A Cover-Based Approach to Multi-Agent Moving Target Pursuit

Alejandro Isaza and **Jieshan Lu** and **Vadim Bulitko** and **Russell Greiner**
Department of Computing Science, University of Alberta
Edmonton, Alberta, T6G 2E8, CANADA
{isaza,jieshan,bulitko,greiner}@cs.ualberta.ca

Abstract

We explore the task of designing an efficient multi-agent system that is capable of capturing a single moving target, assuming that every agent knows the location of all agents on a fixed known graph. Many existing approaches are suboptimal as they do not *coordinate* multiple pursuers and are slow as they re-plan each time the target moves, which makes them fare poorly in *real-time* pursuit scenarios such as video games. We address these shortcomings by developing the concept of *cover set*, which leads to a measure that takes advantage of information about the position and speed of each agent. We first define cover set and then present an algorithm that uses cover to coordinate multiple pursuers. We compare the effectiveness of this algorithm against several classic and state-of-the-art pursuit algorithms, along several performance measures. We also compute the optimal pursuit policy for several small grids, and use the associated optimal scores as yardsticks for our analysis.

1 Introduction

The moving target pursuit (MTP) task arises in numerous domains, ranging from law enforcement to video games. MTP is challenging for several reasons. First, in real-time scenarios, the planning time per move is often severely limited; for example in video games, game designers impose strict limits to maintain a fluid frame-rate, usually on the order of milliseconds for all pathfinding units (Bulitko *et al.* 2008). Second, while the simple strategy of "go to the target's starting position and then repeat all of its moves" does guarantee a capture when the pursuers are faster than the target, we would prefer for our pursuers to *outsmart* the target instead of simply outrunning it. Such scenarios often arise in real-time strategy games like *Company of Heroes* (Relic Entertainment 2005) and *Supreme Commander* (Gas Powered Games 2007) when a squad of soldiers is facing faster opponents. Thus, unlike most literature on the subject, we will be tackling the task of chasing faster targets.

In addition to outsmarting the target, it is also essential for the pursuers to coordinate their behavior. As an illustration, consider the "round-table" MTP scenario shown in Figure 1 [left], where the two pursuers on the right are trying to catch the target on the left. If each pursuer planned in isolation, both might go clockwise, allowing the target to evade capture. The obvious *coordinated* solution is for the agents to split up and surround the target. Surprisingly, even such simple coordination is exhibited by few existing pursuit algorithms. Theoretically, one can achieve such coordination by running a mini-max search in the *joint* state/action space of *all* agents. Unfortunately, this approach does not scale well since the space grows exponentially with the number of agents. In a practical MTP scenario, such a strategy would be too slow to be effective.

This paper describes an approach that balances planning speed with the amount of coordination between pursuers to produce a practical, effective strategy. To do so, we introduce the intuitive concept of *cover set* as the set of states that the pursuers can "cover", that is, that any one pursuer can reach faster than the target. We then use the size of the cover set to heuristically guide a novel MTP algorithm, CRA. CRA computes the cover set for each pursuer independently but with respect to the current position of other pursuers, thereby avoiding the combinatorial explosion of the joint action space. Consequently, by trying to maximize their individual cover set sequentially, the pursuers automatically coordinate their moves.

Our goal is to develop an efficient way for the pursuers to cooperatively reduce the target's mobility; this differs from many existing algorithms that are instead based on finding the shortest path to the target. To further increase the effectiveness of this approach, we augment it with abstraction and a way to explicitly deal with risk, leading to our Cover with Risk and Abstraction (CRA) algorithm. Moreover, although this paper uses grid-worlds for the empirical evaluation, this system is also applicable to any finite graph that is undirected, fixed and known.

Section 2 formalizes our task and defines performance measures. Section 3 examines work related to our task. Section 4 introduces the cover heuristic, which is implemented within the new CRA algorithm presented in Section 5. Section 6 evaluates CRA by empirically comparing its performance to classic and state-of-the-art pursuit algorithms chasing state-of-the-art targets. We further quantify their performances by comparing them to the *optimal* pursuer (computed off-line), which is another contribution of this research.

2 Problem Formulation

We assume there are $n \in \mathbb{N}$ pursuers, all trying to catch a single target, all $n + 1$ agents moving within a finite, fixed, known undirected graph. The vertices of the graph are called states, and edges are called actions, with edge weights that represent travel *distances*. Any number of pursuers can oc-

cupy a single state; the game is over if a pursuer occupies the target's state. Traversing an edge takes an amount of time that depends directly on the edge weight and inversely on the agent's speed. All agents (both target and pursuers) know each other's positions at all times. Furthermore, communication is free and the pursuers can share information instantaneously. The environment is deterministic.

