DOI: 10.1007/s42979-022-01302-x

A Strategy-Based Algorithm for Moving Targets in an Environment with Multiple Agents

Azizkhon Afzalov[1] · Ahmad Lotfi[1] · Benjamin Inden[2] · Mehmet Emin Aydin[3]

Abstract
Most studies in the field of search algorithms have only focused on pursuing agents, while comparatively less attention has been paid to target algorithms that employ strategies to evade multiple pursuing agents. In this study, a state-of-the-art target algorithm, TrailMax, has been enhanced and implemented for multiple agent pathfinding problems. The presented algorithm aims to maximise the capture time if possible until timeout. Empirical analysis is performed on grid-based gaming benchmarks, measuring the capture cost, the success of escape and statistically analysing the results. The new algorithm, Multiple Pursuers TrailMax, doubles the escaping time steps until capture when compared with existing target algorithms and increases the target's escaping success by 13% and in some individual cases by 37%.

Keywords Multiple targets · Multi-agent path planning · Path finding · Assignment strategy · Search algorithm

Introduction

There has been extended research on search algorithms for many years. The study and development of such algorithms were based on the basic scenario of a single agent that is tasked with finding a target or goal state on a graph within minimal time. Each search algorithm has its own purpose and need. Even in a simple, static environment, the pathfinding search algorithm faces several challenges. In complex environments, more challenges arise. Various assumptions of this single agent with a single target, the scenario can be relaxed, leading to more difficult problems: there can be several pursuing agents that need to coordinate their search, assigning strategy to the agents before following targets, there can be multiple targets, all of which need to be caught, and targets can move on the graph over time rather than be in a fixed position.

Many suitable algorithms have been proposed for pursuing agents in the domains of video and computer games, robotics, warehouses [1], and military and surveillance applications [2]. Some of these algorithms are for a single agent, such as MTS [3], D* Lite [4] or RTTES [5] and some are multi-agent, for example, FAR [6], WHCA* [7], CBS [8] and MAMT [9]. These algorithms aim to find the shortest path to the target location(s). While the shortest path is important, the run time is essential, too, as considered by real-time heuristic algorithms [10].

Besides a more standard pathfinding search for a single agent pursuing a single target on a static map, the case could be complicated with an increase in the number of agents or dynamic changes in the environment. For example, in the scenarios with moving targets, the target algorithms also play an essential role in developing multi-agent scenarios, but they are less studied. The goal of such algorithms is to evade capture as long as possible.

This article is part of the topical collection "Agents and Artificial Intelligence" guest edited by Jaap van den Herik, Ana Paula Rocha and Luc Steels.

✉ Azizkhon Afzalov
azizkhon.afzalov2016@my.ntu.ac.uk

Ahmad Lotfi
ahmad.lotfi@ntu.ac.uk

Benjamin Inden
benjamin.inden@mis.mpg.de

Mehmet Emin Aydin
mehmet.aydin@uwe.ac.uk

[1] Nottingham Trent University, Clifton Campus, Nottingham NG11 8NS, UK

[2] Max Planck Institute for Mathematics in the Sciences, Inselstraße 22, 04103 Leipzig, Germany

[3] University of the West of England, Coldharbour Ln, Bristol BS16 1QY, UK

Consider a pursuit and evasion game, where players could be human or computer-controlled. Other examples are video games such as Grand Theft Auto and Need For Speed where both sides of players can be controlled by the algorithms or a flight simulation application where computer-controlled targets are needed to catch or shoot [11]. To make the game more interesting, intriguing, and challenging, the targets need to behave intelligently. Therefore, good target algorithms are an essential factor in improving the gaming experience.

Target algorithms that exist usually have strategies such as maximising the escaping distance [12], random movements to selected, unblocked positions in order to evade from the capturer [13] or, in a state-of-the-art approach called TrailMax, maximising the survival time in the environment by considering the potential moves of pursuing agents on each time step [14].

Multi-agent pathfinding (MAPF) problems have been analysed in detail in the literature [15]. These problems are known to be NP-hard [1]. As an example of such a problem in a video game is when all non-player agents need to navigate from a starting location to the goal location on a conflict-free route in a static or dynamic environment [16].

Algorithms developed for moving, in other words escaping targets, can make the empirical study of MAPF problems more meaningful, useful, and challenging. Thus, how can we improve on existing ones? We previously introduced an algorithm [17] based on TrailMax that can be used for multiple moving targets to flee from multiple agents in a dynamic environment. A good design of such an algorithm can help targets to escape more intelligently, rationally and in a human-like manner.

This study considers more testing scenarios against more pursuer strategies, target algorithms, benchmarked maps, player combinations and improving the cost while the target expands pursuers' nodes. Empirical evaluations report different performance metrics, such as capture cost, success rate, computation time and statistical analysis for the significance of the findings.

In the remaining parts of this paper, the following section presents the related work. “Multiple Pursuers TrailMax: Proposed Approach” describes the new approach to the problem. Empirical comparisons are described in the subsequent section, and "Discussion" and "Conclusion" sections follow up.

Related Works

This section introduces several existing target algorithms in the literature. The following is a brief description of each algorithm.

Target Algorithms

Although there is plenty of research in the literature emphasising algorithms for pursuing agents, there are few studies that are conducted on algorithms for mobile targets. The A* algorithm is a classic example that is implemented as an algorithm for many pursuing agents, as well as target algorithms [15].

TrailMax. TrailMax is an intelligent algorithm that is based on a strategy. It generates a path for a target considering the pursuing agent's possible moves, i.e., it efficiently computes possible routes by expanding its current and adjacent neighbouring nodes and agent's nodes simultaneously [14].

The aim of the TrailMax algorithm is to make the targets stay longer by maximising the capture time. The players can move on the map; thus, the target computes an action on every time step with new updated information about the players. It is for one-to-one player scenarios.

The algorithm works as follows. To compute a path, an escape route that maximises its distance away from the agent, it checks the best cost of the neighbouring states against the pursuer's costs and expands nodes accordingly. The algorithm expands nodes that are not yet expanded and not already occupied in the target closed list and not in the pursuer closed list. The node with the best cost is added to the target's closed list, which would generate the path afterwards. The first element in the path is an action for a target to take. This procedure is repeated from scratch every time step.

It is a state-of-the-art target strategy algorithm that performs the best against pursuing agents, aiming to make the targets less catchable or more difficult to be caught [12].

