# Short Papers

## Fast Algorithm for Catching a Prey Quickly in Known and Partially Known Game Maps

Jorge A. Baier, Adi Botea, Daniel Harabor, and Carlos Hernández

***Abstract*—In moving target search, the objective is to guide a hunter agent to catch a moving prey. Even though in game applications maps are always available at developing time, current approaches to moving target search do not exploit preprocessing to improve search performance. In this paper, we propose MtsCopa, an algorithm that exploits precomputed information in the form of compressed path databases (CPDs), and that is able to guide a hunter agent in both known and partially known terrain. CPDs have previously been used in standard, fixed-target pathfinding but had not been used in the context of moving target search. We evaluated MtsCopa over standard game maps. Our speed results are orders of magnitude better than current state of the art. The time per individual move is improved, which is important in real-time search scenarios, where the time available to make a move is limited. Compared to state of the art, the number of hunter moves is often better and otherwise comparable, since CPDs provide optimal moves along shortest paths. Compared to previous successful methods, such as I-ARA*, our method is simple to understand and implement. In addition, we prove MtsCopa always guides the agent to catch the prey when possible.**

***Index Terms*—Incremental search, moving target search, navigation, path finding, predator prey games.**

### I. Introduction

THE objective of *moving target search* (MTS) [9] is to decide the moves of a *hunter* agent, to allow it to capture a moving agent called the *target* or the *prey*. MTS algorithms can be classified into offline and online methods [19]. While offline algorithms (e.g., [6], [15]) determine an optimal move policy for the hunter considering all possible moving strategies of the target, online algorithms react to the actual moves of the target, searching repeatedly. Complete-path online algorithms (e.g., [18], [19]) compute a complete path between the hunter and the target which is then followed by the hunter until the target is caught or moves off the path. Reactive on-line algorithms (e.g., [9]), on the other hand, carry out a bounded search to compute a prefix of a path towards the target. They can function under strict time deadlines, but, as they tend to make myopic decisions, the hunter may perform poorly.

We consider the problem of online MTS in partially known environments in which a subset of the obstacles in the map are known by the hunter prior to search, while remaining obstacles become known only when they are within the hunter's visual range. When the hunter observes an obstacle, it updates its search graph to reflect obstacles in the actual environment. The additional case when obstacles may appear and disappear dynamically is beyond the focus of this work.

MTS in partially known environments has direct application to video games in which it is necessary to program the movements of an automated character that should chase other characters. Frequently in these applications the map is designed by the game developers but the shape of the terrain may be changed during the game by the players. Since a significant part of the map is known in advance, it seems sensible to design algorithms that exploit such knowledge during the online search required for catching the prey. However, current approaches to MTS do not exploit any knowledge about the map in order to speed up search.

In this paper, we propose Moving Target Search with Compressed Paths (MtsCopa), an online MTS algorithm that exploits precomputed information about the map, in the form of a compressed path database (CPD) [1]. A CPD for a map is a database that allows to quickly retrieve, for any pair of states $(s, t)$, the first move of a shortest path between $s$ and $t$ on such a map. CPDs are built in a preprocessing step, and are reasonably compact [1].

At each iteration, MtsCopa uses the CPD to determine the next move to catch the prey. Since the terrain may change during the game the CPD for a certain map does not always contain legal paths between any two positions. Therefore, sometimes MtsCopa requires to run a search algorithm to determine its next move. Since MtsCopa exploits the CPD, such a search can finish much earlier than a standard A* search. The latter has to continue all the way until the target position is selected for expansion. Instead, MtsCopa runs an A* search which stops as soon as a state which is connected to the target via a path on the CPD is about to be expanded.

Under reasonable assumptions, we prove that MtsCopa is complete. However, similarly to the case of other online MTS

Manuscript received November 13, 2013; revised March 13, 2014; accepted June 11, 2014. Date of publication July 10, 2014; date of current version June 12, 2015. The work of J. A. Baier was supported in part by Fondecyt under Grant 11110321.

J. A. Baier is with the Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Santiago, Chile (e-mail: jabaier@gmail.com).

A. Botea is with IBM Research, Dublin, Ireland (e-mail: adibotea@ie.ibm.com).

D. Harabor is with NICTA and also with the Australian National University, Canberra, ACT 3003, Australia (e-mail: dharabor@gmail.com).

C. Hernández is with the Departamento de Ingeniería Informática, Universidad Católica de la Santísima Concepción, Concepción, Chile (e-mail: chernan@ucsc.cl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2014.2337889

methods, optimality (in terms of the number of hunter moves) cannot be guaranteed. Since the target constantly moves, there is no guarantee that using shortest-path moves for the hunter minimizes the total number of moves. In practice, nonetheless, both our results and previous results (e.g., [19]) show that taking optimal moves towards the current prey position leads to fewer hunter moves.

MtsCopa is very simple to implement as it just requires an interface to the CPD, and the implementation of a simple A* search algorithm. We evaluate MtsCopa on standard pathfinding game maps and compare it to the state-of-the-art I-ARA* algorithm [19]. We show that MtsCopa outperforms I-ARA* usually by two or three orders of magnitude in terms of runtime during a chase. In a more fine-grained evaluation we investigate the performance of both algorithms when a time limit to compute the hunter's move is given. As the limit is decreased, the performance of I-ARA* degrades, exceeding the time limit per episode more frequently and returning worse solutions. MtsCopa's performance is much more robust to setting a time limit per move. Furthermore, MtsCopa requires fewer iterations to catch the target.

To achieve its online speed performance, MtsCopa requires offline preprocessing and memory to store a CPD. These requirements depend on the size and topology of a map, as illustrated in our experiments. On the other hand, existing state-or-the-art methods, such as I-ARA* need no preprocessing, and have a much lighter memory footprint.

The rest of the paper is organized as follows. We first briefly review related work. We continue by describing our approach, and presenting a formal analysis. An experimental evaluation comes next, followed by a summary and future work ideas.