Each agent interleaves planning and moving, and cannot plan while moving. The simulation proceeds in the following pseudo-parallel fashion: at simulation time t, an agent plans its action, taking p seconds of CPU time. The agent then takes an action, requiring $m = d/s$ seconds of simulation time, where d is the edge weight and s is the speed of the agent. The simulation time at which this agent will be allowed to plan again is $t + \tau p + m$ where τ is the "planning penalty" (a simulation parameter). Between times t and $t + \tau p + m$, the other agents are allowed to plan and move in a similar fashion. This approach allows modeling the kind of asynchronous chase scenarios that are common in both real-life and video games. Notice that we can obtain the turn-based synchronous environment common in the literature on MTP (Ishida 1997; Koenig, Likhachev, & Sun 2007) by setting $\tau = 0$, $s = 1$ and $d = 1$.

For a given graph environment, a problem instance is defined by the starting positions and speeds of all the agents. The agents traverse the graph until one of the pursuers occupies the same location as the target (*interception*), or until a pre-specified amount of time elapses (*timeout*). The percentage of problems where the target was intercepted successfully within the time limit is called the *interception rate*. The pursuers try to catch the target and minimize the *interception time*, defined as the earliest simulation time at which the target's position is the same a pursuer's position.

For the empirical evaluation, we will assume that the search graph is a two-dimensional rectangular grid whose cells are either vacant or occupied by obstacles. An agent can visit any vacant grid cell. In Figure 1 [left], the actions for each agent are to stay (with cost $d = 1$), or to move in one of the four cardinal directions, each with cost $d = 1$, provided the destination cell is not occupied by an obstacle.

3 Related Research

Moving Target Search (MTS) extends Learning Real-Time A* (LRTA*) (Korf 1990) to deal with *moving* targets (Ishida & Korf 1991). It is provably complete, assuming that the pursuer is faster than the target or the target makes suboptimal moves, and that the location of the target is always known. An MTS pursuer exploits pairwise distance heuristics to guide the search, and interleaves planning and execution. Moreover, MTS learns, as it improves its heuristic values incrementally based on its experience. MTS achieves real-time performance by using a constant, one-step lookahead horizon, and is thus *reactive*. However, MTS pursuers can perform poorly due to their myopic planning, which leads to *thrashing* behavior and *loss of information* (Melax 1993) when the target moves to a new location. To address these issues, Ishida extended MTS with commitment (to restrict the pursuer to a committed goal and prevent loss of information) and deliberation (to trigger an offline-search in response to detecting a local minima). However, even the improved MTS is still too slow to use in realistically-sized situations.

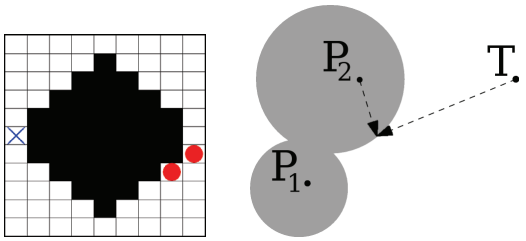

Figure 1: **Left:** A round-table map. The circles are the pursuers and $\times$ is the target. The best strategy here is for the pursuers to split; one going clockwise and the other counter-clockwise. **Right:** Cover for three agents: a target T with speed 1; pursuer P_1 with speed 0.2 and pursuer P_2 with speed 0.4. The dashed line represents the trajectory pursuer P_2 would take, given T's trajectory, to stop T from getting though between the pursuers.

To speed up convergence, MT-Adaptive A* adapts heuristics learned from the previous location of the target to the target's new location (Koenig, Likhachev, & Sun 2007). Koenig et al. demonstrate that MT-Adaptive A* is significantly faster than both MTS and D* Lite, an incremental search algorithm (Koenig & Likhachev 2002). Like Adaptive A* (Koenig & Likhachev 2005), MT-Adaptive A* updates heuristics using a full A* search between the current state of the pursuer and the current state of target. The pursuer, however, requires more than a constant time per move, and thus loses real-time performance.

As each of the aforementioned MTS-based approaches is designed for a *single* pursuer, they will not coordinate multiple pursuers, instead they would drive *all* of the pursuers toward the target in a way that would probably allow the target to easily escape. By contrast, we expect that an effective MTP algorithm should instead limit the target's mobility by "surrounding" it.

