# A Scalable Approach to Chasing Multiple Moving Targets with Multiple Agents

**Fan Xie**[*]
Google Canada
cgfxie@gmail.com

**Adi Botea**
IBM Research, Ireland
adibotea@ie.ibm.com

**Akihiro Kishimoto**
IBM Research, Ireland
akihirok@ie.ibm.com

## Abstract

Chasing multiple mobile targets with multiple agents is important in several applications, such as computer games and police chasing scenarios. Existing approaches can compute optimal policies. However, they have a limited scalability, as they implement expensive minimax searches. We introduce a sub-optimal but scalable approach that assigns individual agents to individual targets and that can dynamically re-compute such assignments. We provide a theoretical analysis, including upper bounds on the number of time steps required to solve an instance. In a detailed empirical evaluation on grid maps, our algorithm scales up very convincingly beyond the limits of previous methods. On small problems, where a comparison to a minimax approach is possible, the results demonstrate a good solution quality for our method.

## 1 Introduction

In moving target search (MTS), introduced in the AI literature by Ishida and Korf [1991], a mobile agent chases a mobile target on a map. As a target's location changes constantly, an agent needs to be able to adapt its strategy very fast, making this a challenging real-time search problem.

We address the problem where multiple agents chase multiple mobile targets. We call this the *multiple agents and targets* variant of MTS, or, shortly, the MAT problem. The problem has multiple practical applications, such as pursuing suspect vehicles with police cars, and controlling non-player characters in video games. The task is to plan the moves of the agents so that they catch the mobile targets.

Addressing MAT efficiently is challenging. Existing approaches rely on minimax searches[1] either on the combined state space of all agents and all targets [Vieira *et al.*, 2008], or on sub-spaces with one target and one team of agents each [Vieira *et al.*, 2009]. Performing minimax search in the MAT problem can provide optimal policies [Vieira *et al.*, 2008]. However, minimax search introduces a major performance bottleneck. Moldenhauer and Sturtevant [2009b] have shown that, even in the simpler case of one agent and one target, minimax search is computationally expensive, reaching millions of node expansions on small gridmaps with up to 1600 nodes, much smaller than modern game gridmaps used in our experiments. The search effort grows sharply with the map size and the number of agents.

We introduce a sub-optimal but scalable approach to the MAT problem. Our method assigns individual agents to individual targets. The assignment can change dynamically. The computation of each assignment considers the current positions of agents and targets, and ignores future possible movements of the targets. This allows to avoid running minimax searches, greatly reducing the computational costs. Given an assignment, our method runs a series of independent single-agent, single-target chases. In each independent chase an agent uses optimal moves towards the current target position. The moves are retrieved from a compressed path database [Botea, 2011], following a state-of-the-art approach in single-agent, single-target MTS [Botea *et al.*, 2013].

We formally show that our algorithm terminates, with all targets captured, and provide upper bounds for the time steps required to complete a chase. We formally analyse the impact of the re-assignment frequency on the solution quality. We provide upper bounds for the suboptimality stemming from the decision of not recomputing a new assignment at one or several iterations in a row.

We perform a detailed empirical analysis on a collection of gridmaps from Sturtevant's repository [Sturtevant, 2012]. Our approach shows a strong scalability performance. It can solve instances with up to 200 agents and 200 targets, on a map with nearly 100,000 nodes, within the order of one second, or even a fraction of a second. On the other hand, previous approaches do not scale beyond small instances with a few agents and much smaller graphs. On small instances, where a comparison to an optimal minimax policy is possible, our system is shown to produce solutions of a good quality, as their sub-optimality is no larger than 12.84 % on average.

## 2 Related Work

The "standard" MTS assumes a single agent, a single target, and full knowledge about the environment and the target's

[*]Part of this work has been performed when the first author was a visitor to IBM Research, Ireland.

[1]In this work, we call a minimax search a search in the combined space of two or more units, where hunter agents minimize the duration of a chase, and targets maximize it.

position. Recent successful approaches to standard MTS include incremental search, reusing part of the results computed in a previous search round [Sun *et al.*, 2012], using compressed path databases [Botea *et al.*, 2013], and searching in subgoal graphs [Nussbaum and Yörükçü, 2015].

Moving target search has been investigated in partially known and dynamic environments [Koenig *et al.*, 2007; Sun *et al.*, 2010; Baier *et al.*, 2014]. Other research has focused on multiple agents chasing one target [Goldenberg *et al.*, 2003; Isaza *et al.*, 2008]. Goldenberg *et al.* [2003] assume that the target position is not fully known to the agents. Replaning paths for multiple agents after each target movement could be slow. In addressing this, Isaza *et al.* [2008] use cover sets, regions that an agent can reach faster than the target. Cover sets are used to coordinate multiple agents chasing one target.

