Moving Target Search with Compressed Path Databases

Adi Botea
IBM Research
Dublin, Ireland

Jorge A. Baier[*]
Computer Science Department
PUC Chile
Santiago, Chile

Daniel Harabor
NICTA and
Australian National University
Canberra, Australia

Carlos Hernández
Depto. de Ingeniería Informática
Universidad Católica
de la Ssma. Concepción
Concepción, Chile

Abstract

Moving target search, where the goal state changes during a search, has recently seen a revived interest. Incremental Anytime Repairing A* (I-ARA*) is a very recent, state-of-the-art algorithm for moving target search in a known terrain. In this work, we address the problem using compressed path databases (CPDs) in moving target search. CPDs have previously been used in standard, fixed-target pathfinding. They encode all-pairs shortest paths in a compressed form and require preprocessing and memory to store the database.

In moving-target search, our speed results are orders of magnitude better than 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. The number of hunter moves is very good, 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.

Introduction

The objective of *moving target search* (MTS) (Ishida and Korf 1991) is to decide the moves of a *hunter* agent in order to allow it to capture a moving, so-called *target* agent or *prey*. MTS algorithms can be classified into offline and online methods (Sun et al. 2012). While offline algorithms (Hahn and MacGillivray 2006; Moldenhauer and Sturtevant 2009) 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 (Sun, Yeoh, and Koenig 2010; Sun et al. 2012) compute a complete path between the hunter and the target which is then followed by the hunter until the target moves off the path or until it is caught. Reactive on-line algorithms (Ishida and Korf 1991; 1995), 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.

In this paper we focus on online MTS in initially known environments. This problem has direct applications in video games, where, often, agents are required to catch targets. We propose MtsCopa, an algorithm that uses a compressed path database (CPD) (Botea 2011) to decide the hunter's moves. CPDs are built in a preprocessing step that computes all-pairs shortest-paths and then stores compressed paths in the CPD. Given a current position s and a goal position t, they allow to retrieve the next move on an optimal path from s to t very quickly. As such, in the main loop our algorithm decides the next move for the hunter by retrieving a record from the CPD and thus carrying out no search on the map at all. In addition, the returned move belongs to a shortest path from the current position to the target agent. Since the target moves dynamically, there is no guarantee that using shortest-path moves for the hunter minimises the total number of moves. However, both our results and previous results (e.g., Sun et al. 2012) show that, in practice, taking optimal moves towards the current prey position leads to fewer hunter moves.

MtsCopa is simple to implement as it just requires an interface to the CPD. Similarly to I-ARA* (Sun et al. 2012) and other online moving-target search methods, our algorithm is suitable for static and fully known terrains. We evaluate MtsCopa on standard pathfinding benchmarks and compare it to I-ARA*. 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 describes our method, presents our experimental evaluation, and finalizes with conclusions.

Approach

A generic moving target search framework is illustrated in Algorithm 1. At each iteration, the hunter and the prey will

[*]J. Baier was partly funded by Fondecyt Grant No. 11110321.

take one action each. Once in a while, the prey's action is to wait, to ensure that the hunter eventually catches it. Flag "captured" becomes true as soon as the hunter's and the prey's locations are identical.

Algorithm 1 Generic framework for moving target search.

```
$h \leftarrow s_0$ {initialise hunter position to a given location $s_0$}
$p \leftarrow t_0$ {initialise prey position}
**while** not captured **do**
    $h \leftarrow$ getHunterNextPosition(...)
    $p \leftarrow$ getPreyNextPosition(...)
```

Using Copa to implement a strategy for the hunter (or the prey) is simple. Given a map, the program performs an offline preprocessing to compute a CPD. At runtime, a CPD has the ability to answer fast *first-move queries*: given *any* start–target pair, return a move from the start towards the target along an optimal path. Such a property makes a CPD particularly useful in problems such as moving target search, where both the hunter's and the prey's position change dynamically. Our Copa-based implementation of the hunter, which we call MtsCopa, queries the CPD for the start–target pair (h, p), and uses the corresponding move.

Experimental Setup

We have compared our moving-target search solver, based on Copa (Botea 2012), with the recent, state-of-the-art program I-ARA* (Sun et al. 2012). The source code of Copa and I-ARA* were obtained from their respective authors.

In our experiments we used 17 grid maps sourced from a variety of domains: computer game maps, maze maps, maps with random obstacles and synthetic maps partitioned into rooms with entrances. All are taken from Sturtevant's online repository[1] (Sturtevant 2012). Following the approach of Sun et al. (2012), all maps are considered to be 4-connected.

Table 1 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 12 times smaller than reported in the table since preprocessing was run on a 12-core machine. Apart from preprocessing, all experiments were run on a serial, single-core setting. We used a 3.47GHz machine running Red Hat Enterprise.

