Evaluating Strategies for Running from the Cops

Carsten Moldenhauer and **Nathan R. Sturtevant**
Department of Computer Science
University of Alberta
Edmonton, AB, Canada T6G 2E8
moldenha, nathanst@cs.ualberta.ca

Abstract

Moving target search (MTS) or the game of cops and robbers has a broad field of application reaching from law enforcement to computer games. Within the recent years research has focused on computing move policies for one or multiple pursuers (cops). The present work motivates to extend this perspective to both sides, thus developing algorithms for the target (robber). We investigate the game with perfect information for both players and propose two new methods, named TrailMax and Dynamic Abstract Trailmax, to compute move policies for the target. Experiments are conducted by simulating games on 20 maps of the commercial computer game Baldur's Gate and measuring survival time and computational complexity. We test seven algorithms: Cover, Dynamic Abstract Minimax, minimax, hill climbing with distance heuristic, a random beacon algorithm, TrailMax and DA-TrailMax. Analysis shows that our methods outperform all the other algorithms in quality, achieving up to 98% optimality, while meeting modern computer game computation time constraints.

1 Introduction

Moving target search (MTS), or the game of cops and robbers, has many applications reaching from law enforcement to video games. The game was introduced into the artificial intelligence literature by [Ishida and Korf, 1991] as a new variant of the classical search problem. Following this study, the question how to catch a moving prey effectively has been studied extensively [Ishida and Korf, 1995; Koenig *et al.*, 2007; Isaza *et al.*, 2008].

In today's computer games, the players control robbers being chased by computer generated police agents. The same game turned around, i.e. the player controlling a cop and having to chase down a computer generated robber, is far from realizable. This is due to the fact that the focus in MTS research has always been in developing strong pursuit strategies. Very little is known on how to compute strategies for the target. This paper presents a systematic study of move policies for the robber, thus enabling better target modelling and widening the current focus in MTS.

The game of cops and robber has also been studied in the mathematical literature (see [Hahn, 2007] for a survey). Here, cops and robber alternatingly choose their initial positions at the beginning of the game and then play as in MTS. The search time of a graph, i.e. the time needed by optimal playing cops to catch the robber, is therefore a constant. Besides bounds for the one cop and one robber problem, little is known about this graph property. However, a first algorithm that runs in polynomial time and which computes the search time has been developed in [Hahn and MacGillivray, 2006]. Given this algorithm it is possible to determine optimal policies for both players.

Computer games require tight bounds on resource usage, especially computation time. Therefore, computing optimal policies, even though possible in polynomial time with the above algorithm, is not practical. This gives rise to the question of how to quickly compute approximations that yield near-optimal move policies. In the following, we will introduce a new algorithm called *TrailMax* and its variant *Dynamic Abstract TrailMax* to respond to this question.

As optimality has only been studied recently, previous work in MTS has been concerned with approximative solutions and has not, whether for the pursuer or the target, compared methods against optimal policies. Therefore, this paper is the first to conduct a study of various target algorithms with respect to their achieved suboptimality.

A precise definition of the cops and robber game considered in this work will be given in Section 2. We review existing algorithms, including Cover and Dynamic Abstract Minimax, and outline their strengths and weaknesses in Section 3. The new methods, TrailMax and Dynamic Abstract TrailMax, are introduced in Section 4. Evaluations of experiments and extensive comparisons of various target algorithms when playing against an optimal cop can be found in Section 5. We wrap up with conclusions in Section 6.

2 Game Definition

The game of cops and robber is played with n cops and one robber. Cops and robber occupy vertices in a finite undirected connected graph G and are allowed to move to an adjacent vertex or remain on their current location in each turn. Turns are taken alternatingly beginning with the first to last cop followed by the robber. The game is played with perfect information, i.e. the graph and all locations of all agents are

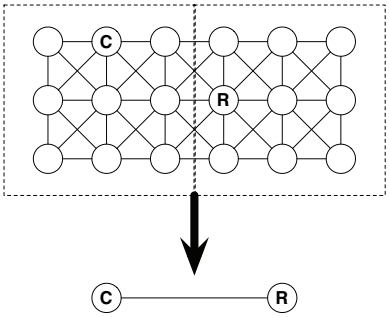

Figure 1: Map abstraction for DAM.

known. G is called n-cop-win if n cops have a winning strategy on G.