Minimax. When used as the target algorithm, it runs an adversarial search that alternates moves between the pursuers and the target. When the pursuing agent gets closer to the target state, then the target distances itself from the pursuing agent's state. To make the algorithm faster, Minimax is run with alpha–beta pruning search, where alpha (α) and beta (β) are constantly updated to avoid the exploration of suboptimal branches [18]. The used depth is 5, i.e., the outcomes after at most 5 moves of each party are considered.

Dynamic Abstract Minimax. Dynamic Abstract Minimax (DAM) is a target algorithm that finds a relevant state on the map environment and directs the target using Minimax with alpha–beta pruning in an abstract space. There is a hierarchy of abstractions. Higher levels might not provide enough information about the map and lose important details, such as an agent at the close by, and fine abstract levels might be very detailed and increase the computation costs.

The search starts on the highest level of abstraction, an abstract space created from the original space. The minimax algorithm runs a search at the highest level of abstract space and continues to the next low level of abstraction. It stops at the level where the target can avoid the capture. Then, on this level of abstraction, if a path exists, an escape route is computed using the PRA* algorithm (described in next section). If the target cannot escape and there is no available move to avoid the capture on the selected abstract space, then the level of abstraction is decreased, and the whole process repeats until the target can successfully run away from being caught [18]. The used depth is 5.

Simple Flee. Another algorithm for targets is Simple Flee (SF), which can be used to escape from the pursuing agents to the predefined states on the map [19]. The SF algorithm works as follows. At the beginning of the search, the target identifies some random locations on the map. When the target starts moving, it navigates to the furthest location away from the pursuers. To disorient the pursuing agents, such as incremental heuristic algorithms, D* Lite [4] and MT-Adaptive A* [20], that can search from the target's state, the direction towards the selected location changes in every five steps, and if it is the furthest location, it keeps moving. The number of locations on the map and the number of steps before the change are the parameters of the algorithm.

Greedy. This is the standard greedy algorithm that repeatedly makes the best local optimal choices that, in hope, would lead to global solutions. This is a simple and fast approach to solving a problem that uses sub-optimal and easily computed heuristics [21].

Greedy runs a cumulative Manhattan distance of maximising the gap towards the pursuers. It evaluates its options and moves to that state. Once it is at that point, it will stay until being captured, if any other maximum states are not available [19].

Target algorithms, without strategy but considering a pursuing agent's location, make their way to the furthest away state possible. When a target escapes from a pursuer, which, in multi-agent scenarios, sometimes might fall into the path of other pursuing agents. This causes an issue in MAPF frameworks. To avoid this limitation, the study in this paper considers all pursuers, and this new approach provides a winning strategy for the target.

Pursuing Algorithms

This study sets out to develop a new multiple target algorithm. Therefore, this part of the section briefly introduces algorithms for pursuing agents, which will be used in the experiments.

*PRA**. Partial-Refinement A* (PRA*) is an algorithm that reduces the cost of search by generating a path on an abstract level of the search space. These abstracted spaces (graphs) are built from the grid map. The abstract level is selected dynamically. The A* algorithm is then used to run a search with sub-goals on the abstract graph. The abstract path creates a corridor of states in the actual search space, through which the optimal path is found.

This is a widely used approach and its variations have been described with different search techniques [22].

*STMTA**. In cases where more than one target exists, an effective strategy for pursuing agents helps to win the game. Strategy Multiple Target A* (STMTA*) algorithm uses methods to intelligently assign agents to targets to create an opportunity of capturing targets faster [23]. All routes towards the targets are computed and based on the given strategy the optimal combination is selected. Once the strategy is assigned, the pursuing agents know the targets they follow, all agents use the A* algorithm to move towards the targets.

The routes are the distances from the pursuer to the target. Depending on the assignment strategy, the distances between pursuer-target pairs are preferred. For the initial assignment, summation cost or mixed cost criteria are minimised [12]. The summation-cost sums all the distances (n) and mixed-cost takes the longest distance, makespan (m) but in cases of tie break, it uses the sum of distances. The mentioned approach does not focus on re-assigning the agents after their assigned targets have been captured.

Variants of this algorithm using different criteria such as twin-cost, cover-cost, and weighted-cost, were introduced and developed [24]. STMTA* uses these three criteria during the tests because the previous study measured their performance, and overall, they produced better results than the other cost criteria. Throughout the experiments, if any target is caught, the pursuing agent is reassigned to another target depending on the strategy followed.

The twin-cost criterion multiplies the sum of distances n with makespan m, $(n * m)$. In situations, if a tie-breaker is needed, then the average of n and m is taken.

The weighted-cost criterion multiplies these values with a given percentage, totalling to 100% and adds them up. During the experiments, the ratio of 50/50 was used for the weighted-cost criterion, $(n * 0.5) + (m * 0.5)$. The combination with the lowest value is selected for twin-cost and weighted-cost criteria.

The cover-cost criterion uses a different approach. Instead of using the distance cost, it computes the area each pursuer covers. By taking turns, a pursuer and a target mark each available, not occupied state covered P or T respectively. The pursuer does need to reach the target, depending on the players' positions on the map, pursuers and targets intersect in between. Each pursuer's cover is measured and the combination with most Ps is assigned to the pursuers. When a pursuer computes its P, it is possible to overlap among other pursuers. For example, the summations-cost criterion adds all distances per combination and the lowest value among all combinations is selected. In the cover-cost, the P values are summed for each combination and the highest result is preferred.

Multiple Pursuers TrailMax: Proposed Approach

In the following section, a new target algorithm is described. First, the motivation is given for the algorithm, then it follows with pseudo code, see Algorithm 1, and finalises with further improvements.

When the problem was described in the Introduction section, it was stated that a smart target algorithm is very useful to have. In the simple scenarios where a single agent pursues one target, the target would know from which agent it needs to escape, as there is only one. Some of the strategies to run away from the agent have been discussed in the previous sections. But if a situation is considered where multiple targets need to escape from the current state and move to the safest destination in the dynamic environment, how would targets know which pursuing agent they need to avoid for a successful run? For example, SF can flee from the closest pursuer but sometimes could run into other pursuers. What would be a smart move for a target while avoiding capture if there are many pursuers?

Although the TrailMax algorithm, as introduced in the previous section, is a state-of-the-art algorithm, it has been designed to work with only one agent, meaning a target does not have any strategy to escape from one pursuer and avoid another approaching pursuer at the same time.