Among the contributions focusing on multi-agent, multi-target scenarios, Hahn and MacGillivray [2006] discuss the problem from a theoretical angle, focusing on the existence of a winning strategy for the agents (the "cops"). Vidal *et al.* [2002] take a probabilistic approach in scenarios with heterogeneous agents, such as helicopters, that can only observe, but not capture, and ground vehicles.

Vieira *et al.*'s work [2008; 2009] is the most closely related to our research, and a summary is necessary to better motivate our work. Vieira *et al.* [2008] describe a minimax strategy for computing optimal agent policies. This approach runs a minimax search in the combined state space of all agents and all targets, which does not scale beyond small problems, with small maps and a few agents.

Vieira *et al.* [2009] take a step towards improving scalability, with a suboptimal decomposition approach. Assuming that $c(G)$ agents are sufficient to eventually capture a target, there are $r$ targets and $p \geq c(G) \times r$ agents, this work enumerates teams of $c(G)$ agents each. For each combination of a target and a team of agents, a minimax search returns a cost such as the max time to capture that target. Teams of agents are statically assigned to targets in such a way that the max capture time, across all team-target pairs considered in the assignment, is minimized. As such, this boils down to a linear bottleneck assignment problem. Vieira *et al.*'s work [2009] requires enumerating potentially many teams of agents. Then, for each combination of a team and a target, an expensive minimax search is employed. Even in the base case when $c(G) = 1$, where only trivial one-agent "teams" are considered, $p \times r$ expensive minimax searches are needed. Furthermore, these have to be performed upfront, potentially causing large first-move lags, before agents start moving.

## 3 Problem Description

Agents and targets share an environment represented as a connected graph. The time is discretized. A transition from one time step to the next is called an *iteration*. At an iteration, agents take an action each, after which targets take an action each. Actions include moving to an adjacent node, or staying put. An agent can capture a target when they share the same location. Due to space limitations, we focus on the case when agents capture only their currently assigned target. An instance is solved when all targets are captured.

Head-to-head collisions between two agents are easier to address than in standard multi-agent path finding, as agents are free to swap their targets in our problem. Swapping the targets of the two agents results in swapping their travel direction. The result is equivalent to allowing the agents to pass by each other. In this work we assume that agents can pass by each other. We further assume that multiple agents can share a given location, a typical assumption in multi-agent moving target search [Goldenberg *et al.*, 2003; Isaza *et al.*, 2008; Vieira *et al.*, 2008; 2009]. More generally, any instance can be solved while restricting the number of agents per location (e.g., at most 1), thanks to the freedom of assigning any target to any agent, but this is beyond the focus of this short paper.

The graph is static and unweighted. The positions of the agents and targets are known. Agents move slightly faster than targets, to ensure that a chase eventually finishes. For a given target, this is implemented by requiring that, once in $l$ iterations, where $l$ is a parameter (the *stay-put parameter*), the only action available to the target at that iteration is staying put. We set $l = 10$, as previously done in the literature (e.g., [Sun *et al.*, 2012; Botea *et al.*, 2013]).

## 4 Our Approach Overview

**Algorithm 1** Generate Agent Moves

**Input** Agent positions $p_a$, target positions $p_t$, map (graph) $g$, (possibly empty) assignment $a$
**Output** Agent moves

```
1: if new assignment desired at this iteration then
2:     $d \leftarrow$ ComputeDistance($p_a, p_t, g$)
3:     $a \leftarrow$ ComputeAssignment($d$)
4: $m \leftarrow$ GenerateMoves($a, p_a, p_t, g$)
5: return $m$
```

Algorithm 1 outlines how agent moves are generated. This is invoked at each iteration, until all targets are captured. We compute $d$ as an exact distance matrix, using compressed distance databases [Xie *et al.*, 2015] as a distance oracle.

Method ComputeAssignment($d$) assigns one target to each agent. When a new assignment is computed, it becomes the *active assignment*. We discuss our assignment strategies in more detail in the next section. The frequency of computing a new active assignment (i.e., lines 1 to 3) is analysed theoretically in Section 6 and evaluated empirically in the experiments section.

As soon as an active assignment is available (step 4), we reduce the problem to a collection of independent single-agent, single-target moving target search instances (i.e., standard MTS instances). This is implemented in method GenerateMoves, which returns a move for each agent. We address each standard MTS instance with a state-of-the-art approach from the literature [Botea *et al.*, 2013]. As such, an optimal move from the current location of an agent to the current location of its assigned target is fetched fast, by querying an oracle such as a compressed path database [Botea, 2011]. No effort is wasted in computing a longer path fragment than just one move at a time.