Since our focus is on the target and the cop is potentially played by a human player, we concentrate on the one cop one robber problem here. However, all the following methods can easily be extended to multiple cops. Furthermore, we are interested in playing on typical video game maps that include obstacles. Hence, one cop cannot catch a robber that plays optimal when both agents play with same speed. To enable execution of experiments, i.e. many simulations of the game, we have to decide between one of the three ways to guarantee termination: the target moves suboptimally from time to time, the game is ended after a certain number of steps, or the cop is faster than the target. The first possibility contradicts our wish to compute near-optimal policies for the robber. The second choice is problematic due to the choice of timeout conditions. Furthermore, it does not measure the full amount of suboptimality generated by a given strategy because the game is truncated after the timer runs out. Moreover, it is easy to construct an algorithm that achieves optimal results in this game: detect all cycles around obstacles of length greater or equal to four in the map, run to a cycle where the cop cannot capture the robber before reaching the cycle, and exploit the cycle. Therefore, we allow the cop to be faster than the robber. For simplicity we allow the cop to make d subsequent moves when the robber only gets one, i.e. to move to any location within a radius of d of his current position.

3 Related Work

There are only two advanced methods in the literature that try to compute move policies for the robber quickly. [Bulitko and Sturtevant, 2006] suggest using *Dynamic Abstract Minimax* (DAM). This algorithm assumes that various resolutions of abstract maps are available, where an abstract map is created by taking sets of states in an original map and merging them together to form a more abstract map. DAM chooses a level of abstraction to begin with, and then computes a minimax solution to a fixed depth. If the robber cannot avoid capture at that level of abstraction, computation proceeds to the next lower level. We illustrate this in Figure 1. In the abstract map two sets of 9 states have been abstracted together to form a 2-node graph. The cop can catch the robber in one move in the abstract graph, so DAM will search again on the lower level of abstraction. Assume there are ℓ levels of abstraction and the cop and the robber occupy distinct nodes up until level m. The original algorithm begins planning at level m. Running the experiments in Section 5 for multiple fractions of m showed that starting at level $m/2$ is superior. We report the experiments for the later case.

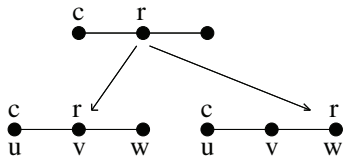

Figure 2: Example of where the original tie breaking of the cover heuristic computation can cause the robber to remain in v instead of going to w.

If the robber can escape, an abstract goal destination is selected and projected onto the actual map. PRA* [Sturtevant and Buro, 2005] is used to compute a path to that node which is subsequently followed for one step. Since only the goal destination is projected onto the ground level, DAM can make mistakes when cycles exist in the strategy. For example, consider a cycle with five nodes and an adjacent cop and robber. The solution is to run around this cycle, but after seven steps the robber will reach the initial position of the cop. Hence, when computing with depth seven, the robber will run towards the cop. The solution is to make DAM only refine one abstract step. However, running the experiments in Section 5 for such a variant showed that the original algorithm, despite its flaws, achieves slightly better results.

Within the present work, we use the same idea of using abstractions for speedup. Our algorithm uses the same policy $(m/2)$ for selecting the first level of abstraction, solves the problem on this level and proceeds to the next lower level if the robber cannot survive long enough due to the computed solution. Otherwise, the abstract solution path is refined into a ground level path.

The *Cover* heuristic, as a state-of-the-art algorithm for moving target search, has been used for both the cop and the robber [Isaza *et al.*, 2008]. This algorithm computes the number of nodes in the graph that the respective agent can get to before any other agent. It then tries to maximize this area with each move, minimizing the area the opponent can reach.

The original algorithm breaks ties by assigning the nodes on the border between two covered areas to the cop. This causes the heuristic to be inaccurate even for simple problems. They used a notion of risk to increase the pursuer's aggressiveness and circumvent this inaccuracy for the cop. As an example for the robber, consider the graph in Figure 2. There are three vertices, u, v, and w. The cop starts on u, the robber on v, and it is the robber's turn. When the robber remains on v, v and w are considered robber cover. If he moves to w, u and v are cop cover (due to the tie-breaking rule) and only w is robber cover. Thus, when maximizing, the robber prefers to stay in v, which is suboptimal. In this work, we modify Cover to eliminate this problem. Vertices are only declared robber cover if he is guaranteed to reach them no matter what the cop does.