For this specific reason, a target algorithm that would be able to identify approaching multiple agents and escape from all pursuers, a novel algorithm, called Multiple Pursuers TrailMax (MPTM), is developed.

The MPTM algorithm uses a similar methodology as TrailMax but is enhanced for MAPF problems. There are two possible benefits that could come from extending TrailMax to MAPF problems. First, the target can identify the state location of other targets and collaborate with them. Second, it can ensure the escape not only from one pursuing agent but from any approaching evading agents. Here the focus is on the second issue. It is exhaustive, meaning it considers all possible moves from the agents. Therefore, it is relatively computationally intensive and provides a solution if one exists.

The Algorithm

The pseudo-code for the MPTM algorithm is depicted in Algorithm 1. First, the current locations of all players (pursuers and target) need to be initialised in line 2. The next step is to group all players according to their role and append their positions into the relevant queues, all pursuers to the *pursuer_node_queue* and a target to the *target_node_queue*. At this point, all players will have a cumulative cost of zero, lines from 3 to 5. To make it easier to follow the code, each movement cost will be equal to one, unless it is in wait action, then it is zero. This is with the assumption that there is no octile distance. However, the algorithm works with different speeds and distances.

Algorithm 1: The Multiple Pursuers TrailMax algorithm

```
1:  function MultiplePursuersTrailMax()
2:    initialise position for all players (pursuers and target)
3:    initial cumulative cost c ← 0 for each player
4:    add target to target_node_queue
5:    add pursuers to pursuer_node_queue
6:    target_caught_states ← 0
7:    if target is not captured then
8:      while target_node_queue not empty do
9:        $c_t$ ← get c from target_node_queue
10:       $c_a$ ← get c from pursuer_node_queue
11:       if ($c_t \leq c_a$) then
12:         remove target from target_node_queue
13:         if target not in target_closed and pursuer_closed and parent node not in pursuer_closed then
14:           insert target into target_closed
15:           append target neighbours onto target_node_queue
16:       else
17:         for each $p_i$ of players do
18:           get state $s_i$ for $p_i$
19:           if $s_i$ is pursuer then
20:             $c_a$ ← get c on pursuer_node_queue
21:             remove $p_i$ from pursuer_node_queue
22:             if $p_i$ not already in pursuer_closed then
23:               insert $p_i$ into pursuer_closed
24:               if $p_i$ in target_closed then
25:                 increment target_caught_states
26:                 if target_caught_states is equal to size of target_closed then
27:                   return true
28:               append $p_i$ neighbours onto pursuer_node_queue
29:   generate target_path
30:     if target_closed not empty then
31:       reverse target_closed
32:   return target_path
```

The algorithm has four different lists. The *target_node_queue* and *pursuer_node_queue* contain expanded, visited nodes, such as the current state or neighbouring states for both target and pursuers. The *target_closed* and *pursuer_closed* lists contain states that are already visited and occupied by players.

Since this is the target algorithm, in line 7, it starts first to check if it is already caught or not. Then loops through if there are any target nodes in the *target_node_queue*. As this is the first step, it only contains the target's current position. Then, it computes the cumulative cost c, the highest value, for target c_t and pursuers c_a at lines 9 and 10. If the c_t is lower or equal to the c_a, then the target expands its nodes, line 11.

During the expansion of nodes for targets in lines 12–15, first, the target node is removed from the *target_node_queue*

Table 1 The name of testbeds used from Baldur's Gate for the experiments with their height and width (number of nodes), and traversable states

Map names	Height x Width (number of nodes)	Empty States to Expand
AR0311SR	54×52	558
AR0407SR	54×52	576
AR0507SR	54×52	739
AR0508SR	54×52	567
AR0512SR	54×56	896
AR0527SR	54×52	531
AR0531SR	54×52	716
AR0707SR	59×56	974

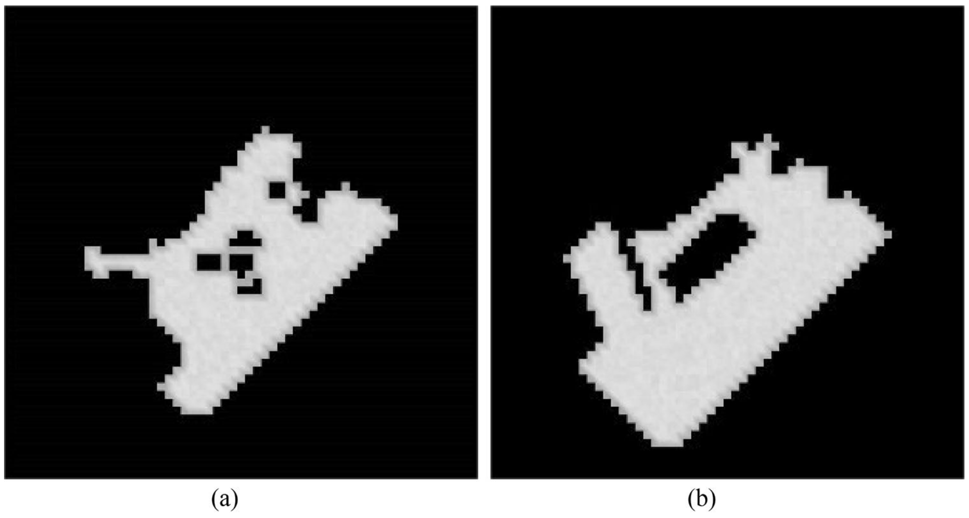


Fig. 1 The experimented sample maps, (**a**) AR0311SR and (**b**) AR0507SR, are used in the Baldur's Gate video game

and placed inside *target_closed* if it is not already in the list and not in the *pursuer_closed* list. It also checks if the target's parent node is not in *pursuer_closed*. The target loops through its available adjacent neighbours and adds them to the *target_node_queue*. These steps are iterated until no state is left to expand. The nodes are expanded like in breadth-first search, first-in-first-out.