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 file can differ but each one contains at least 2000 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. As in previous work (e.g., Sun et al. 2012), the prey 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 (Sun et al. 2012). Choosing a destination from the prey uses 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.

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

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
divide-conq-2	193x193	19,662	3.6	2.8
divide-conq-3	676x676	311,040	203.0	1222.7
thecrucible	130x130	10,280	2.0	0.7
battleground	512x512	92,268	27.0	98.9
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 (BG2)				
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
Domain Rooms				
32room_000	512x512	240,671	185.0	799.8
Domain Mazes				
maze512-32-0	512x512	253,840	56.0	804.3
maze512-2-0	512x512	174,524	35.0	297.2
Domain Random (25% Obstacles)				
rand512-25-0	512x512	195,313	271.0	419.9
rand512-25-1	512x512	195,352	271.0	419.9

Table 1: 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).

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. Each of these eventually returns a new path whose cost is improved with respect to the previously found path. 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, meeting the limit is not always guaranteed. In our experiments, we evaluate I-ARA*'s performance for different time limits.

Results

Figures 1 and 2 detail the main results from our experiments. The set of 17 maps was partitioned into 6 groups, according to the 6 domains outlined in Table 1. We compared the performance of MtsCopa with I-ARA*. We use two metrics:

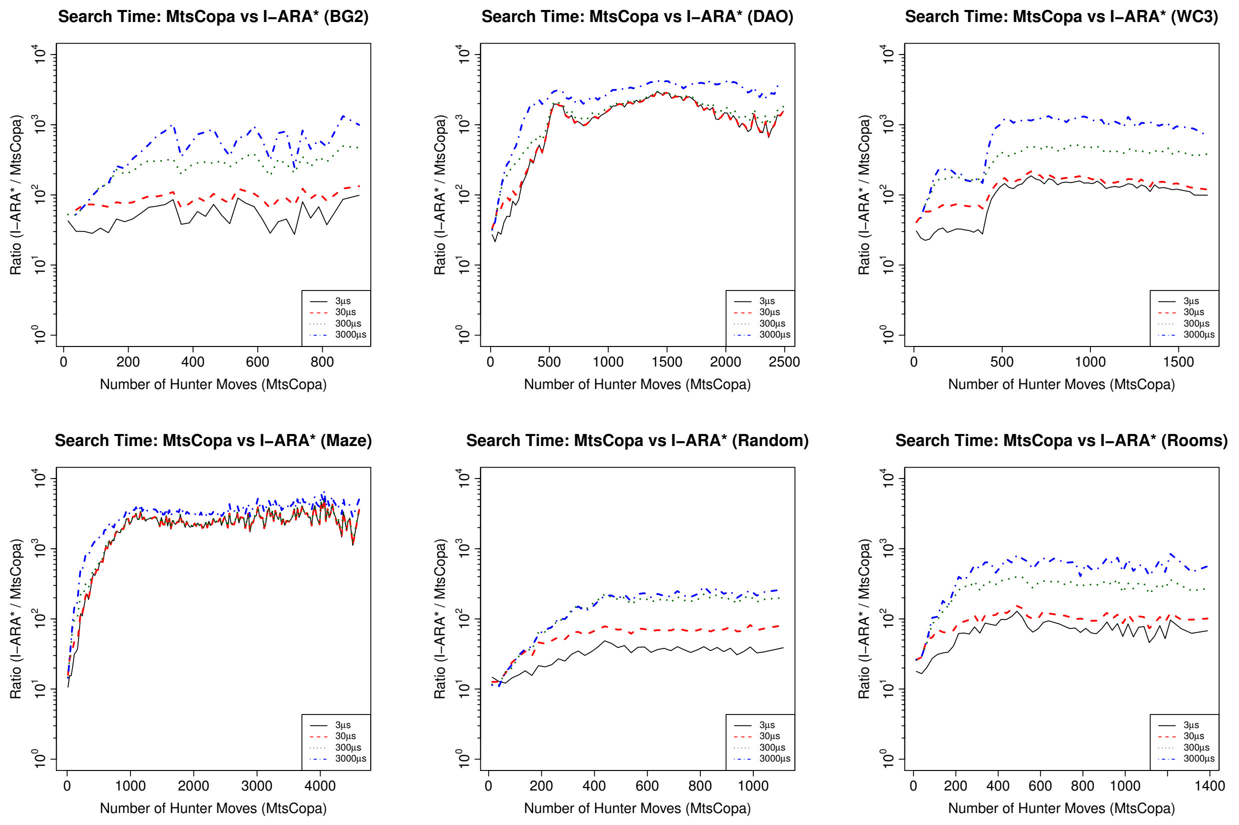

Figure 1: MtsCopa vs. I-ARA*. Relative improvement in the time required by the MtsCopa hunter to compute a next move.