But, we found that no matter how the Cover heuristic is

defined, it is easy to construct a simple example where hill climbing would fail for either of the two players. Using notions of ties and untouchable nodes can solve some of the issues but subsequently turns the heuristic into a search algorithm instead of a static heuristic. Thus, we seek to develop a more principled search method instead of trying to patch cover.

When being used for the pursuer, Cover with Risk and Abstraction (CRA) [Isaza *et al.*, 2008] makes use of abstractions to decrease computation time and to scale to large maps. This has not been used for robber. Since the heuristic is most accurate with full information, using abstractions only trades accuracy against speed. Within our experiments, the Cover heuristic without abstractions already performed poorly in terms of survival time against an optimal cop. Therefore, we did not extend the algorithm to incorporate abstractions.

Optimal move policies for both cops and robbers are studied by [Moldenhauer and Sturtevant, 2009]. They developed algorithms that solve one problem instance, i.e. compute optimal policies for a given initial position. Unfortunately, although well optimized, optimal algorithms do not scale to very large maps and cannot meet tight computation time constraints of modern computer games. An algorithm that solves a map, i.e. computes a strategy for cop and robber for every possible initial position was first proposed by [Hahn and MacGillivray, 2006]. We use an improved version that has been used as a baseline in [Moldenhauer and Sturtevant, 2009] to compute optimal solutions offline and to generate a cop that moves optimally within our experiments.

4 TrailMax

We now outline our approach to computing near-optimal move policies for the robber. We will first motivate the algorithm and then provide more details. For ease of understanding the following ideas will be developed for the game where cop and robber move with same speed. However, all the definitions and theorems are extendible to different speed games.

The robber makes the assumption that the cop knows where he is going to move, i.e. that the cop will play a best response against him. Under this assumption, the robber tries to maximize the time to capture. This can also be interpreted as "running away", i.e. taking the path that the cop takes longest to intersect. We will now formalize this idea. Let $N[v] = \{w|(v,w) \in E(G)\} \cup \{v\}$ denote the closed neighborhood of $v \in G$. Let

$$P(v) = \{p : \mathbb{N} \to V | p(0) = v, \forall i \geq 0 : \ p(i+1) \in N[p(i)]\}$$

be the set of paths starting in v. Given a path p_r and p_c for the robber and cop, respectively, that they will follow disregarding the opponent's actions, we can compute the sum of the numbers of turns both agents take until capture occurs:

$$\begin{aligned} T(p_r, p_c) \quad = \min(\quad & \{2t | t \geq 0, p_c(t) = p_r(t)\} \\ & \cup \{2t-1 | t \geq 1, p_c(t) = p_r(t-1)\}). \end{aligned}$$

Definition 1 (TrailMax) *Let $v_r \in G$ and $v_c \in G$ be the positions of robber and cop in G. We define*

$$\text{TrailMax}(v_r, v_c) = \max_{p_r \in P(v_r)} \min_{p_c \in P(v_c)} T(p_r, p_c). \quad (1)$$

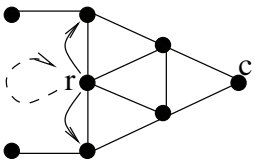

Figure 3: Smallest 1-cop-win graph where the set of moves according to TrailMax (solid) diverges from the set of optimal moves (dashed).

We say G is an octile map if its vertices are positions in a two dimensional grid and each vertex is connected to its up to eight neighbors via the two horizontals, two verticals and four diagonals. Within our experiments we use octile maps to model the environment.

Recall that a graph G is called n-cop-win if n cops have a winning strategy on G for any initial position of the cops and the robber and when all agents move with same speed.

Theorem 1 *Let G be a 1-cop-win octile map. Let v_r and v_c be the initial positions of robber and cop. Then TrailMax(v_r, v_c) returns the optimal value of the game where the cop and robber move at same speed.*

This theorem also holds when the cop is faster as described in Section 2. However, this requires obvious adjustments of the above definitions and is therefore omitted for readability. Unfortunately, the theorem does not hold for general 1-cop-win or n-cop-win graphs ($n \geq 2$).