## II. Previous Work

The version of MtsCopa that we present in this paper is a significant extension of MtsCopa for known environments [3]. The idea of searching only until finding a state which is believed to be connected to the goal via a shortest-path has also been proposed in the context real-time search (e.g., [16]) and incremental heuristic search (e.g., [8]). However, to our knowledge, MtsCopa is the first algorithm to incorporate this idea in the context of moving-target search.

Other recent work has also addressed the problem of moving-target search in various settings. MT-Adaptive A* [11] is an adaptation of Adaptive A* [10] to the problem of moving-target search in unknown environments. The experiments by [11] report MT-Adaptive A* is 1.23 times faster than repeated A*. I-ARA* [10] is a moving-target search algorithm for known environments, which in the evaluation reported by [10] outperforms repeated A* by a factor of 36.81 on average. Finally Moving-Target D*-Lite [18] is an algorithm for moving-target search in dynamic terrain. In dynamic terrain, the agent may encounter previously unknown obstacles but also some blocked cells may become unblocked before the prey is caught. In the evaluation of [18], Moving-Target D*-Lite was shown to be 6.94 times faster on average than repeated A*.

In summary, a variety of algorithms for moving-target search exist. Some of them are applicable to more general settings (e.g., dynamic terrain) than the one we focus on this paper. In addition, from the data currently published, one can conclude that the algorithm that seems to outperform repeated A* by a larger margin is I-ARA*.

Besides CPDs, in fixed-target real-time search, the literature shows other recent database-driven contributions, such as precomputing paths between selected pairs of locations, and caching information that can be used to reconstruct these quickly [4], [12], [13]. A key idea is identifying so called reachability regions on the map, that can successfully be traversed with hill-climbing. In contrast, CPDs perform a full precomputation of all-pairs shortest paths data, which is then compressed [1].

## III. Approach

We assume both the hunter and the target move over an environment defined by an undirected graph $G = (V, E)$, where each of the arcs in $E$ correspond to the legal moves that any of the agents can perform. Initially, the agents have partial knowledge of the obstacles in the map; specifically, we assume the agents know the position of a subset of the obstacles in the map. As such, the hunter and the prey believe that their search graph is defined by $G_0 = (V, E_0)$, such that $E = \{(u, v) \in E_0 \mid u, v \notin M\}$, for some subset $M$ of $V$. In videogame applications, one can think of $G_0$ as the map of the environment of the deployed game, over which new obstacles, determined by the game conditions, may be added.

A generic MTS framework for a partially known environment is illustrated in Algorithm 1. At each iteration, the hunter and the prey will take one action each. Both the hunter and the prey have a general knowledge of the map before they start moving but there are obstacles that are not known to them in advance. As such, both agents observe their local environment before moving, removing the arcs in the search graphs $G_h$ (the hunter's search graph) and $G_t$ (the prey's search graph) that lead to newly observed obstacles. To guarantee that the prey can be caught by the hunter we assume that the former moves more slowly; this can be implemented by ensuring that, after a number of iterations, the target has performed fewer moves than the hunter. This is a standard assumption in the MTS literature.

**Algorithm 1:** Generic framework for moving target search.

```
   Input: node s_0; node t_0; graph G_0
1  pos_h ← s_0        /*init. hunter's position */
2  pos_t ← t_0        /* init. prey's position */
3  G_h ← G_0          /* hunter's search graph */
4  G_t ← G_0          /* prey's search graph */
5  while prey is not captured do
6     Hunter and prey observe environment, updating G_h
      (hunter) and G_t (prey)
7     pos_h ← getHunterNextPosition(G_h, pos_h, pos_t)
      pos_t ← getPreyNextPosition(G_t, pos_t, pos_h).
```

Algorithm 2 shows how MtsCopa implements the getHunterNextPosition procedure. The static variable $path$—which in the first call is initialized as empty—stores a previously computed path to a (previous) position of the target. Such a path is followed unless the next position is an obstacle or its size becomes smaller than a parameter $k$. When a new path is needed, the path is recomputed using the function ComputePath. The parameter $k$ allows to control the search effort performed by MtsCopa. As such, if $k$ is set to a very large value, MtsCopa

will re-compute a path to the target in every iteration of the main algorithm. On the other hand, if $k$ is set to 0, MtsCopa computes a new path only when the current one is blocked or empty; in other words, it sticks to a previously computed path unless such a path is not viable. For values between 0 and infinity, MtsCopa follows a previously computed path until it is $k$ steps away from a previous position of the target.

In a fully known, static graph, using a CPD would be sufficient to provide hunter moves. In our problem, however, potential discrepancies between $G_h$ and $G_0$ may render CPD data, computed for the original graph $G_0$, not fully accurate in the graph $G_h$. This is why function ComputePath uses a combination of CPD data and A* search [7] in the graph $G_h$. Our A* code implements the open list as a priority queue. The goal condition of A*, i.e., the function that determines when the search should stop, becomes true for a state $s$ in $G_h$ when the path from $s$ to $pos_t$ in $G_0$ (which can be retrieved quickly using the CPD) is also a path in the hunter's search graph $G_h$. In other words, the search stops when A* finds a state which the hunter believes is connected to the target using the moves given by the CPD. A* needs, in addition, a heuristic function $h$ which is used to guide search. MtsCopa uses the length of the path from $s$ to $pos_t$ in $G_0$ (which can be retrieved quickly using the CPD) as the admissible heuristic value for $s$.

**Algorithm 2:** MtsCopa for partially known terrain.

```
1 if first call then
2     path ← empty
3 if path.size() ≤ k or path.top() is an obstacle then
4     path ← ComputePath(G_h, pos_h, pos_t)
5 return path.top()
```

Note that both to determine whether a state $s$ is a goal and to compute the heuristic value of a state $s$ it is necessary to iterate through the path that connects $s$ with the position of the target on $G_h$. Depending on how much time resources are available, performing such a search for every single hunter step may be too expensive to afford. That is, then, the main motivation for introducing the $k$ parameter described earlier, which can be used to limit the amount of search carried out by MtsCopa.