	BG	DAO	Maze	Rand	Rooms	WC3
			$t = 3\mu$s			
I-ARA*	73.3	65.1	42.9	79.5	78.0	71.3
MtsCopa	0.0	0.0	0.0	0.0	0.0	0.0
			$t = 30\mu$s			
I-ARA*	45.0	55.0	36.9	47.4	54.2	40.0
MtsCopa	0.0	0.0	0.0	0.0	0.0	0.0
			$t = 300\mu$s			
I-ARA*	15.0	43.0	30.1	17.0	26.4	12.1
MtsCopa	0.0	0.0	0.0	0.0	0.0	0.0
			$t = 3000\mu$s			
I-ARA*	1.0	22.7	18.2	0.5	2.0	1.3
MtsCopa	0.0	0.0	0.0	0.0	0.0	0.0

Table 2: I-ARA* vs. MtsCopa: percentage of moves exceeding a pre-defined search time limit per move.

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. In each case I-ARA* is the baseline and performance is reported as the ratio between I-ARA* and MtsCopa. Values higher than 1 indicate MtsCopa performed better and values less than 1 indicate the opposite. For all experiments in Figure 1 and 2 we imposed a range of time limits: 3μs, 30μs, 300μs and 3000μs. The latter value is the time limit for which Sun et al. (2012) report best results for I-ARA*.

In Figure 1, we observe that MtsCopa dramatically outperforms I-ARA* in terms of runtime on every tested map. On maps from the Random and Rooms domains MtsCopa is between 1-2 orders of magnitude faster than the previously best reported results for I-ARA*. In other cases, such as Mazes and DAO, search times can be 3.5 orders of magnitude faster than I-ARA*.

Since MtsCopa extracts an optimal[2] move at every iteration of the search, we observe, in Figure 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. The largest differences between MtsCopa and I-ARA* are for the easiest instances, after which the difference quickly reduces and stabilises. In some domains, such as Random, Rooms, WC3, the impact of the time limit in I-ARA* is clearly exhibited. The more time available, the better the solutions it returns, and thus the smaller gap to MtsCopa. In other domains, such as Mazes and DAO, the time limit has a much smaller impact on the number of hunter moves. See a discussion at the end of this section.

In Table 2 we recorded, for both I-ARA* and MtsCopa,

[2]Optimal in the sense discussed in the introduction.