TrailMax can be used to generate move policies for the robber. For simplicity, the resulting algorithm will be refered to by the same name. Furthermore, a pair (p_r, p_c) for which (1) is maximal will be called a *TrailMax pair*. The algorithm computes a TrailMax pair (p_r, p_c) and then follows the robber's path p_r for k steps ($k \geq 1$) disregarding the cop's actions. Afterwards, TrailMax is called again and a new path p_r is computed, hence our notation TrailMax(k). Unfortunately, the immediate assumption, that TrailMax(1) might yield an optimal strategy for general n-cop-win graphs is not true. Depicted in Figure 3 is an example of a 1-cop-win graph where the robber is to move and the optimal move is to remain on his current position, marked with a r. This causes the cop to commit to a direction, after which the robber can run away more effectively. However, according to TrailMax, the robber has to move to either of the indicated adjacent positions.

A TrailMax pair is efficiently computed by simultaneously expanding vertices around the robber's and cop's position in a Dijkstra like fashion. Two priority queues are maintained, one for the cop and one for the robber. All nodes of a given cost for the robber are expanded first, because the robber moves immediately after computing a policy. Node expansions for the robber are checked against the cop's expanded nodes to test whether the cop could have already reached that point and captured the robber. If this is the case, the node is discarded. Otherwise, the vertex is declared as robber cover and expanded normally. When taking a node from the queue for the cop, it is always expanded normally.

A visualization is depicted in Figure 4. The grey area indicates the vertices that are declared robber cover but are not

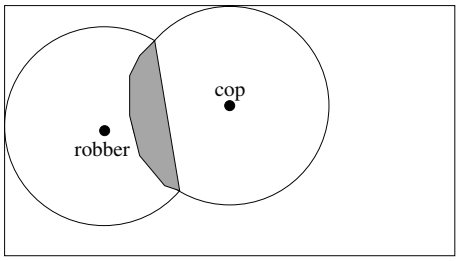

Figure 4: Visualization of TrailMax's computation. The grey area is the nodes that have been reached by the robber first, declared as robber cover but will not be expanded anymore since they were captured by the cop in a previous turn.

Figure 5: One of the maps used in Baldur's Gate that the experiments were conducted on. The black parts are obstacles, white is traversable.

expanded anymore since the expansion around the cop's position captured them in a previous turn. Computation ends when all nodes declared as robber cover have been expanded by the cop as well. The last node that is explored by the cop is the goal node the robber will run to. Path generation can be easily done by maintaining pointers to parents when expanding nodes.

The above computation finds one goal vertex and a shortest path to it. The path then has to be extended by moves that make the robber remain on the goal vertex until capture. It is not hard to show that this extended shortest path is indeed a solution to (1). Note that there might be many possible goal vertices the robber could run to and many different paths to get to them that fulfill (1). Finding all such vertices is possible by remembering all robber nodes that have not been caught before the last cop's turn expansion. This could potentially be used to take advantage of a suboptimal cop, although we do not study this issue here.

Within computer game maps, edge costs are often approximated to enable faster computation. Under the assumption that path costs can only differ by a fixed number of values, i.e. buckets can be used within the priority queue and queue access takes constant time, the above algorithm runs in time linear in the size of the graph. Although TrailMax already scales well to large maps (cf. Section 5) our goal is to make computation time as independent of the size of the input graph as possible. Inspired by DAM we use abstraction to achieve this goal. Starting at an intermediate level of abstraction of the hierarchy relative to the cop and robber positions, TrailMax is computed. If the solution length does not exceed a certain value q (computed by (1)), then computation proceeds to the next lower level. If it does, the computed abstract path is refined to a ground level path using PRA*'s refinement, i.e. progressively computing a path on the next lower level that only goes through nodes whose parents are either on or adjacent to the abstract path. In the following, this algorithm is called *Dynamic Abstract TrailMax* with threshold q and number of steps the solution is followed k, hence DATrailMax(q, k).

5 Experiments

To evaluate our algorithms we compare to the algorithms described in Section 3 and measure the quality and required computation time in terms of node expansions. We set $d = 2$, i.e. the cop can take two turns before the robber gets one and can thus move to any location within a radius of 2 around his current location. First experiments show that greater cop speeds yield the same trends. In contrast, since capture occurs faster, the game becomes easier and less interesting for the robber.

