International Journal of Engineering Research and General Science Volume 3, Issue 3, May-June, 2015
ISSN 2091-2730

A focused dynamic path finding algorithm to pursue a moving target

Rohit-Renoy, Rahul-Prem, Shubhangi Patil

Student, renoyzac@gmail.com, +919923134799

Abstract—Path finding has its applications in both static and dynamic environments. Various algorithms exist to find a path between a hunter and a prey in a static environment, wherein the prey is stationary. However, the complication arises when the prey is continuously moving around the map. Pursuing a moving target in computer games poses several challenges to agents, including real-time response, large-scale search space, severely limited computation resources, incomplete environmental knowledge, adversarial escaping strategy, and outsmarting the opponent. In this paper we improve upon a novel moving target pursuit algorithm to solve problems involving single hunter and single prey. The algorithm makes use of a queue to store the prey's current trajectory. The hunter then selects the optimal position from this queue and moves to that position in incremental steps. This is repeated till either the hunter catches the prey or the prey escapes. In this paper the previous algorithm is improved by modifying the heuristic function to only calculate heuristics for the part of the map around the hunter. Both the authors previous algorithm and the improved version presented in this paper use RTAA* in its core to plan a local path to the target.

Keywords—AI, Path Finding, RTAA*, Game Search, Moving-target pursuit (MTP), A* search, search optimization

INTRODUCTION

At its core, any path finding method starts by searching a graph by starting at one vertex and exploring its adjacent nodes. This is repeated with the adjacent nodes until the destination node is reached. The path thus found is usually the shortest route to the destination. Two primary problems of pathfinding are to find a path between two nodes in a graph and to find the optimal shortest path. Basic algorithms such as Depth first and Breadth first search address the former problem by exhausting the possibilities; processing starts from the given node, iterating over all possible paths until the destination is reached.

In any path finding algorithm there is a hunter and a prey. These algorithms could be categorized into two parts based on whether the prey is stationary or in motion. However there are two main approaches using which we find the optimal path, they are static and dynamic. In a static algorithm information about the environment is known prior to any activity in the environment. This makes it possible to pre compute values necessary for finding the optimal path even before the hunter and prey are in motion. In a dynamic algorithm however, the environment may change while the computation is in process. An overview of techniques to find the optimal path in such situations is given by Leenen et al[14]. Pursuit tasks occur frequently in many domains. For instance, in computer games human-controlled agents are often pursued by hostile agents. In cooperative settings however, it is required that a computer controlled agent follow the other agents in the world.

Moving target pursuit (MTP) task arises in numerous domains, such as law enforcement, video games. MTP poses multiple challenges for several reasons. Firstly, in real-time scenarios, the time available for planning each move is limited, for example in video games, several game designers impose strict limits to maintain a fluid frame-rate. This frame-rate may differ from one designer to another but usually they're of the order of milliseconds for all path finding units. Secondly, even though a strategy such as 'go to the target's starting position and then repeat all of its moves' may guarantee a capture when the pursuers are faster than the target, we prefer an algorithm where the pursuer outsmarts the target instead of just following and outrunning it.

In this paper we propose a novel algorithm which is built upon a previously established algorithm TAO-MTP [7]. Consider an environment consisting of a map, a hunter and a prey. Certain assumptions about the environment and other entities are made such that, there is an 'escape point' on the map such that if the prey reaches it before the hunter, then we say that the prey has escaped. The positional details about the prey is stored in a queue called as the PreyTrajectory queue, using this queue we calculate the best point for the hunter to intercept the prey. The algorithm works in two phases namely exploration phase and tracking phase. During the exploration phase the hunter searches for the prey using the PreyTrajectory queue, once the position of the prey has been determined then the tracking phase begins. At this point in the algorithm the hunter follow the prey and provided that the hunter's speed is greater than that of the prey then it will eventually catch the prey. This interception point where the hunter intercepts the prey is called as $v_{f\ best}$ and is calculated when the PreyTrajectory queue is full. Under test conditions we found out that after every 4-5 seconds the algorithm calculated $v_{f\ best}$. However, initially when the program starts, the starting position of the prey which is stored in the PreyTrajectory queue is considered as $v_{f\ best}$.