When the target c_t is higher than c_a, the condition on line 11, the pursuers take the turn, and they start to expand their nodes. The main part of this algorithm is the lines between 16 and 28, where each pursuer loops through its state and expands its nodes independently from other pursuers. The target needs to know the position of pursuers' states and loops through each player. If it is a pursuing agent, then this

Table 2 The average number of steps (the capture cost) for each target algorithm against pursuer algorithms

Player combinations (Pursuer vs Target)	Target Algorithms	Pursuer Algorithms (number of steps)			
		PRA*	STMTA* twin-cost	STMTA* cover-cost	STMTA* weighted-cost (50/50)
4 vs 2	SF	50.44	51.22	51.94	52.73
	Greedy	53.32	50.44	49.80	50.77
	MMX	73.11	57.38	58.3	57.86
	MPTM	117.30	119.45	125.62	117.92
4 vs 3	SF	52.78	55.98	57.14	55.33
	Greedy	60.78	52.02	51.22	51.63
	MMX	68.23	59.33	60.43	61.08
	MPTM	130.27	138.97	146.26	132.28
5 vs 2	SF	48.31	49.60	50.00	49.53
	Greedy	52.70	49.55	50.20	50.00
	MMX	67.59	56.26	55.79	55.05
	MPTM	106.72	100.77	108.36	104.81
5 vs 3	SF	51.31	52.94	54.33	53.45
	Greedy	58.57	54.04	51.66	54.94
	MMX	63.59	56.36	56.93	56.00
	MPTM	126.92	123.55	133.58	112.23
Mean for all combinations	SF	50.71	52.44	53.35	52.76
	Greedy	56.34	51.51	50.72	51.84
	MMX	68.13	57.33	57.86	57.50
	MPTM	120.30	120.69	128.46	116.81

A larger number is better as it avoids the capture from the pursuing agents particular agent will be removed from *pursuer_node_queue* and placed inside *pursuer_closed* if not already in. The neighbours will be added to the *pursuer_node_queue*. Any state visited by the pursuer exists inside the *target_closed* list, then *target_caught_states* is incremented and compared to the size of *target_closed*, which returns true if equal.

Lines 29–32 generate a path. The last element in *target_closed* is the furthest state that the target could move to. This list is reversed to identify the route, and the first element in the list is the action that the target takes. The function repeats every time step to find the best action for the target.

This turn-based expansion goes to the point where all states on the map have been occupied either by the target or the pursuers. The target could only win if its state is not taken by any pursuers until the timeout.

For multiple targets, the algorithm is run on each target, and normally, each will get a different outcome based on their location. The result will be the same if they are all in the same state. Even if the starting position is different, the targets could join their path if that is the optimal option.

Further Improvements

The strategy of TrailMax works for one-to-one agent scenarios, and to get the *best cost* from the list for each player is straightforward. But this is not the case for the MPTM algorithm as it considers many pursuing agents in one search. The *pursuer_node_queue* contains information for all pursuers and their moving directions with costs.

It has already been discussed that the initial cost is zero for all players. When line 11 is called, it will be true, and the target will take turns to expand and increase its cost by one. On the next iteration, this condition will be false, as the cost for the target is 1, and all pursuers' cost is still zero. The expansion takes place for pursuers. As there are many pursuers, line 20 will request for the first pursuer's cost from the *pursuer_node_queue*. Then this pursuer will expand and increase its cost to 1. There is a problem here because TrailMax requests the *best cost* on each iteration. It would have been fine if there was only one pursuer, but this is an issue with multiple pursuers. If the *best cost* was considered for multiple pursuers, then only the first pursuer would be expanded as only its cost would be incremented. This leads to the fact that only the same pursuer is requested with the *best cost* and all other pursuers are left without expansion with initial cost zero.

To fix the above problem, the cost requested on lines 10 and 20 is not the *best cost* but a cost for each pursuer in order of from the *pursuer*_node_*queue*. This gives greater opportunity for a target to evaluate all pursuers' moves and make decisions more accurately.

Another enhancement is that MPTM does not only consider and run away from the closest pursuing agent but takes into consideration all pursuers on the map by checking each pursuer's state on line 18.

Empirical Evaluations

In this section, the empirical results will be presented to demonstrate the efficiency of the proposed algorithm. First, the experimental setup will be described, then, performance results of the MPTM algorithm described in previous section will be reported.

Experimental Setup

For better comparability, standardised grid-based maps from the commercial game industry are used as a benchmark [25]. The environments used are eight maps from Baldur's Gate

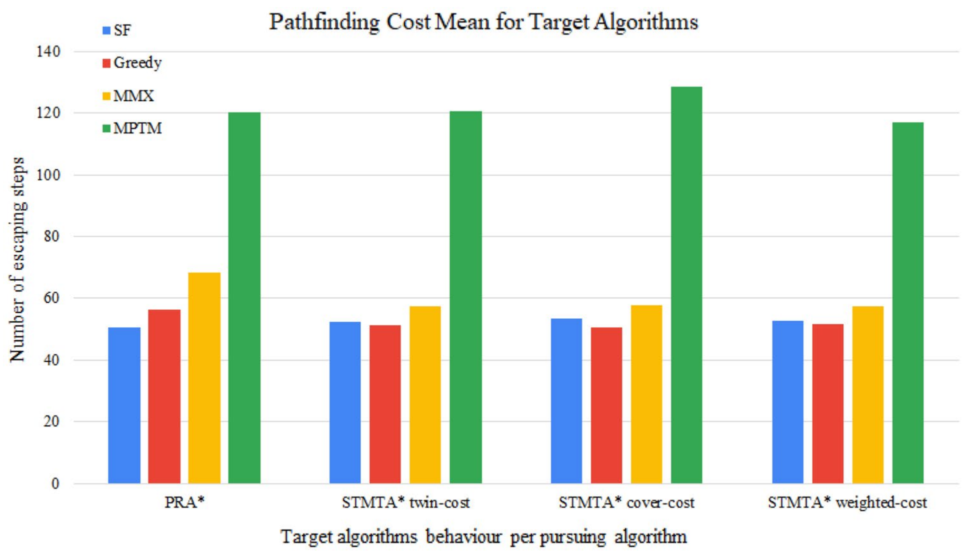


Fig. 2 The overall comparison of the MPTM algorithm with other target algorithms per a pursuing agent algorithm. The graph displays the mean for all maps and all player combinations

Table 3 Wilcoxon Rank Sum test results (p values) for MTPM compared against SF, Greedy and MMX algorithms