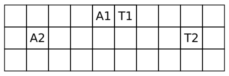

Figure 1: Illustrating assignments on a 4-connected gridmap.

Our approach is modular, decomposing the original problem into subproblems. This allows to address part of the subproblems involved with state-of-the-art approaches to those subproblems from the literature. For example, we leverage existing solutions to the assignment problem and to the single agent, single target MTS problem.

Given $m$ agents and $n$ targets, in the rest of this paper we focus on scenarios with $m = n$. However, our approach easily allows addressing more general scenarios. For instance, when $m < n$, focus on $m$ targets at a time. When a target is captured, another target is assigned to an agent. When $m > n$, assign at least one agent per target. In experiments we include a brief evaluation on cases with $m \neq n$.

## 5 Assigning Agents to Targets

Computing an active assignment considers the current positions of agents and targets, but not their potential future positions. This allows to reduce the task of assigning agents to targets to classical assignment problems [Munkres, 1957; Burkard *et al.*, 2009], as shown later in this section. The exact assignment problem formulation depends on the solution quality criterion at hand. We consider three such criteria: the sum of the distances between each agent and its target (*summation cost*), the max distance (*makespan*), and a combination with the makespan as the main component. Strategies presented in this section are optimal with respect to the current positions of targets but, as mentioned earlier, not necessarily optimal with respect to future target positions. For this reason, we call them *statically optimal assignments*.

**Summation Cost Criterion.** This criterion is relevant in problems where reducing the total traveled distance matters (e.g., to reduce fuel consumption). In this case, we obtain a linear assignment problem. It consists of finding an assignment that minimizes the sum of distances between each agent and its assigned target at the current time. The problem is solved exactly in an $O(n^3)$ time [Edmonds and Karp, 1972]. We say an assignment computed this way is *statically optimal* with respect to summation cost. In Figure 1, the assignment $(A_1, T_1), (A_2, T_2)$ is statically optimal with respect to the summation cost. Its cost is $1 + 7 = 8$, whereas the cost of the assignment $(A_1, T_2), (A_2, T_1)$ is $5 + 5 = 10$.

**Theorem 1.** *With the summation cost in use, a chase terminates (with all targets captured) in at most $S_0 \times l$ iterations, where $S_0$ is the summation cost of the initial assignment and $l$ is the stay-put parameter.*

*Proof.* Let $S$ be the summation cost, initialized to $S_0$. $S$ never increases from an iteration to the next, as each agent chases its assigned target. $S$ never increases when a re-assignment is performed, as every new assignment minimizes $S$. $S$ strictly decreases when a target stays put. It follows that $S$ converges to 0 after at most $S_0 \times l$ iterations. □

**Makespan Criterion.** Reducing the makespan is important when a chase should finish as soon as possible (e.g., when police are chasing suspect cars). We obtain the known linear bottleneck assignment problem, which consists of finding an assignment that minimizes the maximum distance between an agent and its assigned target. In Figure 1, the assignment $(A_1, T_2), (A_2, T_1)$ is statically optimal with respect to the makespan. Its makespan is $\max(5, 5) = 5$, whereas the makespan of the other assignment is $\max(1, 7) = 7$.

We solve the linear bottleneck assignment problem as a modified bipartite match problem as follows. We find a minimal edge weight $w$ with the property that after removing all edges with larger weights than $w$ there still exists at least one one-to-one match. Our algorithm has the following steps: 1) Sort all edge weights in ascending order, and store them in a list $L$; 2) Binary search $L$ to find the minimal value $w$ such that, after removing all edges with larger weight than $w$, there still exists at least one assignment; 3) Return such an assignment. Step 2 dominates the time complexity. Testing whether a perfect bipartite match exists is done with the Kopcroft-Karp algorithm [Hopcroft and Karp, 1973], with a time complexity of $O(n^{2.5})$. After factoring in the binary search part, the complexity becomes $O(n^{2.5} \log n)$. We say that an assignment computed as shown above is *statically optimal* with respect to the makespan cost.

Let $M_0$ be the makespan of the initial assignment.

**Theorem 2.** *Assume re-assignments give priority to the old assignment, when this remains optimal. With the makespan in use, a chase terminates (with all targets captured).*

*Proof.* Let $M$ be the makespan, initialized to $M_0$. $M$ cannot increase from one iteration to another, as each agent follows its target. $M$ cannot increase during a re-assignment, as the re-assignment minimizes $M$.