RELATED WORK

Not many techniques are available for pursuing a moving target. However the study on moving target pursuit falls into two broad categories.[1]

(i) The (MTS) moving target search family of algorithms [2]-[6]; these algorithms learn the shortest path distances between pairs of states on a map. Using learning real time A* (LRTA*) the original MTS algorithm was made for a moving target. When tested in turn-based settings where the target periodically skipped moves MTS was shown to be complete. But however, it was subject to "thrashing" and "loss of information" [6] while the target was in motion. The state of the art MTAA* [7], GAA* [8] and fringe-retrieving A* (FRA*) [9] algorithms also belong to this family. MTAA* extended the Adaptive A* algorithm, an incremental heuristic search method, for moving target search and demonstrated experimentally that the resulting MT-Adaptive A* is faster than isolated A* searches. [7]. Adaptive A* being an incremental heuristic search algorithm solves a series of similar search problems faster than A* as it updates the h-values using information obtained from previous searches. Generalized Adaptive A* (GAA*) finds the shortest paths in state spaces where the action costs can increase or decrease over time [8] FRA* is an incremental version of A* that repeatedly finds shortest paths for moving-target search in known grid-worlds. [9]

(ii) A second group of algorithms based on learning a map as compared to a heuristic function such as Trailblazer [9] [10] and Trailblazer with abstraction[11] were designed to build a map of a priori in unknown environment and then used to plan a path towards the target. Another state of the art PR MTS [1] [12] algorithm followed a similar approach. TrailMax and its variant dynamic abstract TrailMax [11] are also such algorithms. The PR MTS algorithm prebuilds a hierarchical state abstraction map, then uses the LRTS *(d,r,T)* learning algorithm [13] in an abstract space to come up with an abstract path of d actions and finally uses A* to refine the abstract path towards the goal.

These previous works however, fail to consider the challenges in modern games as identified in [1]. Firstly, the prey and hunter agents don't take turns in a game environment. Secondly, the maps used in commercial games are not random and tend to exhibit patterns that can be used to the prey's advantage. Thirdly the environments are known to be *a priori.* Fourthly, the prey agents could use sophisticated escaping strategies such as human-controlled agents. Lastly, some methods such as the PR-MTS have an initial pre-computation overhead. And due to the fact that the hunter may use up a large portion of the computational resources available, not much is left for these strategies so that they can be scaled to large maps.

PROBLEMS IN PATH FINDING

One of the most challenging problems in Game Theory is that of pursuing a moving target. Moving target pursuit algorithms consume a lot of computational resources ineffectively in modern computer games. The Artificial Intelligence (AI) engine present in these environments is responsible for delegating hundreds and thousands of hostile agents to plan their pursuit paths simultaneously over large game maps so that they can chase the other escaping weak agents, despite being executed in constrained environmental conditions such as but not limited to real time interactivity, avoiding obstacles,, agent property settings such as the sense scope or move speed, physical dynamics model and adversarial escaping strategy. Consider a small scale example: the Age of Empires 2 game develop by Ensemble-Studios in 1999, in this game almost 60-70% of the game simulation time was spent in finding a path, a major portion of which was consumed for pursuing moving targets. However, moving-target pursuit algorithms can only realize the general tactical action which include actions such as follow-up and pursue. Such algorithms also help in the study pertaining to inverse escaping algorithms and also acts as a basis for implementing game agents with high-level intelligence.

We could thus say that, the underlying problem is a real-time agent-centered tactical search for a moving target over a considerably large map, this is perceived by exploration within the agents' limited sense scope (such as vision scope and hearing scope). Thus it becomes necessary for the hunter to interleave the planning and execution actions with an incomplete knowledge about the environment. The distinctive properties of this problem renders existing static target search algorithms such as Dijkstra, A* and D* inapplicable.

However a significant portion of the computational resources available is consumed as a result of the game designers wanting to maintain a steady and smooth frame rate. These limit are usually to the order of milliseconds for all path finding units, while the players require the agents' responses to them to be made as early as possible. They would prefer the hunter to outsmart the prey rather than simply outrunning it. This approach raises the following questions:

1. How can we minimize the planning time before the move is made and particularly before the first-move delay so that the hunter responds to the players' request in real time?

2. If only the local sensory information pertaining to a large map is given, how can the hunter plan a near to optimal pursuit path and simultaneously minimize the memory and overall learning time required?

3. How can the prey be outsmarted and captured by the hunter, knowing that the prey has a sophisticated escaping strategy.