Finally, note that MtsCopa can be improved significantly if the search graph is fully known prior to search; i.e., in the case $G_0$ corresponds to the actual terrain. In such a case, Algorithm 3 should be used.[1]

**Algorithm 3:** MtsCopa for completely known terrain.

```
1 move ← GetMoveCPD(pos_t, pos_h)
2 return move
```

## IV. Theoretical Analysis

When obstacles newly discovered by the hunter divide the map, separating the hunter from the prey, the algorithm correctly reports that the problem has no solution. This remark allows us to focus, with no loss of generality, on the case when a map remains connected at all times. For simplicity, the analysis presented in this section assumes that all edges have a cost of 1. We prove that, when the prey stays put once in a while, MtsCopa is guaranteed to lead the hunter to a successful catch.

Let $d_{G_h}(x, y)$ be the length (i.e., number of steps) of a shortest path from $x$ to $y$ on graph $G_h$. Furthermore, let $pos_h^n$ and $pos_t^n$ denote, respectively, the position of the hunter and the prey at the start of iteration $n$ of Algorithm 1's main loop, right before the agents' positions are updated (i.e., at Line 6). Let $G_h^n$, on the other hand, denote the graph $G_h$ right after the execution of Line 6, with $n$ a valid iteration of Algorithm 1. To simplify our notation, we write $d(pos_h^n, pos_t^n)$ instead of $d_{G_h^n}(pos_h^n, pos_t^n)$, for all $n$. Below we say that a path is computed at iteration $n$ if the invocation of getHunterNextPosition() at iteration $n$ of Algorithm 1 results in the execution of Line 4 of Algorithm 2.

*Lemma 1:* Let $n$ and $m$ be such that $n < m$, such that a path is computed at iteration $n$, and such that for every $j \in \{n+1, n+2, \ldots, m\}$, no path is computed at iteration $j$. Then $d(pos_h^n, pos_t^n) \geq d(pos_h^m, pos_t^m) + \tau$, where $\tau$ is the number of times the prey stayed put between iterations $n$ and $m$.

*Proof:* Let $\sigma$ denote the path computed at iteration $n$. Note that because $\sigma$ was returned by A*, $|\sigma| = d(pos_h^n, pos_t^n)$. Furthermore let $\sigma_t$ and $\sigma_h$ be the sequence of moves (represented as graph edges) taken by the target and the hunter, respectively, between iteration $n$ and iteration $m$. $\sigma_h$ is a prefix of $\sigma$, since the hunter has been following the path $\sigma$; thus $\sigma = \sigma_h \sigma'$. Observe furthermore that $|\sigma_h| = |\sigma_t| + \tau$, where $\tau$ denotes the number of times the prey stayed put between iterations $n$ and $m$. At iteration $m$, $\sigma' \sigma_t$ is a path between the hunter and the prey. Observe that

$$\begin{aligned} |\sigma' \sigma_t| &= |\sigma'| + |\sigma_t| \\ &= |\sigma'| + |\sigma_h| - \tau \\ &= |\sigma| - \tau \\ &= d(pos_h^n, pos_t^n) - \tau. \end{aligned}$$

Since a path with cost $d(pos_h^n, pos_t^n) - \tau$ exists between $pos_h^m$ and $pos_t^m$, then $d(pos_h^n, pos_t^n) - \tau \geq d(pos_h^m, pos_t^m)$, which finishes the proof. ■

Now observe that if at iteration $n$ a path is computed it is possible that $d(pos_h^{n-1}, pos_t^{n-1}) < d(pos_h^n, pos_t^n)$. This can only happen because a new obstacle was discovered and indeed the graph $G_h$ has changed, as established by the following result.

*Lemma 2:* Let $n$ and $m$ be such that $n < m$, such that a path is computed at both iteration $n$ and iteration $m$. Furthermore, assume $G_h^m = G_h^n$. Then $d(pos_h^n, pos_t^n) \geq d(pos_h^m, pos_t^m) + \tau$, where $\tau$ is the number of times the prey stayed put between iterations $n$ and $m$.

*Proof:* By Lemma 1 we know that the distance between the hunter and the prey does not increase as we follow a previously computed path. Now, let $j$ be an iteration at which a path was not computed and $j+1$ an iteration at which a path is computed. Then because the graph $G_h$ does not change between iterations, the path that exists at iteration $j$ between $pos_h^j$ and $pos_t^j$ can be converted into a path from $pos_h^{j+1}$ to $pos_t^{j+1}$ of the same size or less, if the prey stayed put. Thus the distance between the hunter and the prey may not increase between iteration $j$ and $j+1$. ■

*Theorem 1:* Let Algorithm 1 be such that the prey stays put every $n$th iteration. Furthermore, assume there exists a path be-

[1]This is actually the algorithm presented in [3].

TABLE I
MAP DATA (NAME, HEIGHT, WIDTH, NUMBER OF NODES) AND PREPROCESSING STATISTICS (SIZE OF CPD AND TIME TO BUILD CPD ASSUMING ONE CPU CORE IN USE)

| Map | HxW | Nodes | CPD size (MB) | Total preproc. minutes |
|---|---|---|---|---|
| Domain Warcraft III (WC3) | | | | |
| darkforest | 161x161 | 10,691 | 1.8 | 0.8 |
| divide-conq-1 | 169x169 | 19,440 | 3.6 | 2.8 |
| thecrucible | 130x130 | 10,280 | 2.0 | 0.7 |
| Domain Dragon Age: Origins (DAO) | | | | |
| orz100d | 395x412 | 99,626 | 38.0 | 107.2 |
| orz900d | 656x1491 | 96,603 | 15.0 | 90.9 |
| Domain Baldur's Gate II (BG) | | | | |
| AR0603SR | 512x512 | 57,372 | 9.2 | 29.9 |
| AR0300SR | 320x320 | 26,950 | 5.5 | 5.6 |
| AR0500SR | 320x320 | 29,160 | 5.0 | 6.7 |
| AR0700SR | 320x320 | 51,586 | 18.0 | 25.8 |

tween $pos_h$ and $pos_t$ in the search graph $G$. Then, independently of the value of $k$, MtsCopa guides the hunter to eventually catch the prey.