We show that $M$ decreases (by at least 1) after a finite number of steps. Assume by contradiction that $M$ never decreases. It follows that $M$ does not decrease during a re-assignment, which further implies that the current assignment is preserved forever. However, after a finite number of steps, all targets have stayed put at least once, and therefore the distance between each agent and its target decreases by at least 1. This implies that $M$ decreases by at least 1, which is a contradiction with our assumption.

Thus, $M$ never increases and it decreases by at least 1 after a finite number of iterations. Applying this recursively, it follows that $M$ eventually converges to 0, which is equivalent to saying that all targets have been captured. □

**Corollary 1.** *If all targets stay put at the same iterations, a chase finishes in at most $M_0 \times l$ iterations.*

*Proof.* In the previous proof, $M$ decreases for sure at an iteration where all targets stay put. It follows that $M$ converges to 0 after at most $M_0 \times l$ iterations. □

**Mixed-Cost Criterion.** Minimizing the makespan alone can sometimes lead to a large solution summation cost. This can be alleviated with a mixed-cost criterion, where the makespan is the main component, and ties are broken on the summation cost. The following assignment algorithm handles such a case: 1) Find the minimum weight $w$ such that, after removing all edges with a larger weight than $w$, there still exists one perfect match; 2) Find the minimum weight bipartite match in the map after removing all edges with a larger weight than $w$.

The time complexity for step 1 is $O(n^{2.5} \log n)$, as discussed in the case of the makespan cost. For step 2, the complexity is the same as for the summation cost. Thus, step 2 dominates the complexity, and the overall complexity is the same as for the summation cost.

**Theorem 3.** *With the mixed cost in use, a chase terminates, reaching a state where all targets are captured.*

*Proof.* Given $P_1 = (M_1, S_1)$ and $P_2 = (M_2, S_2)$, we say that $P_1 < P_2$ if $M_1 < M_2$ or ($M_1 = M_2$ and $S_1 < S_2$). Let $P = (M, S)$ be the initial mixed cost, with $M$ being the makespan and $S$ the summation cost. $P$ never increases from an iteration to the next, as each agent chases its assigned target. $P$ never increases at a re-assignment, as every new assignment minimizes $P$. $P$ strictly decreases every time when a target stays put, as at least $S$ decreases. Hence $P$ converges to $(0, 0)$ after a finite number of iterations. $\square$

Corollary 1 applies to the mixed-cost case as well, providing an $M_0 \times l$ upper bound to the number of iterations. We omit the proof due to space limitations.

## 6 Static Assignment Error

If new assignments are computed at every iteration, the active assignment will always be statically optimal. However, if an assignment is kept active for a number of iterations, for computational time savings, an error might accumulate as compared to a statically optimal assignment. We analyse this formally, deriving upper bounds for such errors.

Recall that the time is discretized into iterations. We assume that $t = 0$ is the time at the beginning of the first iteration. Given a time $t$, let $d^t(i, j)$ be the distance on the map from agent $i$ to target $j$.

In the rest of this section we focus on the static error for the summation cost, and the static error for the makespan cost.

**Definition 1.** *At a given time $t$, the* summation static error *of an assignment $f$ is $E_s(f) = \sum_{a \in A}(d^t(a, f(a)) - d^t(a, f^*(a)))$, where $f^*$ is a statically optimal assignment for the summation cost at time $t$.*

**Theorem 4.** *Within a given iteration, if $\ell$ out of $|A|$ targets move, the summation static error of the active assignment $f$ cannot increase by more than $2\ell$.*

We omit the proofs to Theorems 4 and 5 to save space.

**Corollary 2.** *Consider a statically optimal assignment $f$ at a starting time $0$. If $f$ remains active in the next $p$ iterations, $E_s(f) \leq 2p|A|$ after $p$ iterations.*

Figure 2 shows an example. At iteration $k$, the active assignment $(A_1, T_1), (A_2, T_2)$ is statically optimal with respect to the summation cost. However, at iteration $k + 1$, its summation cost is $4 + 4 = 8$, whereas the the assignment $(A_1, T_2), (A_2, T_1)$ has a summation cost of $2 + 2 = 4$.

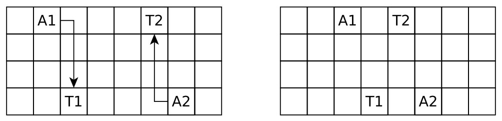

Figure 2: Illustrating the static error on a 4-connected gridmap. Left: iteration $k$. Right: iteration $k + 1$.