To generate meaningful statistics we use 20 maps from the commercial game Baldur's Gate as a testbed. The smallest of these maps has 2638, the largest 22,216 vertices. A plot of a sample map can be found in Figure 5. Furthermore, 1000 initial positions for each map are generated randomly. We choose the selection at random because we want to explore the performances of the algorithms for all scenarios since in a video game, both agents could potentially be spawned anywhere in the map.

We choose octile connections for the map representation and subsequent levels of abstraction are generated using Clique Abstraction [Sturtevant and Buro, 2005]. To enable effective transposition table lookups in minimax and DAM we set all edge costs to one in all levels of abstraction. Thus, the distance heuristic between two positions (on an abstraction or ground level) becomes the maximum norm of these positions. Furthermore, equidistant edge costs mean we are optimizing the number of turns both players take rather than the distance they travel. All the tested algorithms can be used for nonequidistant edge costs, only minimax's and DAM's performance is expected to be lower.

Using an improved version of the algorithm in [Hahn and MacGillivray, 2006] the entire joint state space is solved first, i.e. we compute the values of an optimal game for each tuple of positions of the robber and cop. This is done in an offline computation and is used to generate optimal move policies for the cop as well as to know the optimal value of the game. Generation of these offline solutions took up to 2.5 hours per map.

We study the following target algorithms:
Cover. The target performs hill climbing due to the Cover heuristic (cf. Section 3). The heuristic has to be computed in every step and for every possible move.
Greedy. The target performs hill climbing using the distance heuristic. This is extremly fast since distance evaluation is very simple.
Minimax. The target runs minimax with α-β pruning, trans-

algorithm	optim.	nE/c	nT/c	nE/t	nT/t
Cover	61.9%	4.687	156.831		
RBeacons(1)	64.3%	0.158	1.065		
RBeacons(5)	65.9%	0.159	1.070	0.037	0.248
RBeacons(10)	67.4%	0.160	1.075	0.022	0.148
RBeacons(15)	68.6%	0.161	1.083	0.017	0.117
RBeacons(20)	69.5%	0.162	1.091	0.015	0.102
Greedy	76.0%	0.0002	0.001		
Minimax(5)	78.7%	0.031	0.216		
Minimax(7)	79.2%	0.146	1.027		
Minimax(9)	79.8%	0.499	3.546		
Minimax(11)	80.3%	1.354	9.709		
DAM(5)	88.8%	0.039	0.238		
DAM(7)	88.4%	0.123	0.752		
DAM(9)	87.8%	0.323	1.985		
DAM(11)	87.1%	0.729	4.476		
TrailMax(1)	98.3%	0.502	16.682		
TrailMax(5)	98.0%	0.520	17.301	0.108	3.598
TrailMax(10)	97.7%	0.543	18.060	0.059	1.970
TrailMax(15)	97.5%	0.565	18.769	0.043	1.433
TrailMax(20)	97.3%	0.585	19.436	0.035	1.169
DATrailMax(1)	97.0%	0.101	2.283		
DATrailMax(5)	97.1%	0.104	2.342	0.023	0.515
DATrailMax(10)	97.0%	0.110	2.487	0.014	0.311
DATrailMax(15)	96.8%	0.107	2.395	0.011	0.251
DATrailMax(20)	96.7%	0.106	2.359	0.010	0.225

Table 1: Experimental results.

position tables and distance heuristic as evaluation function. We experimented with depths from 1 to 11.

DAM. The target runs dynamic abstract minimax with α-β pruning, transposition tables and distance heuristic as evaluation function (cf. Section 3). We experimented with depths from 1 to 11. The depth is used for computation on every level of abstraction.

RandomBeacons(1-20). The target randomly distributes 40 beacons on the map. It then selects the beacon that is heuristically furthest away from the cop's position and computes a path to this location. The path is followed k steps before computing a new path, hence RandomBeacons(k). We tested RandomBeacons(k) for $k = 1, \ldots, 20$.

TrailMax(1-20). We tested TrailMax(k) for $k = 1, \ldots, 20$.

DATrailMax(10,1-20). We tested DATrailMax(10,k) for $k = 1, \ldots, 20$. $q = 10$ was chosen by hand. The question whether there is a better setting remains for future investigation.