Player combinations (Pursuer vs Target)	Target Algorithms	Maps used from Baldur's Gate Video Game (p values)															
		Map AR0311SR		Map AR0407SR		Map AR0507SR		Map AR0508SR		Map AR0512SR		Map AR0527SR		Map AR0531SR		Map AR0707SR	
		Position 1	Position 2	Position 1	Position 2	Position 1	Position 2	Position 1	Position 2	Position 1	Position 2	Position 1	Position 2	Position 1	Position 2	Position 1	Position 2
4 vs 2	SF	0.0005	0.0006	0.0838	0.0386	3.6E-07	0.3907	0.0010	2.3E-07	4.6E-05	0.3284	0.4404	0.0532	0.0021	0.0630	0.0594	0.0009
	Greedy	0.8597	5.3E-08	0.1866	4.0E-05	3.4E-07	0.0600	7.8E-09	3.9E-08	0.6926	0.0366	7.9E-09	0.7811	5.9E-09	0.1084	9.6E-06	5.2E-09
	MMX	0.0395	0.0009	0.6398	5.3E-05	4.6E-07	4.3E-06	3.7E-06	5.2E-06	0.1500	0.0045	2.3E-07	0.5473	1.5E-08	0.0003	0.0075	5.7E-08
4 vs 3	SF	1.6E-05	0.0236	0.6354	0.0121	5.6E-07	0.1853	6.6E-06	1.3E-05	0.4311	0.2133	0.0080	0.3486	0.0065	0.0130	0.0528	0.0060
	Greedy	0.0050	0.0013	9.8E-05	2.4E-06	5.7E-07	0.0363	7.9E-09	3.5E-07	0.1892	0.3248	7.8E-09	0.3353	6.8E-09	0.0028	0.0002	0.0134
	MMX	0.1842	0.0004	0.0110	0.0118	2.8E-07	1.0E-05	2.5E-08	0.0004	0.2319	0.0160	1.5E-08	0.3353	1.3E-08	0.3939	0.0110	3.1E-07
5 vs 2	SF	0.0002	1.2E-05	0.0068	0.0007	4.6E-08	0.7649	1.1E-06	3.5E-08	4.0E-07	0.9455	0.6161	0.0024	0.2908	0.0025	0.3639	5.8E-08
	Greedy	0.0250	4.6E-07	0.0015	0.2622	5.5E-08	0.0872	7.8E-09	1.3E-08	0.9891	0.0858	7.5E-09	0.7806	7.8E-09	0.5570	0.0006	6.3E-05
	MMX	0.4322	0.0477	0.0070	0.01364	1.3E-08	0.0920	5.9E-08	8.8E-07	0.0202	0.0997	1.1E-07	0.0497	6.7E-08	6.2E-07	0.0156	1.0E-04
5 vs 3	SF	1.8E-06	0.0005	0.5956	0.0289	3.0E-07	0.2380	0.0002	9.6E-06	0.0369	0.8374	0.3935	0.4524	0.0066	0.4874	0.1164	1.1E-05
	Greedy	0.0400	0.0001	0.0004	0.0162	9.5E-07	0.0053	7.9E-09	1.6E-05	0.1549	0.1025	7.9E-09	0.0005	5.9E-09	0.0091	2.1E-05	1.1E-06
	MMX	0.0197	6.2E-05	0.0018	0.6448	1.1E-07	0.4472	3.0E-08	0.0773	0.0209	0.1036	7.9E-09	0.0005	8.4E-09	0.0016	0.0008	1.1E-06

video game as shown in Table 1. Within the experiments, these maps are used with a four-connected grid and impassable obstacles. Figure 1 displays sample maps used for the experiments, where black coloured spaces are the obstacles, and the white space is a traversable area. The maps were chosen based on the presence of obstacles and difficulty of navigation. The movement directions could be up, down, left, and right with a cost of one each. That said, the approach should work with different moving costs as well.

The scenarios were chosen to have multiple targets, and for the experiments, initially, two and later three targets were tested. The combination of pursuers versus targets is displayed in Table 2. These scenarios help to understand the behaviour of the MPTM algorithm when targets are outnumbered.

All players are placed at different randomly selected locations on each map. There were two different sets of starting positions. The first set has all pursuers in the same location and all targets in the same location, and targets are positioned at the farthest distance from pursuers. The second set has all players randomly positioned in disperse, in various walls of the map. This helps to measure and analyse the performance of the algorithms.

Each configuration runs 20 times. The implementation [19] kindly provided by Alejandro Isaza was used as a basis but extended such that multiple targets and various agent-target assignment strategies could be used. The results were obtained using a Linux machine on Intel Core i7 with a 2.2 GHz CPU and RAM with 16 GB.

Experimental Results

Performance analysis is conducted with respect to three key indicators: (i) the number of steps taken for each target algorithm before being caught, (ii) its success rate and (iii) computation time. The first two of the measurements are averaged considering all targets, and the time is normalised per step.

During the experiments, each test run finishes when all targets are caught or there is a timeout. If some pursuers already caught their assigned targets, the chase continues as long as there are still uncaught targets. With PRA*, all pursuers continue with the next closest target, and it is possible that all pursuers will chase only one closest target and leave others because of their far distance. Whereas the STMTA* algorithm has an assignment strategy, all targets are being chased, and when one target is caught, the pursuer that becomes idle is reassigned next uncaught target. Success for pursuers is achieved when all targets are caught, and the number of steps until the targets have been caught is recorded. The success for the targets is to avoid the capture or stay on the map as long as possible.

Capture Cost. To evaluate the MPTM algorithm, a comparison with SF, Minimax and Greedy is displayed in Table 2. This measures the performance in terms of the number of steps for all targets. The numbers indicate the mean of steps for target algorithms on all maps.

Table 2 displays results for different target algorithms. Each value is the mean of eight tested maps. The proposed MPTM target algorithm offers a much longer stay on the maps for all configurations. This indicates that it avoids capture and demonstrates smarter decisions. The higher number is better.

Some maps have island-type obstacles that allow the targets to escape from pursuers more easily, see Fig. 1. Although each map has many states to explore, as seen in Table 1, all algorithms managed to find an escape route. SF and Greedy both display similar capture time and their results are close to each other. Minimax is better than SF and Greedy but still not as good as MPTM.

The results compared in Table 2 show that for all player combinations, the MPTM algorithm managed to escape all pursuing agents two times longer than MMX. The same algorithm when compared against SF or Greedy, the results display that on average MPTM manages to run away from the pursuers 2.3 times longer. The graph in Fig. 2 provides a visual comparison of the times to capture between MPTM and the other three target algorithms.