The static error of the initial assignment is $8 - 4 = 4$. According to Theorem 4, the maximum error at one iteration is $2\ell = 2 \times 2 = 4$. Thus, our example shows a worst-case scenario in terms of the static error for the summation cost.

**Definition 2.** *At a given time $t$, the* makespan static error *of an assignment $f$ is $E_m(f) = \max_{a \in A} d^t(a, f(a)) - \max_{a \in A} d^t(a, f^*(a))$, where $f^*$ is a statically optimal assignment for the makespan cost at time $t$.*

**Theorem 5.** *The makespan static error of an active assignment $f$ cannot increase at one iteration by more than 2.*

**Corollary 3.** *With the makespan cost in use, consider a statically optimal assignment $f$ at a starting time $0$. If $f$ remains active in the next $p$ iterations, the makespan static error $E_m(f)$ after $p$ iterations is bounded by $2p$.*

Consider again the example in Figure 2. At iteration $k$, the active assignment $(A_1, T_1), (A_2, T_2)$ is statically optimal makespan-wise. However, at iteration $k + 1$, its makespan is $\max(4, 4) = 4$, whereas the assignment $(A_1, T_2), (A_2, T_1)$ has a makespan of $\max(2, 2) = 2$. The static error of the initial assignment is $4 - 2 = 2$. According to Theorem 5, the maximum error at one iteration is 2. Thus, our example shows a worst-case static error for the makespan.

## 7 Experimental Results

We evaluate four agent strategies. The DIS strategy performs assignments based on the summation cost. MKS and MIX are based on the makespan cost and the mixed-cost criterion, respectively. The fourth strategy, GDY, is a baseline, greedy approach that iterates through the set of agents and assigns to an agent the closest target not yet assigned.

We implemented three target strategies. TrailMax [Moldenhauer and Sturtevant, 2009a] aims at planning a path for the target that will provably not intersect with a given agent's path for $k$ time steps, where $k$ is a parameter. We use $k = 50$. The Escape strategy maximizes the minimum distance to any agent. Naive is a baseline, random walking strategy.

Experiments use maps from Sturtevant's [2012] repository: AR0603SR, from Baldur's Gate (BG), with 13,764 nodes; AR0703SR (BG, 51,585 nodes); orz100d (Dragon Age: Origins – DAO, 99,626 nodes); orz900d (DAO, 96,603 nodes); deadwaterdrop (Warcraft III – WC3, 76,028 nodes); and darkforest (WC3, 99,758 nodes).

Besides the agent strategies, target strategies and maps described earlier, we have implemented an optimal agent strategy and an optimal target strategy, which we evaluate on 2 small maps where such optimal strategies are practical.

Following the approach of Sun *et al.* [2012], maps are considered to be 4-connected. Unless said otherwise, numbers

| Target strategy → | TrMax | Esc. | Naive | TrMax | Esc. | Naive |
|---|---|---|---|---|---|---|
| Map | Number of steps | | | Number of iterations | | |
| AR0603SR | 6.11% | 0.63% | 1.03% | 3.84% | 5.71% | 5.77% |
| AR0700SR | 11.69% | 1.37% | 2.02% | 10.47% | 2.64% | 12.48% |
| darkforest | 4.42% | 2.03% | 1.65% | 4.09% | 4.84% | 7.54% |
| deadwaterdrop | 5.34% | 1.42% | 0.70% | 4.71% | 1.89% | 9.08% |
| orz100d | 3.10% | 0.55% | 1.16% | 5.61% | 5.85% | 6.29% |
| orz900d | 1.51% | 0.34% | 0.50% | 4.81% | 4.33% | 6.71% |

Table 1: The impact of the re-assignment gap on the solution quality, for the MIX agent strategy.

shown in figures and tables are averages over 100 instances with initial agent and target locations picked at random.

**Evaluating the Agent and the Target Strategies.** Figure 3 compares the 4 agent strategies in combination with the three target strategies. To save room, we show results for a representative subset of data, with three maps and with assignments computed at every iteration.

The main conclusions, shown next, are consistent across all maps. MIX combines strengths from both DIS and MKS, and this helps MIX be the best strategy in terms of iterations (Figure 3, bottom). In terms of number of steps, MIX and DIS have a good performance, being close to each other and significantly better than MKS and GDY (Figure 3, top).

We conclude that MIX is overall the best, which is why our subsequent analysis focuses more on MIX. DIS has a good performance as well, whereas MKS is weaker. In terms of number of agent steps, MKS is even outperformed by GDY.

