On the whole, therefore, the average number of moves may drop even when the maze size increases. This is shown in Figure 5, from 100 to 150 nodes as well as from 250 to 300 nodes. As the problem space increases, however, the overall trend shown by the AMTS algorithm is still an increasing number of moves.

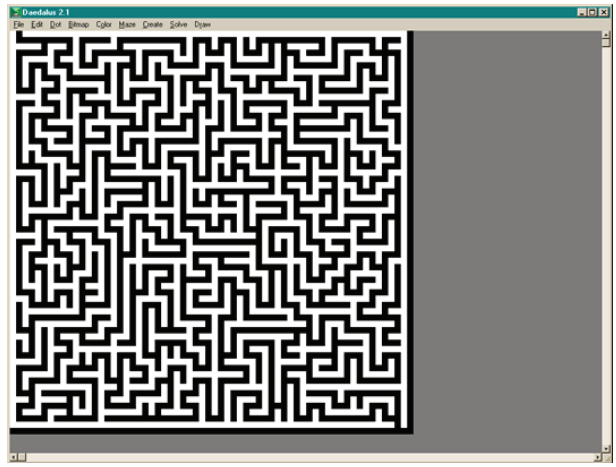


Figure 4: Two-dimensional maze generated by Daedalus program [13].

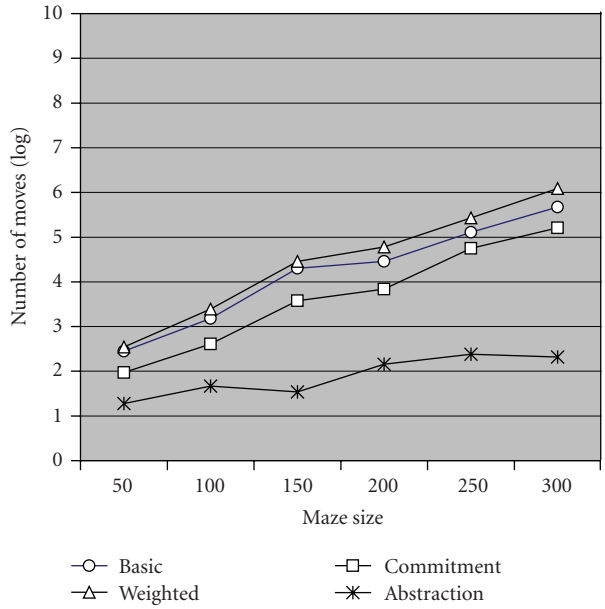


Figure 5: Target acquisition moves comparison.

Figure 6 shows the maximum time required for each movement in the moving target search algorithm. It is known that BMTS, WMTS, and CDMTS have $O(1)$ computation complexity and $O(n^2)$ memory requirements. The upper bound for computation complexity is $O(p^k)$; where p is the actual path length and k is number of branching in a node (constant value) [8]. Hence, the performance of these algorithms scales exponentially with increasing maze size. AMTS, however, has computation complexity that depends on the maze structure and the actual path length. The computation complexity is also greatly reduced by introducing abstraction maze and computing partial path

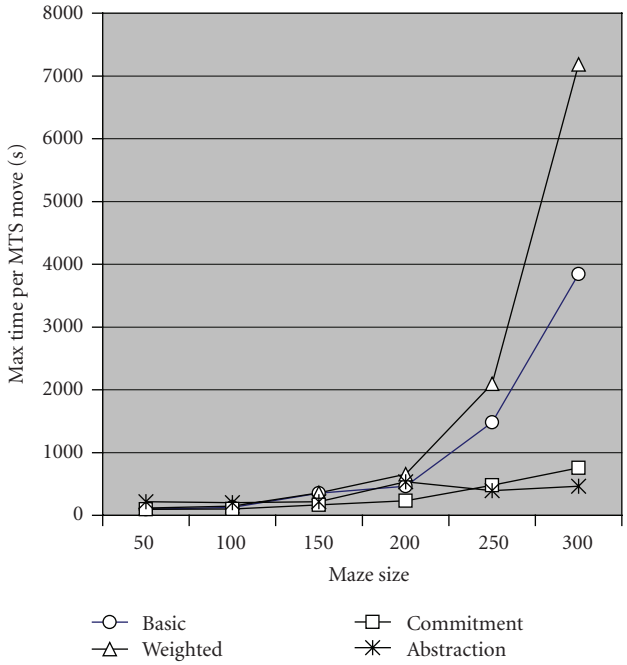


Figure 6: Maximum time per MTS move.