Comparing scenarios with a different pursuing agent and target numbers shows that, as expected, when the pursuer to target ratio increases, capture times tend to decrease, while when the pursuer to target ratio decreases, capture times tend to increase.

The evidence shows that the new MPTM algorithm outperforms SF, Minimax and Greedy algorithms in the number of steps in all test configurations.

While the experiments were designed to study target algorithms, it is also interesting to note that the STMTA* algorithm with its assignment strategy variations performs overall better than PRA*.

Statistical tests are also used on the capture costs to find out which of the results are significantly different. The proposed MPTM algorithm is compared against existing SF, Greedy and MMX algorithms. Only the STMTA* weighted-cost algorithm's results are used for the comparison as it has shown overall the best results among other pursuer algorithms as shown in Table 2. The capture costs are not normally distributed; therefore, the statistical results are obtained using the Wilcoxon Rank Sum tests. A significance level of 0.05 is used. The values obtained from the statistical tests are provided on a map in Table 3.

Table 3 displays *p* values for all eight maps and four different player configurations separately that are used during the experiments. There were two starting positions on each map. Each set of players was aggregated on the first position and on the second position, all players were dispersed. The results in the table display *p* values individually for each starting position.

From this data, it can be seen that the majority of the results display statistically significant differences. p values presented in Table 3, the results below 0.05 indicate significant differences, while there are results that are below 0.01 the level of significance. Although some results are close significant. Most of the aggregated positions show significance, in contrast to dispersed positions.

It is possible to conclude that the results of the experiments for capture cost are significant for 0.01 on most of the tests. The findings should make an important contribution to the field of target search algorithms.

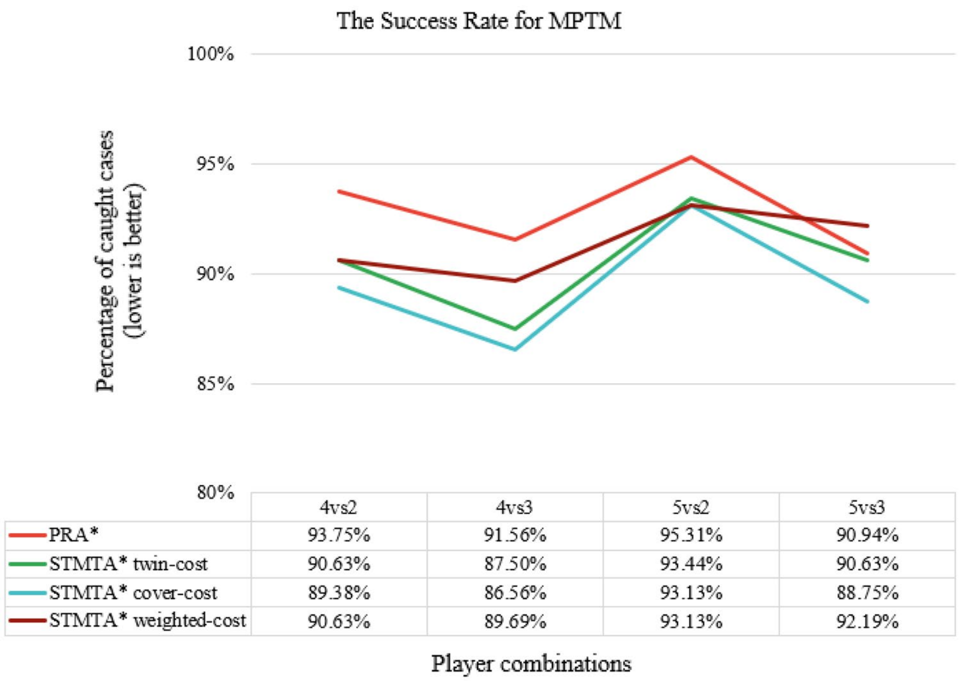
Success Rate. Success for the agents is achieved when a pursuing agent gets to the position of the target. In multi-target scenarios, success is achieved when all targets have been captured. For the target(s), success is the absence of agent success. The success rate for algorithms is shown in Table 4. The results presented in the table are for four target algorithms against four pursuing agent algorithms for all sets of configurations.

From this Table 4, the SF and MMX algorithm performs the worst, and they always get caught by pursuing agents in any tested combination. The Greedy algorithm shows being caught in every possible test against STMTA* algorithm and its variations. It also failed against PRA*, but only in one instance, where it managed to succeed when the deadlock occurred. It happened on the 5vs3 player configuration. In this particular example, when the pursuers caught one target, instead of approaching and catching the remaining targets,

Table 4 The overall success rate of capture for all scenarios. For targets, the lower is better

Target Algorithms	Pursuer Algorithms (success rate)			
	PRA*	STMTA* twin-cost	STMTA* cover-cost	STMTA* weighted-cost
SF	100.00%	100.00%	100.00%	100.00%
Greedy	99.92%	100.00%	100.00%	100.00%
MMX	100.00%	100.00%	100.00%	100.00%
MPTM	92.89%	90.55%	89.46%	91.41%

Fig. 3 The performance rate of success for the MPTM target algorithm for all test configurations and maps. Lower is better



	4vs2	4vs3	5vs2	5vs3
PRA*	93.75%	91.56%	95.31%	90.94%
STMTA* twin-cost	90.63%	87.50%	93.44%	90.63%
STMTA* cover-cost	89.38%	86.56%	93.13%	88.75%
STMTA* weighted-cost	90.63%	89.69%	93.13%	92.19%