Figure 3 shows that, among the target strategies, TrailMax performs the best, resulting in more iterations and agent steps as compared to Escape (2nd place) and Naive (3rd place). To our knowledge, TrailMax has not been tested in multi-agent scenarios before. Our results confirm that TrailMax is a state-of-the-art target strategy in multi-agent chases too.

**Sensitivity to the Re-assignment Gap.** By definition, the *re-assignment gap* is the number $g$ with the property that a new assignment is re-computed at every $g$-th iteration. E.g., when $g = 1$ an assignment is computed at every iteration. We have varied $g$ in the range $R = \{1, 2, 4, 8, 16, 32, 64, 128, \infty\}$. For $\infty$, no assignment is computed except for the initial one.

Table 1 summarizes the results for MIX. The number-of-steps values are computed as $100 \times (\mathbf{M} - \mathbf{m})/\mathbf{M}$, where $\mathbf{M} = \max_{g \in R} S(g)$, $\mathbf{m} = \min_{g \in R} S(g)$, and $S(g)$ is the number of steps with the re-assignment gap $g$ in use. The number-of-iterations numbers are computed similarly.

The results show that, when using MIX, the quality of solutions is stable when varying $g$. The number-of-steps values range from as little as 0.34% to 11.69%. The number-of-iterations values range from 2.64% to 12.48%. A majority of the numbers shown in the table are smaller than 5%.

In the next section we present an additional comparison of a few gap values on a range of numbers of agents and targets.

**Scalability to Large Instances.** For a scalability analysis, we vary the number of agents and the number of targets from 10 to 200 in increments of 10. We measure the CPU time to compute agent moves, the number of steps in a solution (summation cost) and the number of iterations (makespan). We focus on MIX, the best agent strategy, and TrailMax, the best target strategy and thus the most difficult opponent of MIX. We considered three gaps $g$: 1, 10 and $\infty$. MIX with gap 1 is up to 8.94 times slower than with gap 10. It has no benefits in terms of number of iterations, and the benefits on the number of steps are small, ranging from -1% to 7.92%. Thus, gap 10 works well in comparison to gap 1, and we skip gap 1 from this analysis, to save room and avoid clutter.

Figure 4 shows the performance on darkforest, the largest map used. Other maps show a similar behavior. Instances are solved very fast. For the largest instances (200x200), the average solving time ranges from 0.08 seconds (map deadwaterdrop) to 0.13 seconds (maps orz100d and orz900d) when $g = \infty$; and from 0.57 seconds (map darkforest) to 1.08 seconds (map orz900d) when $g = 10$.

On the other hand, previous work does not scale beyond small instances. Even Vieira *et al.*'s work [2009], which is suboptimal and somewhat more scalable than optimal centralized minimax, needs to perform $n \times n$ one-chase-one minimax searches before it starts performing any action. However, even one single one-chase-one minimax search is impractical on maps of a reasonable size, such as our benchmark maps. Their reported tests have up to 9 units (6 agents, 3 targets) on graphs with up to 16 nodes. Furthermore, Moldenhauer and Sturtevant [2009b] showed that a state-of-the-art minimax search for the basic 1-chase-1 case requires in the order of 1 million to over 10 million node expansions on a map as small as 1600 nodes.

In contrast, our maps are much larger, with up to nearly 100,000 nodes, and we have many more agents. Yet, our method solves such large problems within the order of 1 second or less. These results show an substantial boost in speed and scalability in comparison to previous work. To achieve such a scalability, we give away solution optimality. See an analysis of solution quality later in this section.

Figure 4 illustrates the trade-off between the computational time (left) and the solution quality (number of steps at the center and number of iterations at the right). Performing no re-assignments is faster, and it has a very small impact on the number of steps. The variation in the number of iterations is also reasonably small, not exceeding 14.74% in Figure 4 (right). Thus, if speed is more important and a moderate increase in the number of iterations is acceptable, use no re-assignments. Otherwise, perform re-assignments, which still allows to solve problems within the order of 1 second or less.

**Comparison to a Minimax Solver.** We have compared MIX to an optimal minimax solver on small instances where minimax search is feasible. The purpose of this experiment is evaluating the quality of solutions computed with MIX. We used $2 \times 2$ chases on two maps, one with 10 nodes, with a topology inspired from Vieira *et al.* [2009], and one with 28 nodes (called the "ijcai" map), which to our knowledge is created by Sturtevant. On each map, we solve 50 randomly-generated instances. On such small maps, MIX results are identical for $g = 1, 10, \infty$.