based on neighbouring abstract group information instead of problem space-wide heuristic arrays. This results in the performance of AMTS being linear rather than exponential. As shown in Figure 6, the increase rate for computation complexity in AMTS is on average linear, and the upper bound of memory storage in AMTS is $O(n)$; where n is number of states.

Since the target movement is random, there could be instants in some runs when the target leaves the detection range of the agent while it is executing either "abstracted" or "real" moves. As a result, the agent switches back to exploration with the position tracking algorithm before it can complete the movements that lead to target acquisition. This invariably leads to a higher overhead in decision time per move even though the target starts off closer to the agent in a smaller maze. However, since the target moves slower than the agent, these "irregularities" do not occur often. Figure 6 shows such an irregularity occurring from 200 to 250 nodes before the max time per move increases again.

6. Conclusion

This paper compared and analysed several variants of the MTS algorithm. The main focus of performance, such as effectiveness of learning and speed of response, has been compared with various algorithms. Overall, abstraction MTS has the best performance. It may have the highest exploration move, but it has the lowest MTS move. However, there are some weaknesses of abstraction MTS that we will be studying.

(1) It is difficult to determine *abstractDistance* correctly, especially when the agent does not know the size of maze. *AbstractDistance* is the distance from one node to its group head.

(2) As the maze size increases, it will take a longer time to generate the movement path.

(3) Abstraction MTS may not generate optimal path since it computes complete path in abstraction level, not in actual maze.

Acknowledgment

The authors gratefully thank and acknowledge the critique from the anonymous reviewers that have significantly improved the presentation, organisation, and content of this manuscript.

References

[1] Hellgate Alliance, "The Official Hellgate: London Community Site for South East Asia, offering community, news, information, contests, guides, and more," September 2008, http://hellgate.iahgames.com/.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press and McGraw-Hill, Boston, Mass, USA, 2nd edition, 2001.

[3] A. J. Patel, "Amit's Game Programming," September 2006, http://theory.stanford.edu/~amitp/GameProgramming/.

[4] D. C. Pottinger, "Terrain analysis in realtime strategy games," in *Proceedings of Computer Game Developers Conference (CGDC '00)*, 2000.

[5] S. Koenig, "A comparison of fast search methods for real-time situated agents," in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '04)*, vol. 2, pp. 864–871, IEEE Computer Society, New York, NY, USA, August 2004.

[6] T. Ishida and M. Shimbo, "Improving the learning efficiencies of realtime search," in *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI '96)*, vol. 1, pp. 305–310, Portland, Ore, USA, August 1996.

[7] S. Rabin, *AI Game Programming Wisdom*, Charles River Media, Rockland, Mass, USA, 2002.

[8] T. C. H. John, E. C. Prakash, and N. S. Chaudhari, "Strategic team AI path plans: probabilistic pathfinding," *International Journal of Computer Games Technology*, vol. 2008, Article ID 834616, 6 pages, 2008.

[9] V. Bulitko and G. Lee, "Learning in real-time search: a unifying framework," *Journal of Artificial Intelligence Research*, vol. 25, pp. 119–157, 2006.

[10] T. Ishida and R. E. Korf, "Moving-target search: a real-time search for changing goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 6, pp. 609–619, 1995.

[11] R. E. Korf, "Artificial intelligence search algorithms," in *Handbook of Algorithms and Theory of Computation*, CRC Press, Boca Raton, Fla, USA, 1999.

[12] M. Shimbo and T. Ishida, "Controlling the learning process of real-time heuristic search," *Artificial Intelligence*, vol. 146, no. 1, pp. 1–41, 2003.

[13] Think Labyrinth: Computer Mazes, September 2008, http://www.astrolog.org/labyrnth/daedalus.htm.