*Proof:* Assume the hunter does not catch the prey and that Algorithm 1 iterates forever. Since the number of obstacles is finite, at some point in time $G_h$ cannot change anymore. From this point we use Lemma 2 to conclude the distance between the hunter and the prey will decrease by one each time it stays put.[2] The hunter eventually catches the prey, which contradicts our initial assumption, proving the theorem. ■

We note that simpler, known strategies, such as navigating to a previous location of the prey, and then following the prey steps, are also complete. However, ignoring the possibility of taking shortcuts towards the current location of the prey has a corresponding price in terms of solution quality.

## V. EXPERIMENTAL SETUP

We compared MtsCopa with the recent, state-of-the-art MTS solver I-ARA* [19]. In MtsCopa, the CPD construction and querying uses the source code of Copa [2]. Copa and I-ARA* were obtained from their respective authors.

In our experiments we used 9 game maps taken from Nathan Sturtevant's online repository[3][17]. Following the approach of Sun *et al.* [19], all maps are considered to be 4-connected.

Table I provides details on the size of each map and includes additional statistics from our preprocessing phase. The CPD size is reported in MBs, and the total preprocessing time (i.e., time to build a CPD) is measured in minutes. Note that the actual wall-clock preprocessing time is substantially smaller, as preprocessing was performed with independent threads on a machine with 6 physical cores. It is a 3.47-GHz machine, running Red Hat Enterprise. Apart from preprocessing, all experiments were run on a serial, single-core setting, on a 2.00-GHz machine running Ubuntu Linux.

Each map is associated with a scenario file that contains a list of problems modeled as randomly selected start-target pairs. In every case the target is reachable from the start. The number of instances in each scenario is 1000 start-target pairs. To execute a problem we place the hunter at the start location and the prey at the target location. We evaluate an MtsCopa hunter and an I-ARA* hunter. The movement strategy for the prey is as in previous work (e.g., [19]): it moves along an optimal path towards a destination picked at random.

After reaching the destination, a new destination is selected and the process repeats. Every 10th move the prey stays put, to ensure that the hunter eventually catches the prey [19]. To choose a destination we use a fixed random seed, to make sure that the moves of the prey can be reproduced exactly in all runs of the experiment. In particular, this is useful to ensure that the prey behaves identically when both MtsCopa and I-ARA* are tested as hunters.

An important feature of I-ARA* is that it is able to receive a *time limit* parameter, used to restrict the time spent to compute one hunter move (i.e., time per episode). Each time I-ARA* searches for a new path towards the target it does so by running a sequence of search rounds. The first of those rounds returns an $\epsilon$-optimal solution, where $\epsilon$ is a parameter. The remaining rounds return increasingly better solutions. If after finishing a search round the time limit has passed, then the search episode is concluded. As such, different runtimes, and solutions of different quality may be obtained by varying the parameter. In addition, respecting the limit is not always guaranteed. In our experiments, we evaluate I-ARA*'s performance for different time limits.

The set of 9 maps was partitioned into 3 groups, according to the 3 domains outlined in Table I. We compared the performance of MtsCopa with I-ARA* since, as implied by the analysis in Section II, it represents the state-of-the-art in moving target search. Both known terrains and partially known terrains are used in the evaluation. In a partially known terrain the original map file is known to both agents whereas additional obstacles generated at random are not. We left an evaluation in dynamic domains—in which initially unknown obstacles may disappear during execution—out of the scope of this short paper.

Following [20], in the first part of our evaluation we use a 10% obstacle rate, and thus if a cell is not an obstacle in the original map, we set it to be an obstacle with 10% probability. The hunter and the prey agents can observe the actual status of such a cell when they attempt to move to it. For MtsCopa we evaluated three values for the $k$ parameter: 0, 20, and $\infty$. In addition we considered an implementation with a variable $k$, in which after each search for a new path, $k$ is set to the maximum between 10 and a percentage of the size of newly found path. Specifically we used 25%, 50%, and 75%.

In partially known terrain I-ARA* is run in repeated mode, which means that each time an unknown obstacle is found by the hunter, search is restarted from scratch. I-ARA* was always run with its parameter $\epsilon$ set to 2.

[2]The distance could decrease by more than one if the prey is not escaping from the hunter.