In a minimax search, agents minimize the number of iterations, and targets maximize it. A minimax search provides the number of iterations when both agents and targets play optimally. In addition, we use minimax search to build an

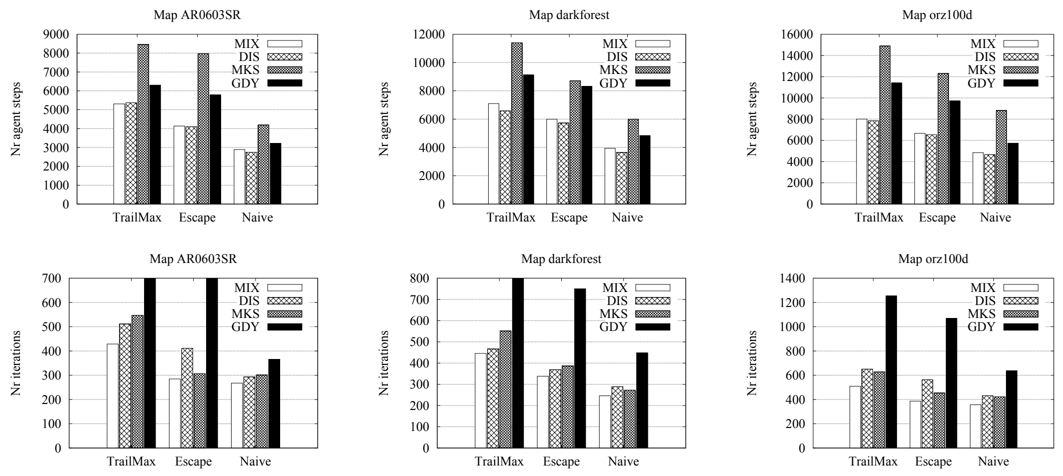

Figure 3: Summary comparison on instances with 40 agents and 40 targets.

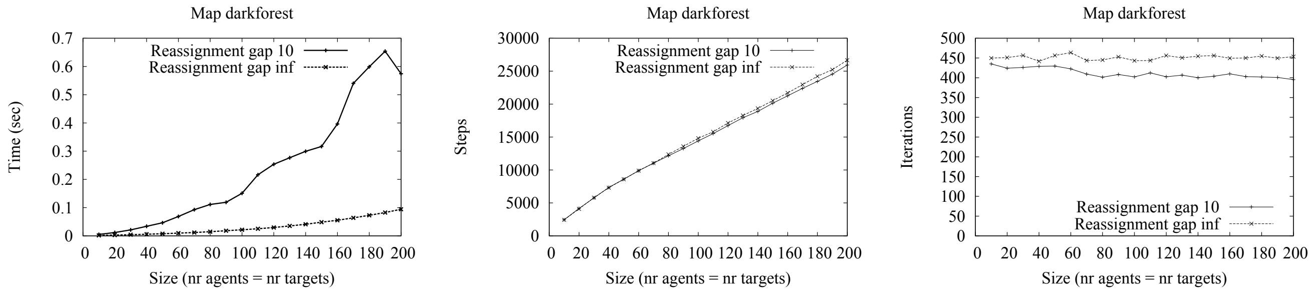

Figure 4: Scalability to increasing numbers of agents and targets.

| | 10-node map | 28-node map |
|---|---|---|
| Optimal agent strategy | 13.44 | 19.40 |
| MIX agent strategy | 15.42 | 21.44 |
| **Diff %** | **12.84%** | **9.51%** |

Table 2: Average number of iterations over 50 instances, when playing against an optimal target strategy.

optimal target policy in response to MIX agent moves.

Table 2 shows that MIX produces good-quality solutions, only about 2 iterations longer than optimal in average.

**Beyond $n \times n$ Scenarios.** To show that the system can handle "uneven" cases with $m \neq n$, we considered cases with $n_0 = 2m$, where $m$ is the number of agents, $n_0$ is the initial number of targets, and $m$ varies from 10 to 100 in increments of 10. At any given point in time, an agent is assigned to exactly one target, unless the remaining number of targets $n$ is smaller than $m$. In the latter case, each remaining target is assigned one agent. Initially, half of the targets (selected at random) are assigned to agents and the other half remain unassigned. When a target is captured, one unassigned target (selected at random), if any is left, changes its status into assigned, and a re-assignment is performed. An instance finishes when all targets are captured. As shown in Table 3, the system scales well in uneven scenarios as well.

| Map | CPU time | Nr assignments |
|---|---|---|
| AR0603SR | 0.91 | 74.22 |
| AR0700SR | 1.25 | 79.96 |
| orz100d | 1.24 | 75.51 |
| orz900d | 1.21 | 95.08 |
| deadwaterdrop | 0.58 | 66.88 |
| darkforest | 0.70 | 70.89 |

Table 3: Summary statistics for the *largest* (100x200) uneven instances. Numbers are averages over 100 instances.