To evaluate performance the game is simulated for each initial position on each map. Within these simulations, the target algorithm is called whenever a new move has to be generated. TrailMax, DATrailMax and RandomBeacons are only called when a new path has to be computed, thus the number of turns and algorithm calls differ in this case. All other algorithms are called once per turn and therefore these two numbers are equal. In fact, it is not possible for TrailMax, DATrailMax and RandomBeacons to spread their computation among the turns where the previous computed path is followed because the future position of the cop is unknown. Nonetheless, when used in computer games, these algorithms will only require computation once every k steps and therefore make the frames during path execution available to other tasks. Therefore, we can also analyze the computation time per turn for these three methods.

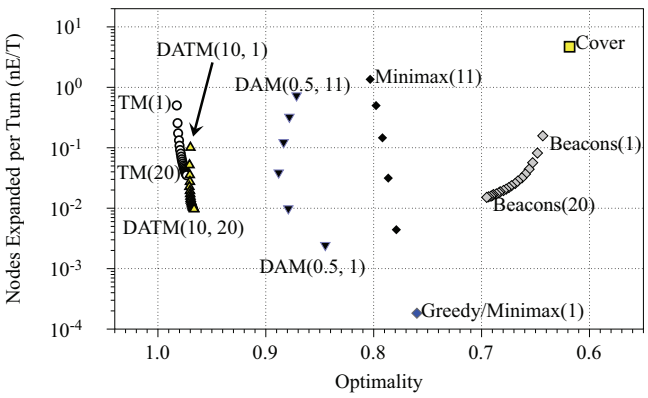

Figure 6: Optimality versus node expansions per turn in one game simulation. Averaged over the number of games played in the experiments. Left bottom corner is best, right upper corner is worst.

We are interested in the following performance measures:

- the expected survival time of the target measured in percentage of the optimal survival time (suboptimality),
- the number of node expansions per call to the algorithm within one game simulation (nE/c) and
- for TrailMax, DATrailMax and RandomBeacons the amortized number of nodes expanded per turn within one game simulation (nE/t).

Similar measures are presented for nodes touched per call (nT/c) and per turn (nT/t). To account for variable sized maps, the numbers of nodes expanded and touched are further normalized and measured as a percentage of the map size. Nodes expanded counts how many times the neighbors of a node were generated, while nodes touched measures how many times a node is visited in memory.

The results are in Table 1 and are plotted in Figure 6. The x-axis is reversed so the best algorithms are near the origin, with high optimality and few expansions per move. Notice further the logarithmic scale on the number of node expansions. A pareto-optimal boundary is formed by Greedy and the TrailMax algorithms, meaning that all other algorithms have both worse optimality and more node expansions per move, on average.

Cover clearly performs the worst. Having to compute the heuristic in every step and for every possible move, its computation time is beyond any computer game requirement. Although solutions on abstractions can be computed in less time, Cover is also the worst algorithm with respect to optimality and optimality decreases when using abstract solutions.

Considering quality, RandomBeacons is the second worst algorithm. This is due to the fact that it does not play very well in the endgame, i.e. when the target is cornered and is about to be captured. When distributing the beacons, many of them lie in parts of the map that are heuristically far away from the cop. Thus, the robber runs towards these positions. Since he is cornered, this results in running into the cop.

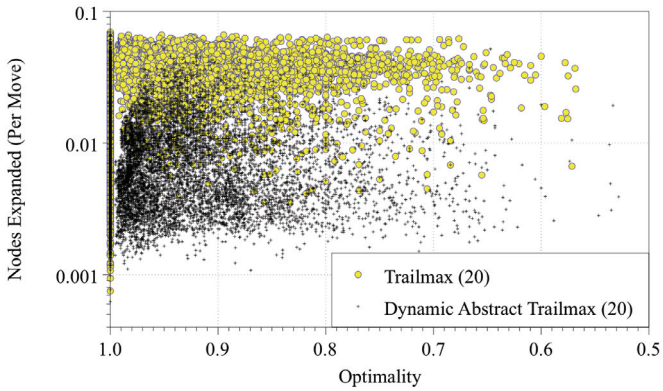

Figure 7: Optimality versus node expansions per move in one game simulation. Plotted for all games played in the experiments. Left bottom corner is best, right upper corner is worst.

As expected, minimax becomes more optimal when the depth is increased. However, its computation time increases exponentially. When using a depth of seven and greater it already expands more nodes per call then DATrailMax.