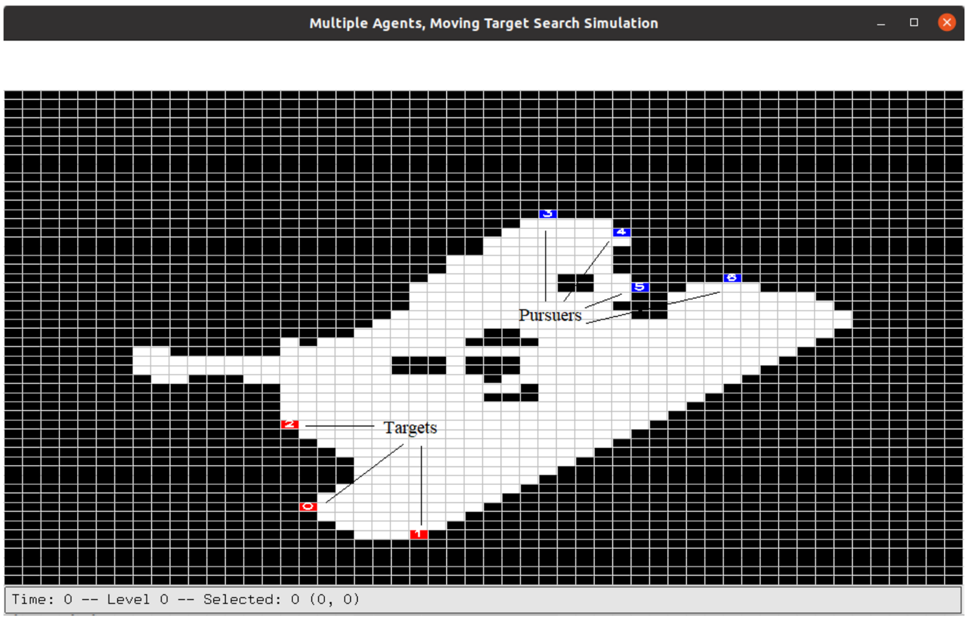
Table 5 The computation time (in seconds) per step for each target algorithm

Player combinations (Pursuer vs Target)	Target Algorithms	Pursuer Algorithms (runtime in seconds)			
		PRA*	STMTA* twin-cost	STMTA* cover-cost	STMTA* weighted-cost (50/50)
4 vs 2	SF	0.00037	0.00038	0.00038	0.00038
	Greedy	0.00019	0.00008	0.00008	0.00009
	MMX	0.00166	0.00155	0.00154	0.00156
	MPTM	0.15913	0.16537	0.16234	0.16284
4 vs 3	SF	0.00057	0.00055	0.00055	0.00055
	Greedy	0.00020	0.00011	0.00011	0.00011
	MMX	0.00141	0.00129	0.00140	0.00138
	MPTM	0.24819	0.24975	0.24949	0.23835
5 vs 2	SF	0.00036	0.00038	0.00038	0.00037
	Greedy	0.00014	0.00008	0.00009	0.00009
	MMX	0.00171	0.00163	0.00163	0.00160
	MPTM	0.15882	0.15846	0.16146	0.15964
5 vs 3	SF	0.00054	0.00054	0.00054	0.00054
	Greedy	0.00019	0.00013	0.00012	0.00012
	MMX	0.00151	0.00136	0.00143	0.00136
	MPTM	0.24287	0.25920	0.24776	0.24295

the pursuers kept moving one step back and forward until timeout.

On the other hand, MPTM shows better results in comparison with SF, Greedy and MMX. Although it has the cases where it eventually gets caught 100% but in overall performance MPTM manages quite well. The graph in Fig. 3 illustrates how MPTM performed for all test configurations on all maps.

Like for capture costs, success rates are also dependent on pursuer and target ratios. The success was proportional to the number of pursuers and targets. More pursuers for the same number of targets increased the captivity. The success



```
Time: 0 -- Level 0 -- Selected: 0 (0, 0)
```

Fig. 4 The Baldur's Gate benchmarked gaming AR0311SR map (Fig. 3a) with pursuers the targets at the initial position

rate was increased when the number of targets incremented versus the same number of agents, as displayed on the graph, see Fig. 3.

The behaviour of the MPTM algorithm is better on the maps that have obstacles that could be navigated around, for example, the maps illustrated in Fig, 1. These types of maps may be suitable for adaptive target algorithms as they offer opportunities for escape but may be difficult for the pursuing agent algorithms if they do not have strategies such as the trap strategy [26]. The maps AR0311SR, AR0527SR and AR0707SR have dead-ends or blind alleys and thus make it more difficult to find an escape route, leading to lower target performance on these maps.

With some algorithms, pursuing agents sometimes fail to catch the targets, although these are outnumbered. They might catch one target but fail to catch the other, or keep following the target, or end in a deadlock until timeout. This is commonly seen in PRA* as there is no assignment strategy before starting the move, unlike STMTA*.

On average, over all maps per player configuration, the success rate can be 13% better than Minimax, Greedy and SF.

Timing. This section measures the time taken for each algorithm during the same tests that measured the capture cost and the success rate. Each experiment is recorded in seconds and averaged over all tests.

Table 5 provides the results for each target algorithm. SF, Greedy and MMX do not do as much computation as MPTM prior to moving, therefore their results are smaller and closer to each other in comparison to MPTM, which has greater differences.

To find the best possible action, the MPTM algorithm computes all possible moves for the target and all pursuers on the map, therefore the computation time is much higher.

Discussion

Results presented in the previous section show that the MPTM algorithm has a greater chance of escaping from multiple pursuing agents, which has been the main focus of this study. The MPTM algorithm can predict the possible future movements of pursuers and therefore MPTM can function smartly by avoiding capture and fleeing as far as possible until it runs out of all options. This could be similar to cop and robber situations, where the robber is a villain and escapes from the cops as illustrated in the simulation gaming map from Baldur's Gate in Fig. 4. The simulation displays the initial position of four cops (pursuers) and three robbers (targets) on the map.

The proposed MPTM algorithm is measured and compared against SF, Greedy and MMX algorithms. MPTM offers better results by staying much longer on the maps and manages to escape the pursuing agents. The number of steps is the capture cost, where in some cases the MPTM avoids capture by 2.6, 2.9 and 2.4 times longer than SF, Greedy and MMX, respectively. Moreover, these results were statistically tested using the Wilcoxon Rank Sum test to establish the significance of the findings. Table 3 displays the p-values and with a 95% level of confidence, most of the results indicate significant differences. Another key measurement is the success rate that exceeds expectations for MPTM with

91.08% of being caught, the lower is better, whereas SF and MMX get caught 100%, and Greedy with 99.98%.

Based on different maps and various player configuration settings, the suggested new algorithm allows functioning efficiently. Despite MPTM's success rate and outsmarting pursuers, further research is needed on improving the computation process. To avoid exhaustive and intensive computation with larger player configurations and to speed up the search, it might be more beneficial to have a branching factor or window-based search.

Conclusion

The aim of this paper was to provide a solution for MAPF problems and develop a target algorithm that would consider multiple pursuers and make a smart escape. Numerous interesting studies have been conducted on search algorithms, and among them are solutions to the MAPF frameworks. Only a few studies have been carried out on target algorithms, especially in multi-target environments.