## 8 Conclusion and Future Work

We have introduced a fast and scalable approach to moving target search with multiple agents and multiple targets. We have presented a thorough empirical evaluation and a theoretical analysis. Trading optimality for scalability, our system shows a strong speed and scalability performance. While previous techniques are impractical even for moderate-size problems, our method solves large instances on real game maps with hundreds of agents within the order of 1 second or less. On small instances, where a comparison to minimax search is possible, we find that our system produces solutions of a good quality. In future work, we plan to address problems with physical constraints such as velocity and size constraints.

## References

[Baier *et al.*, 2014] Jorge A. Baier, Adi Botea, Daniel Harabor, and Carlos Hernández. A Fast Algorithm for Catching a Prey Quickly in Known and Partially Known Game Maps. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2):193–199, 2014.

[Botea *et al.*, 2013] Adi Botea, Jorge A. Baier, Daniel Harabor, and Carlos Hernández. Moving Target Search with Compressed Path Databases. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS*, 2013.

[Botea, 2011] Adi Botea. Ultra-Fast Optimal Pathfinding without Runtime Search. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*, pages 122–127, 2011.

[Burkard *et al.*, 2009] R.E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.

[Edmonds and Karp, 1972] Jack Edmonds and Richard M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, 1972.

[Goldenberg *et al.*, 2003] Mark Goldenberg, Alexander Kovarsky, Xiaomeng Wu, and Jonathan Schaeffer. Multiple Agents Moving Target Search. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1536–1538, 2003.

[Hahn and MacGillivray, 2006] Gea Hahn and Gary MacGillivray. A Note on k-Cop, l-Robber Games on Graphs. *Discrete Mathematics*, 306(19–20):2492–2497, 2006.

[Hopcroft and Karp, 1973] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[Isaza *et al.*, 2008] Alejandro Isaza, Jieshan Lu, Vadim Bulitko, and Russell Greiner. A Cover-Based Approach to Multi-Agent Moving Target Pursuit. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE*, pages 54–59, 2008.

[Ishida and Korf, 1991] Toru Ishida and Richard E. Korf. Moving Target Search. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 204–210, 1991.

[Koenig *et al.*, 2007] Sven Koenig, Maxim Likhachev, and Xiaoxun Sun. Speeding up Moving-Target Search. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2007.

[Moldenhauer and Sturtevant, 2009a] C. Moldenhauer and N.R. Sturtevant. Evaluating Strategies for Running from the Cops. *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 584–589, 2009.

[Moldenhauer and Sturtevant, 2009b] Carsten Moldenhauer and Nathan R. Sturtevant. Optimal Solutions for Moving Target Search. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 1249–1250, 2009.

[Munkres, 1957] James Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[Nussbaum and Yörükçü, 2015] Doron Nussbaum and Alper Yörükçü. Moving Target Search with Subgoal Graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 179–187, 2015.

[Sturtevant, 2012] N. Sturtevant. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[Sun *et al.*, 2010] Xiaoxun Sun, William Yeoh, and Sven Koenig. Moving Target D* Lite. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 67–74, 2010.

[Sun *et al.*, 2012] Xiaoxun Sun, William Yeoh, Tansel Uras, and Sven Koenig. Incremental ARA*: An Incremental Anytime Search Algorithm for Moving-Target Search. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS*, pages 243–251, 2012.

[Vidal *et al.*, 2002] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic Pursuit-Evasion Games: Theory, Implementation, and Experimental Evaluation. *Robotics and Automation, IEEE Transactions on*, 18(5):662–669, 2002.

[Vieira *et al.*, 2008] M.A.M. Vieira, R. Govindan, and G.S. Sukhatme. Optimal Policy in Discrete Pursuit-Evasion Games. Technical Report 08-900, Department of Computer Science, University of Southern California, 2008.

[Vieira *et al.*, 2009] Marcos A. M. Vieira, Ramesh Govindan, and Gaurav S. Sukhatme. Scalable and Practical Pursuit-Evasion. In *Second International Conference on Robot Communication and Coordination, ROBOCOMM*, March 2009.

[Xie *et al.*, 2015] Fan Xie, Adi Botea, and Akihiro Kishimoto. Heuristic-Aided Compressed Distance Databases. In *AAAI-15 Workshop on Planning, Search, and Optimization (PlanSOpt-15)*, 2015.