[3]http://movingai.com/benchmarks/

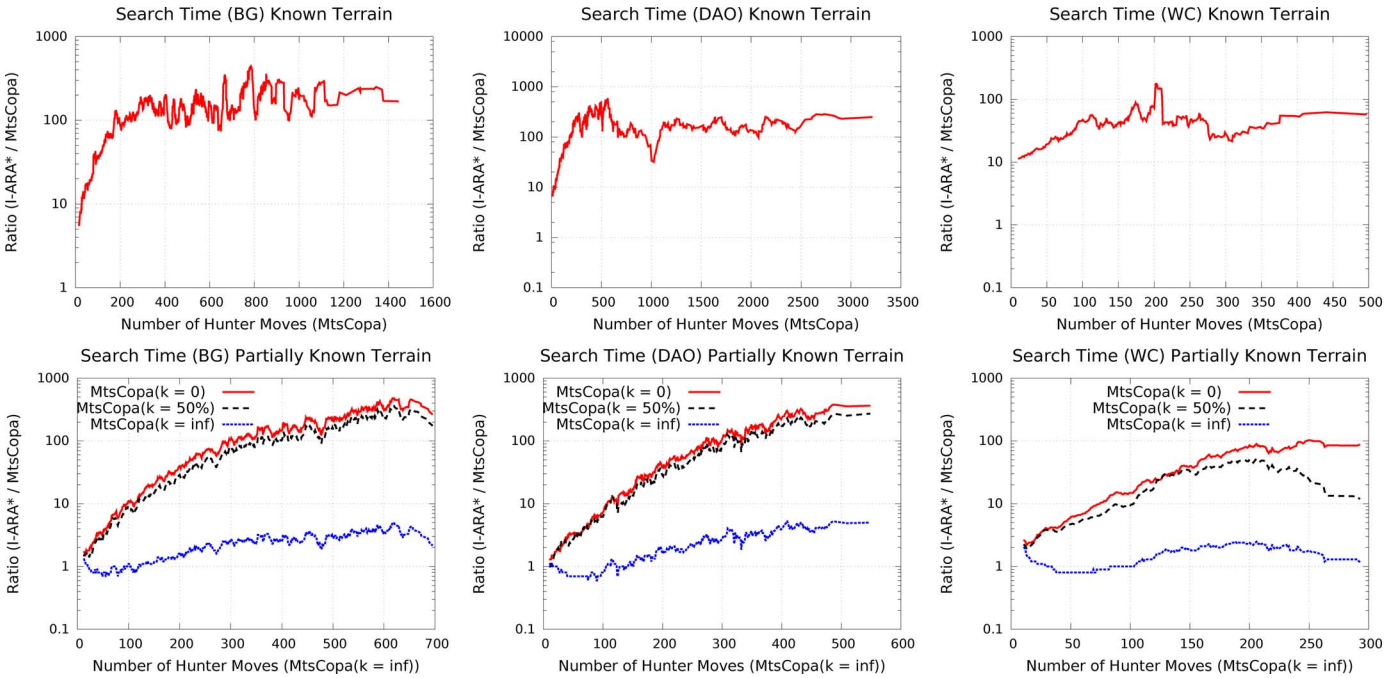


Fig. 1. MtsCopa versus I-ARA*. Ratio between time required by the MtsCopa hunter and an I-ARA* hunter in partially known and known terrain.

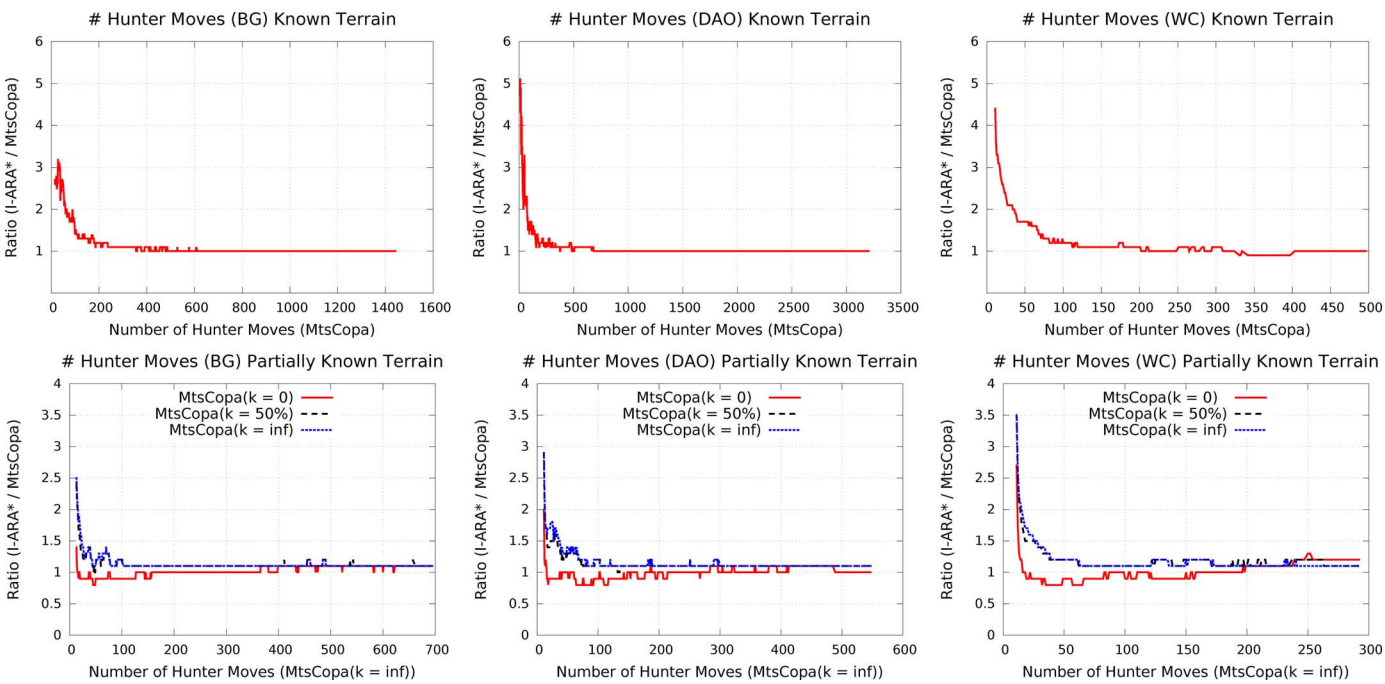


Fig. 2. MtsCopa versus I-ARA*. Relative improvement in the number of hunter moves taken by MtsCopa in known and partially known terrain.