This research shows that the TrailMax is a successful algorithm for control of targets if developed further for dealing with multiple pursuers. We have proposed amendments to the TrailMax algorithm to make it work as a strategy for multi-agent multi-target search problems in dynamic environments.

The resulting MPTM algorithm has been shown to outperform other target algorithms for the same scenario, and that can make pursuit and evasion scenarios in computer games more challenging, meaningful, and interesting. The results clearly show that the MPTM algorithm performs far better, with at least doubling capture cost and escaping success by 13% on the gaming maps used for benchmarking.

The issue of comparatively high computational costs could be explored in further research, for example, by exploring the use of heuristics that cut off parts of the search space. Although this study focused on evasion from multiple pursuers, further investigation to extend MPTM to collaborate with other targets would be very interesting.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval Not applicable.

References

1. Li J, Gange G, Harabor D, Stuckey PJ, Ma H, Koenig S New techniques for pairwise symmetry breaking in multi-agent path finding. presented at the proceedings international conference on automated planning and scheduling (2020).
2. Panait L, Luke S Cooperative multi-agent learning: the state of the art. 11, 387–434 (2005); https://doi.org/10.1007/s10458-005-2631-2.
3. Ishida T Moving target search with intelligence. Presented at the proceedings tenth national conference on artificial intelligence. pp. 525–532 (1992).
4. Koenig S, Likhachev M: D* lite. presented at the proceedings of the national conference on artificial intelligence (2002).
5. Undeger C, Polat F. RTTES: Real-time search in dynamic environments. Appl Intell. 2007;27:113–29. https://doi.org/10.1007/s10489-006-0023-1.
6. Wang K-HC, Botea A: Fast and memory-efficient multi-agent pathfinding. Presented at the ICAPS 2008 - Proceedings of the 18th international conference on automated planning and scheduling (2008).
7. Silver D Cooperative pathfinding. Presented at the proceedings of the first AAAI conference on artificial intelligence and interactive digital entertainment (AIIDE'05), Marina del Rey, California (2005).
8. Sharon G, Stern R, Felner A, Sturtevant NR. Conflict-based search for optimal multi-agent pathfinding. Artif Intell. 2015;219:40–66. https://doi.org/10.1016/j.artint.2014.11.006.
9. Goldenberg M, Kovarsky A, Wu X, Schaeffer J Multiple agents moving target search. Presented at the IJCAI international joint conference on artificial intelligence (2003).
10. Loh PKK, Prakash EC: Novel moving target search algorithms for computer gaming. 7, 27:1–27:16 (2009); https://doi.org/10.1145/1541895.1541907.
11. Moldenhauer C: Game tree search algorithms for the game of cops and robber, (2009).
12. Xie F, Botea A, Kishimoto A A scalable approach to chasing multiple moving targets with multiple agents. Presented at the proceedings of the 26th international joint conference on artificial intelligence, Melbourne, Australia (2017); https://doi.org/10.24963/ijcai.2017/624.
13. Pellier D, Fiorino H, Métivier M: Planning when goals change: a moving target search approach. Presented at the 12th international conference on advances in practical applications of heterogeneous multi-agent systems: the PAAMS collection (2014); https://doi.org/10.1007/978-3-319-07551-8_20.
14. Moldenhauer C, Sturtevant NR: Evaluating strategies for running from the cops. Presented at the IJCAI international joint conference on artificial intelligence (2009).
15. Sigurdson D, Bulitko V, Yeoh W, Hernández C, Koenig S: Multi-agent pathfinding with real-time heuristic search. Presented at the 14th IEEE conference on computational intelligence and games (CIG) (2018). https://doi.org/10.1109/CIG.2018.8490436.
16. Chouhan SS, Niyogi R. DiMPP: a complete distributed algorithm for multi-agent path planning. J Exp Theor Artif Intell.

2017;29:1129–48. https://doi.org/10.1080/0952813X.2017.1310142.

17. Afzalov A, Lotfi A, Inden B, Aydin ME: Multiple pursuers TrailMax algorithm for dynamic environments. In: ICAART 2021—Proceedings of the 13th international conference on agents and artificial intelligence (2), pp. 437 (2021). https://doi.org/10.5220/0010392404370443.
18. Bulitko V, Sturtevant N State abstraction for real-time moving target pursuit: a pilot study. presented at the proceedings of AAAI workshop on learning for search (2006).
19. Isaza A, Lu J, Bulitko V, Greiner R A cover-based approach to multi-Agent moving target pursuit. Presented at the proceedings of the 4th artificial intelligence and interactive digital entertainment conference, AIIDE 2008 (2008).
20. Koenig S, Likhachev M, Sun X Speeding up moving-target search. Presented at the proceedings of the 6th international joint conference on autonomous agents and multiagent systems, Honolulu, Hawaii (2007); https://doi.org/10.1145/1329125.1329353.
21. Burke EK, Burke EK, Kendall G, Kendall G Search methodologies: introductory tutorials in optimization and decision support techniques. Springer (2014).
22. Sturtevant NR, Sigurdson D, Taylor B, Gibson T Pathfinding and abstraction with dynamic terrain costs. Presented at the proceedings of the 15th AAAI conference on artificial intelligence and interactive digital entertainment, AIIDE 2019 (2019).
23. Afzalov A, Lotfi A, Aydin ME A strategic search algorithm in multi-agent and multiple target environment. pp. 195. Springer (2021); https://doi.org/10.1007/978-981-16-4803-8_21.
24. Afzalov A, He J, Lotfi A, Aydin ME: Multi-agent path planning approach using assignment strategy variations in pursuit of moving targets. In: smart innovation, systems and technologies. Springer (2021); https://doi.org/10.1007/978-981-16-2994-5_38.
25. Stern R, Sturtevant NR, Felner A, Koenig S, Ma H, Walker TT, Li J, Atzmon D, Cohen L, Kumar TKS, Boyarski E, Barták R Multi-Agent Pathfinding: Definitions, variants, and benchmarks. presented at the proceedings of the 12th international symposium on combinatorial search, SoCS 2019 (2019).
26. John TCH, Prakash EC, Chaudhari NS. Strategic team AI path plans: probabilistic pathfinding. Int J Comput Games Technol. 2008;2008:1–6. https://doi.org/10.1155/2008/834616.