Abstract levels have cycles in them and minimax can find how to exploit such cycles even with shallow searches. Hence, DAM's computed strategies on abstract levels are similar for different search depths. Therefore, DAM does not significantly increase in optimality when its depth parameter is increased.

It is surprising that Greedy, i.e. hill climbing with a distance heuristic, performs extremely well. Due to the fact that this algorithm requires almost no computation time, we can conclude that Greedy is the method of choice when optimality is of minor importance.

TrailMax and DATrailMax perform best with respect to optimality. Although DATrailMax uses TrailMax on abstract levels it experiences only a small reduction in optimality. On the contrary computation time decreases drastically. (About $5\times$ fewer node expansions per call.) Notice that, although the computation time per call is fairly high, the amortized time per move is small and even comparable to RandomBeacons. When conducting experiments on relatively small maps we found that DATrailMax expands and touches a higher percentage of nodes. This is because the abstraction is not as useful and therefore the algorithm degenerates into TrailMax.

While Figure 6 shows the averaged points of all game simulations, the actual results are clouds of points where each point represents the performance in one game. We compare this underlying data for the two best algorithms, TrailMax and DATrailMax, in Figure 7. The small dark points contain data for DATrailMax, while the larger, light circles are the data points for TrailMax. The x-axis is reversed and the y-axis is logarithmic. DATrailMax is clearly faster. Trailmax has a slight advantage in the number of times it makes optimal moves, resulting in slightly better optimality. Notice that although there are games where both algorithms perform poorly with respect to optimality, the majority are above 90%. Furthermore, node expansions for both algorithms are uniformly bounded at around 7% of the size of the map.

6 Conclusions

Despite research throughout the last two decades, the focus in moving target search has been on computing move policies for the pursuers. In the past, very little was known about how to compute strategies for the target. Due to computer game requirements on computation time optimal algorithms are no feasible approach. Therefore, fast approximations of near-optimal behavior for the target are needed.

The present work conducts a study on such approximations and evaluates their suboptimality. We find that our new algorithms, TrailMax and Dynamic Abstract TrailMax provide the best performance, with near-optimal policies. Surprisingly, we discover that, in our testbed, a greedy strategy is better than most of the previous algorithms. Thus, the present work redefines the state-of-the-art in perfect information MTS.

Future work will address how computation time can be further reduced. The performance of the greedy algorithm, which is the fastest approach, suggests that a greedy algorithm with a better heuristic may perform well. Finally, although we have focused on strategies for the robbers, similar methodology can also be used to evaluate strategies for the cops, and a variant of TrailMax could be used to compute policies for the cops as well.

Acknowledgments

This research was supported by Canada's NSERC, Alberta's iCORE and the German Academic Exchange Service.

References

[Bulitko and Sturtevant, 2006] V. Bulitko and N. Sturtevant. State abstraction for real-time moving target pursuit: A pilot study. *AAAI Workshop on Learning For Search*, 2006.

[Hahn and MacGillivray, 2006] Geňa Hahn and Gary MacGillivray. A note on k-cop, l-robber games on graphs. *Discrete Mathematics*, 306(19-20):2492–2497, 2006.

[Hahn, 2007] G. Hahn. Cops, robbers and graphs. *Tatra Mountains Mathematical Publications*, 36(2):163–176, 2007.

[Isaza *et al.*, 2008] A. Isaza, J. Lu, V. Bulitko, and R. Greiner. A cover-based approach to multi-agent moving target pursuit. *AIIDE*, 2008.

[Ishida and Korf, 1991] T. Ishida and R. E. Korf. Moving target search. *IJCAI*, pages 204–210, 1991.

[Ishida and Korf, 1995] T. Ishida and R. E. Korf. Moving-target search: A real-time search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):609–619, 1995.

[Koenig *et al.*, 2007] S. Koenig, M. Likhachev, and X. Sun. Speeding up moving-target search. *AAMAS*, 2007.

[Moldenhauer and Sturtevant, 2009] C. Moldenhauer and N. Sturtevant. Optimal solutions for moving target search (extended abstract). *AAMAS*, 2009.

[Sturtevant and Buro, 2005] N. Sturtevant and M. Buro. Partial pathfinding using map abstraction and refinement. *AAAI*, pages 1392–1397, 2005.