To report our results, we use two metrics: the total time the hunter spent computing where to move next (i.e., total search time) and the total number of moves taken by the hunter to catch the prey. For all experiments we imposed a range of time limits per move: 3, 30, 300, and 3000 $\mu$s. The latter value is the time limit for which Sun *et al.* [19] report best results for I-ARA*, and this is the value we use to compare MtsCopa against I-ARA* in Figs. 1 and 2. For partially known terrain those figures report the variant of MtsCopa ($k = 50\%$) which exhibited the best compromise between number of moves and total search time; we also report the variants $k = 0$ and $k = \infty$ as a reference.

## VI. Results

Tables II and III show, for both I-ARA* and MtsCopa, how often the computation of a hunter move exceeded the time limit at hand, and the average number of hunter moves required to reach the target. When the time limit is very small, the I-ARA* hunter often is unable to compute a move within the alotted time but as the time limit is increased I-ARA* meets the limit more frequently. In known terrain MtsCopa never exceeds the time limits when the limit is more than 30 $\mu$s and exceeds it very seldom otherwise (less than 0.5% of the time). In partially known terrain, both algorithms exceed the time limit more often. This is because more search is

TABLE II
I-ARA* VERSUS MTSCOPA: PERCENTAGE OF MOVES EXCEEDING A PRE-DEFINED SEARCH TIME LIMIT PER MOVE AND AVERAGE NUMBER OF MOVES REQUIRED TO CATCH THE TARGET IN KNOWN TERRAIN

| | BG | | DAO | | WC3 | |
|---|---|---|---|---|---|---|
| | Exceeded | Moves | Exceeded | Moves | Exceeded | Moves |
| $t = 3\mu$s | | | | | | |
| I-ARA* | 73.2% | 380.0 | 72.0% | 492.8 | 66.3% | 150.4 |
| MtsCopa | 0.9% | 300.1 | 0.3% | 391.4 | 0.5% | 109.9 |
| $t = 30\mu$s | | | | | | |
| I-ARA* | 41.2% | 362.7 | 36.2% | 471.7 | 17.3% | 135.6 |
| MtsCopa | 0.0% | 300.1 | 0.0% | 391.4 | 0.0% | 109.9 |
| $t = 300\mu$s | | | | | | |
| I-ARA* | 9.5% | 336.5 | 9.5% | 437.3 | 1.7% | 128.3 |
| MtsCopa | 0.0% | 300.1 | 0.0% | 391.4 | 0.0% | 109.9 |
| $t = 3000\mu$s | | | | | | |
| I-ARA* | 0.4% | 331.7 | 0.6% | 434.9 | 0.0% | 128.1 |
| MtsCopa | 0.0% | 300.1 | 0.0% | 391.4 | 0.0% | 109.9 |

needed to deal with additional obstacles. MtsCopa, run with $k \in \{0, 20, 25\%, 50\%, 75\%\}$ rarely exceeds the time deadlines we tried. However, as expected, MtsCopa with $k = \infty$ can require more time than I-ARA*, depending on the deadline.

As expected, when given a tighter deadline I-ARA* returns worse solutions, since it has less time to refine a solution. In known terrain, average solution quality is between 14% and 17% worse when the time limit is set to 3 $\mu$s compared to when it is set to 3000 $\mu$s. In partially known terrain, on the other hand, average solution quality is between 4.4% to 11% worse when the time limit is set to 3 $\mu$s compared to when it is set to 3000 $\mu$s.

In Fig. 1, we observe that MtsCopa outperforms I-ARA* significantly in terms of runtime on every tested map, in both known and partially known terrain. In known terrain, speedups are usually around 2 orders of magnitude for the BG and DAO maps, while for the WC maps they are usually around 1.5 orders of magnitude. In partially known terrain, on the other hand, when $k \in \{0, 50\%\}$, the speedup ranges between 1 to 2.5 orders of magnitude for around 75% of the hardest instances of the BG and DAO maps, while they are between 1 and 2 orders of magnitude for around 70% of the instances on the WC maps. On the other hand, when $k = \infty$—a value for which MtsCopa runs an A* search in each iteration—the speedup is still significant in the DAO and BG maps: usually MtsCopa is between 2 and 3 times faster. Recall that, in MtsCopa, an A* search can finish earlier than a standard A* search, as CPDs can be used for heuristic guidance and for a customized goal test. This helps explaining this speed up. At the same time, for $k = \infty$, in a few problems we observe I-ARA* that outperforms MtsCopa.

Since MtsCopa moves over a shortest path to the target, we observe, in Fig. 2, an improvement over I-ARA* in the total number of moves required by the hunter to catch the prey. As instances get harder in terms of MtsCopa hunter moves, the hunter-moves performance of I-ARA* improves in known terrain. We observe a similar behavior in partially known terrain for $k = 50\%$ and $k = \infty$. However, $k = 0$ most times obtains solutions worse than I-ARA*.

TABLE III
I-ARA* VERSUS MTSCOPA: PERCENTAGE OF MOVES EXCEEDING A PREDEFINED SEARCH TIME LIMIT PER MOVE AND AVERAGE NUMBER OF MOVES REQUIRED TO CATCH THE TARGET IN PARTIALLY KNOWN TERRAIN