PROBLEM FORMULATION

The problem of Moving Target Pursuit can be summarized into a single definition as a tuple. The tuple will be $\left(G, A_h, A_p, (v_h^0, u_h^0, d_h^0, s_h^0), \left(v_p^0, u_p^0, d_p^0, s_p^0\right)\right)$ where G is the map in which the simulation is run, A_p and A_h denote the Prey and the Hunter respectively and $(v_h^0, u_h^0, d_h^0, s_h^0)$ denotes the initial parameters of the Hunter, while $\left(v_p^0, u_p^0, d_p^0, s_p^0\right)$ denotes the initial parameters of the Prey. The distance between the Hunter and the Prey is the Manhattan distance between v_h and v_p denoted as $|v_h v_p|$. Thus, the termination condition of the algorithm is when $v_h = v_p$. Equal v_h and v_p indicate that A_h and A_p are present at the same position and A_h has captured A_p. If $\left|v_h v_p\right| > s_h$ where s_h is the sense scope of the Hunter then, this indicates that A_p has moved out of the sense scope of the Hunter meaning that A_p has escaped.

In order to focus on the essence of the problem, we have made the following basic assumptions that have also been made in previous works on this subject.

1) The study is based on maps of commercial games, where each of the maps is made of several cells and each cell can either be occupied by an Agent or a Wall.

2) Agents can move only in four directions, i.e., Up, Down, Left, and Right. If the agent can travel between any two free neighbouring cells and, an edge is added to the Graph. The cost of travelling along that edge will be 1.

3) u_h and u_p are measured by the number of vertices agents have moved per time unit. We assume they are constant and $u_h > u_p$.

4) v_p in any time is reachable for A_h to move to, in order to avoid the meaningless situation where it is impossible for the Hunter to reach the Prey. This is needed for completeness in all real-time heuristic search algorithms.

5) The environment, except for the moving target, is static and deterministic.

6) We consider only admissible heuristic functions which do not overestimate the actual remaining cost to the goal.

ALGORITHM

The TAO-MTP algorithm which is used in this paper is derived from a previous paper [7].

The main idea of TAO-MTP algorithm mostly lies in the tracking strategy: it uses a queue to store the prey's trajectory and simultaneously employs the highly efficient runs real-time Adaptive A*(RTAA*) [8] algorithm to search for any position in the queue. Once it has explored the position, it will stop searching and move to the position, then move along the stored trajectory to pursue the prey. As long as the hunter's moving speed is faster than that of the prey, and its sense scope is large enough, it will eventually capture the prey.

The contributions, in this paper, are mainly made to further improve the performance while calculating the heuristics in the exploration phase.

The working of the TAO-MTP algorithm can be described as follows –

First, the target game map is converted into its equivalent graph representation. Each of the nodes of the graph represent whether the corresponding cell in the map is traversable or not. An edge is added to the graph between any two nodes corresponding to which the cells in the map are traversable.

Second, the prey's Trajectory is stored in a queue which is used to estimate the convergence point between Hunter and prey.

Third, a heuristic function is designed to measure the Hunter's overall pursuit cost, which includes the cost of moving to the position in the stored trajectory and the cost of moving along this trajectory from the position until capturing the prey.

Fourth, RTAA* is repeatedly executed within limited steps to approach the current optimal position, so as to get a suboptimal overall pursuit cost. Meanwhile, the Hunter speculatively moves to any position in the stored Trajectory to speed up the convergence.

Fifth, a logic threshold is set to limit the storage requirements of the prey's stored trajectory. The segment of the trajectory which is of little value is culled, thereby freeing up storage space.

Sixth, the ratio of the cost of not using the tracking strategy to the cost of using it in the process of moving along the stored trajectory is monitored. When the ratio is below a minimum threshold value there is a large curve in the stored trajectory and the advantage of using the tracking strategy disappears in current situation, the stored trajectory will be automatically cleared, and TAO-MTP algorithm is restarted to optimize the pursuit path.

OPTIMIZATION

In order to automatically reduce the storage cost of the escaping trajectory, the queue Q sets a logical threshold, denoted as $|Q|_{max}$, to limit the length of the stored trajectory. The queue Q will automatically delete the segment of the stored trajectory from its head once $|Q| > |Q|_{max}$, because the segment stored earlier is of less usage to store than that most recently stored.

To make further optimizations while calculating the path in the exploration phase of the algorithm, the concept of Heuristic window is introduced. In the previous algorithm while calculating a local path to a point in the trajectory, the heuristics for that point would be calculated at each iteration of the game loop. The previous approach is inefficient especially because the Hunter is executing k-Steps RTAA*, i.e., it is taking k-steps towards a fixed point. Throughout the duration of the exploration phase there is no need to calculate the heuristics outside the window of k-Steps around the current position of the hunter, since the hunter will move, at the maximum k-Steps from its current position.