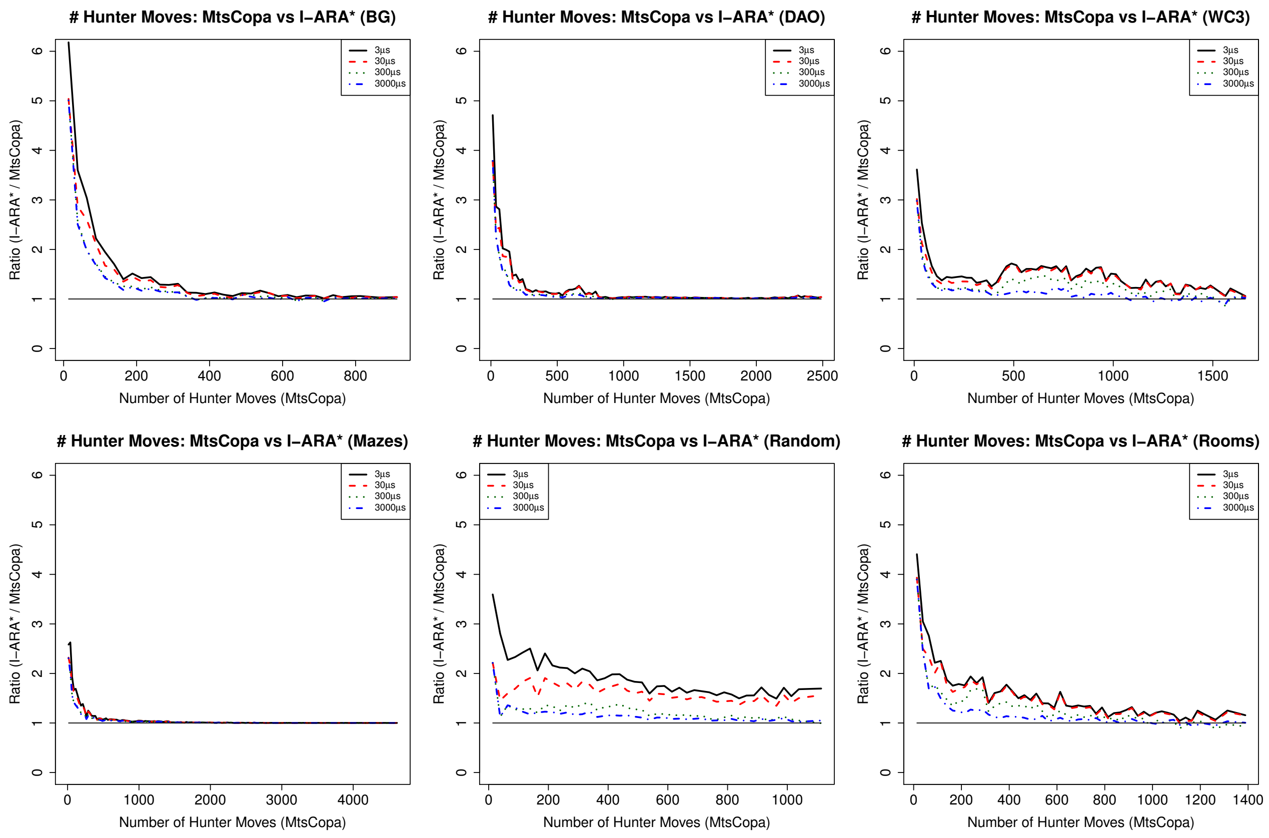

Figure 2: MtsCopa vs. I-ARA*. Relative improvement in the number of hunter moves taken by MtsCopa.

how often the computation of a hunter move exceeded the time limit at hand. When the time limit is very small, the I-ARA* hunter often is unable to compute a move within the alotted time. As the time limit is increased I-ARA* meets the limit more frequently. MtsCopa, on the other hand, never exceeds the time limits when the limit is more than 30μs and very seldom otherwise (less than 0.1% of the time).

Domains DAO and especially Mazes exhibit a different behaviour from the other domains. The speed gap is greater (Figure 1), the solution quality gap is smaller (Figure 2), and the performance of I-ARA* is more stable when varying the time limit per episode (Figures 1 and 2, and Table 2). We attribute this to the map topologies which, in Mazes and partly in DAO, contain many narrow corridors. A maze-like topology makes searches with the Manhattan heuristic more difficult, resulting in a greater speed gap. Searches can exceed even the largest time limit used in experiments, as shown in Table 2, at the bottom. Narrow corridors help I-ARA* make optimal or nearly optimal moves towards the target, since there are fewer paths between the hunter and the target. When the time limit is small, I-ARA* stays within the alloted time more often in DAO and Mazes than in other domains (Table 2, at the top), which suggests that the results of a search can be reused more often in Mazes and DAO.

Conclusion

In moving target search (Ishida and Korf 1991), fully computed paths are often invalidated, as both the hunter and the prey change their positions frequently. In this work, we took an approach that wastes no online effort to compute subsequent moves earlier than needed. MtsCopa, our simple moving-target search solver, makes use of compressed path databases (Botea 2011). Given *any* combination (h, p) of a hunter's and a prey's positions, a CPD provides fast an optimal move from h towards p.

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, since Copa returns optimal moves towards a given target. We showed that the program has a very good real-time performance in practice. It provides individual moves very quickly at all stages of a chase, including the first move.

In future work, we plan to study our ideas in a multi-hunter, multi-prey setting. Another interesting direction is adapting MtsCopa to partially known and dynamic environments, possibly in a hybrid approach in combination with real-time search or incremental search.

References

Botea, A. 2011. Ultra-fast Optimal Pathfinding without Runtime Search. In *Proceedings of AIIDE-11*, 122–127.

Botea, A. 2012. Fast, optimal pathfinding with compressed path databases. In Borrajo, D.; Felner, A.; Korf, R. E.; Likhachev, M.; López, C. L.; Ruml, W.; and Sturtevant, N. R., eds., *SOCS*. AAAI Press.

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

Ishida, T., and Korf, R. E. 1991. Moving Target Search. In *IJCAI*, 204–210.

Ishida, T., and Korf, R. E. 1995. Moving-target search: A real-time search for changing goals. *IEEE Trans. Pattern Anal. Mach. Intell.* 17(6):609–619.

Moldenhauer, C., and Sturtevant, N. R. 2009. Optimal solutions for moving target search. In Sierra, C.; Castelfranchi, C.; Decker, K. S.; and Sichman, J. S., eds., *AAMAS (2)*, 1249–1250. IFAAMAS.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144 – 148.

Sun, X.; Yeoh, W.; Uras, T.; and Koenig, S. 2012. Incremental ara*: An incremental anytime search algorithm for moving-target search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *ICAPS*. AAAI.

Sun, X.; Yeoh, W.; and Koenig, S. 2010. Moving target d* lite. In van der Hoek, W.; Kaminka, G. A.; Lespérance, Y.; Luck, M.; and Sen, S., eds., *AAMAS*, 67–74. IFAAMAS.