| | BG | | DAO | | WC3 | |
|---|---|---|---|---|---|---|
| | Exceeded | Moves | Exceeded | Moves | Exceeded | Moves |
| $t = 3\mu$s | | | | | | |
| I-ARA* | 61.6% | 307.7 | 59.1% | 264.5 | 58.9% | 116.7 |
| MTSCopa(0) | 2.0% | 289.6 | 2.8% | 258.5 | 3.9% | 117.6 |
| MTSCopa(20) | 10.0% | 271.1 | 12.5% | 238.8 | 22.2% | 96.9 |
| MTSCopa($\infty$) | 96.1% | 262.3 | 95.1% | 225.4 | 90.3% | 91.2 |
| MTSCopa(25%) | 3.9% | 264.5 | 4.6% | 230.6 | 9.0% | 96.3 |
| MTSCopa(50%) | 4.7% | 262.3 | 5.5% | 226.3 | 10.8% | 91.5 |
| MTSCopa(75%) | 7.0% | 262.2 | 8.0% | 225.5 | 15.3% | 91.1 |
| $t = 30\mu$s | | | | | | |
| I-ARA* | 34.7% | 305.4 | 28.4% | 262.2 | 17.5% | 114.3 |
| MTSCopa(0) | 0.8% | 289.6 | 1.0% | 258.5 | 1.1% | 117.6 |
| MTSCopa(20) | 1.0% | 271.1 | 1.2% | 238.8 | 1.4% | 96.9 |
| MTSCopa($\infty$) | 64.7% | 262.3 | 60.4% | 225.4 | 27.7% | 91.2 |
| MTSCopa(25%) | 0.9% | 264.5 | 1.0% | 230.6 | 1.3% | 96.3 |
| MTSCopa(50%) | 1.3% | 262.3 | 1.4% | 226.3 | 1.9% | 91.5 |
| MTSCopa(75%) | 2.3% | 262.2 | 2.5% | 225.5 | 2.9% | 91.1 |
| $t = 300\mu$s | | | | | | |
| I-ARA* | 11.4% | 299.1 | 9.5% | 256.4 | 2.3% | 106.9 |
| MTSCopa(0) | 0.1% | 289.6 | 0.1% | 258.5 | 0.0% | 117.6 |
| MTSCopa(20) | 0.2% | 271.1 | 0.1% | 238.8 | 0.0% | 96.9 |
| MTSCopa($\infty$) | 1.5% | 262.3 | 0.2% | 225.4 | 0.1% | 91.2 |
| MTSCopa(25%) | 0.2% | 264.5 | 0.2% | 230.6 | 0.0% | 96.3 |
| MTSCopa(50%) | 0.2% | 262.3 | 0.2% | 226.3 | 0.0% | 91.5 |
| MTSCopa(75%) | 0.2% | 262.2 | 0.2% | 225.5 | 0.0% | 91.1 |
| $t = 3000\mu$s | | | | | | |
| I-ARA* | 0.8% | 294.0 | 0.8% | 253.2 | 0.0% | 104.6 |
| MTSCopa(0) | 0.0% | 289.6 | 0.0% | 258.5 | 0.0% | 117.6 |
| MTSCopa(20) | 0.0% | 271.1 | 0.0% | 238.8 | 0.0% | 96.9 |
| MTSCopa($\infty$) | 0.0% | 262.3 | 0.0% | 225.4 | 0.0% | 91.2 |
| MTSCopa(25%) | 0.0% | 264.5 | 0.0% | 230.6 | 0.0% | 96.3 |
| MTSCopa(50%) | 0.0% | 262.3 | 0.0% | 226.3 | 0.0% | 91.5 |
| MTSCopa(75%) | 0.0% | 262.2 | 0.0% | 225.5 | 0.0% | 91.1 |

To observe the performance of the algorithms over different obstacle rates, we ran an additional experiment, in which we chose one map from each benchmark group. Specifically we tested for 4%, 8%, and 12% obstacle rates. Table IV shows a summary of performance indicators. As before, on average MTSCopa outperforms I-ARA* by two orders of magnitude in terms of time, finding slightly better solutions.

We conclude that in known terrain MtsCopa is clearly superior to I-ARA*, in terms of speed and solution quality. In partially known terrain, MtsCopa(50%) is the algorithm achieves the best balance in between of hunter moves and time performance, outperforming I-ARA* significantly in runtime, while obtaining solutions of similar quality.

TABLE IV
AVERAGE NUMBER OF MOVES AND RUNTIME FOR 3 GAME MAPS (DARKFOREST, ORZ100D, AR0300SR) FOR DIFFERENT OBSTACLE RATES

| | Moves | | | Runtime(ms) | | |
|---|---|---|---|---|---|---|
| | 4% obs | 8% obs | 12% obs | 4% obs | 8% obs | 12% obs |
| IARA* | 182.9 | 184.0 | 191.0 | 34.71 | 39.16 | 46.11 |
| MTSCopa(25%) | 184.3 | 172.4 | 169.7 | 0.48 | 0.40 | 0.41 |
| MTSCopa(50%) | 170.4 | 167.0 | 166.8 | 0.51 | 0.47 | 0.49 |
| MTSCopa(75%) | 168.3 | 166.2 | 166.7 | 0.89 | 1.21 | 1.56 |

### *A. Alternative Prey Movement Strategies*

Above we evaluated a standard prey moving strategy [19]. Since MTSCopa outperforms I-ARA* by a large margin we regarded as unlikely that a change in the prey's moving strategy would alter the outcome of the comparison. To confirm this we implemented a hunter-aware movement strategy in which the prey, instead of choosing a random location as the next movement, draws 10 possible destinations at random and chooses the one that maximizes the angle formed between the hunter, the current position (vertex), and the candidate destination. We tested both algorithms over 2000 problems on the AR0300SR map. As before, MTSCopa(50%) significantly outperformed I-ARA*. With respect to the original moving strategy, hunter moves increased by 6.7% and 5.2% for I-ARA* and MtsCopa(50%), respectively. While with the original strategy MtsCopa(50%) ran 26 times faster than I-ARA*, with this new one it ran 21 times faster. This small evaluation supports the fact that MtsCopa outperforms I-ARA* significantly regardless of the prey movement strategy. A large-scale evaluation under other more sophisticated moving strategies (e.g., [14]) is left for future work.

## VII. SUMMARY AND FUTURE WORK

Moving target search [9] differs from standard path planning in that the goal location changes as the hunter moves along. Previous approaches to moving target search do not exploit previously computed information. In this work, we show how precomputed information in the form of a compressed path database [1] can be exploited to improve efficiency. We proposed MtsCopa, a simple moving-target search solver, which is suitable for both known and partially known terrain.