This window of size k-Steps around the Hunter is known as Heuristic Window. Whenever the heuristics for any path are to be calculated, the heuristics for only those nodes are updated which lie inside the Heuristic Window.

The size of this Heuristic Window is defined by four parameters –

fromX – Starting position of the window along the X axis

fromY – Starting position of the window along the Y axis

toX – Ending position of the window along the X axis

toY – Ending position of the window along the Y axis

The calculation of the Heuristic Window can be summarized by the pseudo-code below:

```
if(X_coordinate of Hunter < 0 )
{
    fromX = 0
}
else
{
    fromX = X coordinate of Hunter - heuristicWindow
}
if(Y coordinate of Hunter < 0 )
{
    fromY = 0
}
else
{
    fromY = Y coordinate of Hunter - heuristicWindow
}
if(X coordinate of Hunter > size of Map along X axis )
{
    toX = size of Map along X axis
```

```
}
else
{
    toX = X coordinate of Hunter + heuristicWindow
}
if(Y coordinate of Hunter > size of Map along Y axis )
{
    toY = size of Map along X axis
}
else
{
    toY = Y coordinate of Hunter + heuristicWindow
}
```

CONCLUSION

Moving Target Pursuit Algorithm has a multitude applications in the field of Game development, Military applications, Automation, AI, etc. The main purpose of this project was to improve upon an existing algorithm to make improvements in it regarding its resource usage. All of the above challenges have been achieved .The resulting algorithm which has been implemented using java, has a significantly reduced memory footprint. The running time of the previous algorithm has also been improved thanks to the upgraded Heuristic function which calculates heuristic values only for the nodes which may be required by the Hunter in the immediate future. The resulting algorithm has been applied to a Hunter-Prey simulation.

REFERENCES:

[1] V. Bulitko and N. Sturtevant, "State abstraction for real-time moving target pursuit:A pilot study," in Proc. Nat. Conf. Artif. Intell.,Workshop Learn. Search, 2006, pp. 72–79.
[2] T. Ishida, "A moving target search algorithm and its performance improvement," J. Jpn. Soc. Artif. Intell., vol. 8, no. 6, pp. 760–768, 1993.
[3] T. Ishida and R. Korf, "Moving target search," in Proc. Int. Joint Conf. Artif. Intell., 1991, pp. 204–210.
[4] T. Ishida and R. Korf, "A real-time search for changing goals," IEEE Trans. Pattern Anal. Mach. Intell., vol. 17, no. 6, pp. 609–619, Jun. 1995.
[5] T. Ishida, "Moving target search with intelligence," in Proc. Nat. Conf. Artif. Intell., 1992, pp. 525–532.
[6] S. Melax, "New approaches to moving target search," in Proc. AAAI Fall Symp. Games Planning Learn., 1993, pp. 30–38.
[7] Mingliang Xu, Zhigeng Pan, Hongxing Lu, Yangdong Ye, Pei Lv, and Abdennour El Rhalibi. "Moving Target Pursuit Algorithm Using Improved Tracking Strategy." IEEE Transactions on Computational Intelligence and AI in Games, VOL. 2, NO. 1, MARCH 2010
[8] S. Koenig, M. Likhachev "Real-Time Adaptive A*" in Proc. Int. Joint Conf. Autonom. Agents Multiagent Syst., 2006, pp. 281–288
[9] F. Chimura and M. Tokoro, "The Trailblazer search: A new method for searching and capturing moving targets," in Proc. Nat. Conf. Artif. Intell., 1994, pp. 1347–1352.
[10] S. Edelkamp, "Updating shortest paths," in Proc. Eur. Conf. Artif. Intell., 1998, pp. 655–659.
[11] T. Sasaki, F. Chimura, and M. Tokoro, "The Trailblazer search with a hierarchical abstract map," in Proc. Int. Joint Conf. Artif. Intell., 1995, pp. 259–265.
[12] V. Bulitko, N. Sturtevant, J. Lu, and T. Yau, "Graph abstraction in realtime heuristic search," J. Artif. Intell. Res., vol. 30, pp. 51–100, 2007.
[13] V. Bulitko and G. Lee, "Learning in real time search:A unifying framework,"J. Artif. Intell. Res., vol. 25, pp. 119–157, 2006.
[14] L. Leenen, A. Terlunen, and W. le Roux, "A Constraint Programming Solution for the Military Unit Path Finding Problem", ser. Mobile Intelligent Autonomous Systems: Recent Advances. Bota Raton, USA: Taylor & Francis Group, 2012