Unlike existing moving-target search methods, MtsCopa needs preprocessing time and memory to cache the results. When such resources are available, MtsCopa advances state-of-the-art in moving target search convincingly. Its speed is orders of magnitude better than I-ARA*. The quality of solutions is empirically shown to be good. We showed that the MtsCopa has a very good real-time performance in practice, providing individual moves very quickly at all stages of a chase, including the first move. In addition, we proved that MtsCopa leads the hunter to catch its prey in partially known environments if the prey moves more slowly.

In future work, we plan to study our ideas in a multihunter, multiprey setting. In addition, we believe that our algorithm could be successful to static-target search in dynamic environments. The ability to adjust a CPD as changes are discovered in the graph would be beneficial.

## ACKNOWLEDGMENT

The authors thank X. Sun for making the I-ARA* source code available to us. They also thank the anonymous reviewers who provided detailed feedback that help improve the quality of this paper.

## REFERENCES

[1] A. Botea, V. Bulitko and M. O. Riedl, Eds., "Ultra-fast optimal pathfinding without runtime search," in *Proc. 7th AAAI Conf. Artif. Intell. Interact. Digit. Entertain. (AIIDE)*, Stanford, CA, USA, Oct. 2011.

[2] A. Botea, D. Borrajo, A. Felner, R. E. Korf, M. Likhachev, C. L. López, W. Ruml, and N. R. Sturtevant, Eds., "Fast, optimal pathfinding with compressed path databases," in *Proc. 5th Ann. Symp. Combinator. Search (SoCS)*, Niagara Falls, Ontario, Canada, 2012.

[3] A. Botea, J. A. Baier, D. Harabor, and C. Hernández, D. Borrajo, S. Kambhampati, A. Oddi, and S. Fratini, Eds., "Moving target search with compressed path databases," in *Proc. 23rd Int. Conf. Autom. Plan. Schedul. (ICAPS)*, Rome, Italy, 2013.

[4] V. Bulitko, D. C. Rayner, and R. Lawrence, M. Riedl and G. Sukthankar, Eds., "On case base formation in real-time heuristic search," in *Proc. AIIDE*, 2012.

[5] V. Bulitko and M. O. Riedl, Eds., in *Proc. 7th AAAI Conf. Artif. Intell. Interact. Digit. Entertain.*, Stanford, CA, USA, Oct. 2011.

[6] G. Hahn and G. MacGillivray, "A note on k-cop, l-robber games on graphs," *Discr. Math.*, vol. 306, no. 19-20, pp. 2492–2497, 2006.

[7] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci., Cybern.*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[8] C. Hernández, X. Sun, S. Koenig, and P. Meseguer, L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, Eds., "Tree Adaptive A*," in *Proc. 10th Int. Conf. Autonom. Agents Multiagent Syst. (AAMAS)*, Taipei, Taiwan, May 2011, pp. 123–130.

[9] T. Ishida and R. E. Korf, "Moving-target search: A real-time search for changing goals," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 6, pp. 609–619, Jun. 1995.

[10] S. Koenig and M. Likhachev, "Adaptive A*," in *Proc. Int. Joint Conf. Autonom. Agents Multiagent Syst. (AAMAS)*, 2005, pp. 1311–1312.

[11] S. Koenig, M. Likhachev, and X. Sun, E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds., "Speeding up moving-target search," in *Proc. 6th Int. Joint Conf. Autonom. Agents Multiagent Syst. (AAMAS)*, Honolulu, HI, USA, May 2007, p. 188.

[12] R. Lawrence and V. Bulitko, Li J, Ed., "Taking learning out of real-time heuristic search for video-game pathfinding," in *Proc. Austral. Conf. Artif. Intell.*, 2010, vol. 6464, Lecture Notes in Comput. Sci., pp. 405–414.

[13] R. Lawrence and V. Bulitko, "Database-driven real-time heuristic search in video-game pathfinding," *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 3, pp. 227–241, 2013.

[14] C. Moldenhauer and N. R. Sturtevant, C. Boutilier, Ed., "Evaluating strategies for running from the cops," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, Pasadena, CA, USA, Jul. 2009, pp. 584–589.

[15] C. Moldenhauer and N. R. Sturtevant, C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, Eds., "Optimal solutions for moving target search," in *Proc. 8th Int. Joint Conf. Autonom. Agents Multiagent Syst. (AAMAS)*, Budapest, Hungary, May 2009, vol. 2, pp. 1249–1250.

[16] N. Rivera, L. Illanes, J. A. Baier, and C. Hernández, M. Helmert and G. Röger, Eds., "Reconnecting with the ideal tree: An alternative to heuristic learning in real-time search," in *Proc. 6th Ann. Symp. Combinator. Search (SoCS)*, Leavenworth, WA, USA, Jul. 2013.

[17] N. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 144–148, 2012.

[18] X. Sun, W. Yeoh, and S. Koenig, W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, Eds., "Moving target D* Lite," in *Proc. 9th Int. Conf. Autonom. Agents Multiagent Syst. (AAMAS)*, Toronto, Canada, 2010, pp. 67–74.

[19] X. Sun, W. Yeoh, T. Uras, and S. Koenig, L. McCluskey, B. Williams, J. R. Silva, and B. Bonet, Eds., "Incremental ARA*: An Incremental Anytime Search Algorithm for Moving-Target Search," in *Proc. 22nd Int. Conf. Autom. Plan. Schedul. (ICAPS)*, Atibaia, São Paulo, Brazil, 2012.

[20] P. K. Y. Yap, N. Burch, R. C. Holte, and J. Schaeffer, V. Bulitko and M. O. Riedl, Eds., "Any-angle path planning for computer games," in *Proc. 7th AAAI Conf. Artificial Intell. Interact. Digit. Entertain. (AIIDE)*, Stanford, CA, USA, Oct. 2011.