Multiple Agent Moving Target Search (MAMTS) extends existing real-time search agents by coordinating the multiple pursuers (Goldenberg *et al.* 2003). MAMTS first groups agents by a visibility chain, and then applies a single-agent real-time search algorithm (*e.g.,* greedy or minimax) using inferences about the target and the team of pursuers. Each agent maintains a *belief set* about all possible locations of both target and pursuers, and then selects a sub-goal from its own belief set. MAMTS is the first real-time algorithm that coordinates pursuers to limit the target's mobility. However, it does not exhibit the obvious "surrounding" strategy because the pursuers only coordinate when the target is out of sight (as in Figure 1 [left]). When the target is in sight they switch to the shortest-path search and ignore other pursuers, thus losing the ability to minimize the target's mobility.

4 The Cover Heuristic

To motivate the notion of *cover*, consider the task of determining whether a target can pass between two pursuers before they close the gap between them, assuming that the target will travel in a straight line. If we know which states the

pursuers can reach faster than the target, then we can simply send the pursuer to the interception; see Figure 1 [right].

This inspires the notion of *cover set*, which is the set of all states that any pursuer can reach before the target; and its complement, a *target-cover set*. For each state s_d, suppose that the target takes the fastest path $S_d = \langle s_1, ..., s_d \rangle$ of distance d to reach state s_d at time t_d. For each state s_d, if there is a pursuer that can reach s_d in time $t'_d \leq t_d$ then s_d belongs to the cover set. We will refer to the number of states in the cover set as *cover*.

However, while the cover set is useful the case of Figure 1 [right], it is not provably sufficient for capture, as the target might not maintain a straight line trajectory. Nonetheless, the concept of cover gives heuristic information about the "catchability" of the target, given the states of target and pursuers. For example, if the pursuers are slow, then the cover will be small; if the target is cornered, then the cover will be large.

A More Practical Approximation. Cover, as defined above, is difficult to compute. Instead, we use the following (re)definition for the target-cover set: A state s is target-covered if (1) s is the target's current location, or (2) s is adjacent to a state s' in the target-cover set, and the target can reach s faster than any pursuer. This suggests the algorithm appearing in Figure 2, which performs a breath-first expansion of the cover and target-cover sets until the two fronts meet, at which point the expansion stops. The time complexity of the algorithm is linear in the total number of states N.

```
calculateCover
 1  for each agent a (with speed s(a) and type t(a) at location l ) do
 2    add [location l, agent a, time t = 0]
      to the priority queue q
 3  end for
 4  while q is not empty do
 5    pop the element e = [l, a, t] with smallest t from q
 6    if l is marked then continue
 7    if t(a) is pursuer then
 8      mark l as covered
 9    else
10      mark l as target-covered
11    end if
12    for each neighbour l' of l at distance d do
13      if l' is not marked then
14        add [l', a, t + d/s(a)] to q
15      end if
16    end for
17  end while
```

Figure 2: Cover calculation algorithm.

5 CRA: Cover with Risk and Abstraction

A simple pursuit algorithm will greedily take the action that maximizes the cover: on every move, each pursuer sequentially computes its cover for each possible successor state and picks the action that leads to the state with the highest cover value. Notice that this is done on a per-agent basis, without explicitly considering joint moves; yet the positions of other pursuers are taken into account implicitly in the cover computation for each pursuer.

While this algorithm gives good results in some cases, it has some drawbacks. First, this naive approach is slow due to the overhead of computing cover. Another problem is tie-breaking, especially if all possible moves have the same cover value because the entire area is already covered. This causes the pursuers to make uninformed moves. Finally, the action that maximizes cover often corresponds to holding back as, essentially, one pursuer waits for another pursuer to catch up; however, there are situations where one of the pursuers should rush in for a capture. In light of the drawbacks of this naive approach, our cover-based algorithm, CRA, includes two enhancements: abstraction and risk.

5.1 State Abstraction

State aggregation is a technique that represents a set of states that possess similar characteristics as a single conceptual state. It can be used to build abstractions, which have fewer states but effectively approximate the structure of the original state space. Once an abstraction has been built, an abstract solution can be found more quickly by searching in the abstract space; this abstract solution can subsequently be refined back into the original space. To speed up CRA, we take advantage of an abstraction scheme particularly suited for grid worlds, *clique abstraction*, which builds abstract states by aggregating fully connected components of the original state space (*i.e.,* cliques) (Sturtevant & Buro 2005).

CRA builds a hierarchy of progressively smaller clique abstractions of the original search space, stopping when the abstraction consists of a single state. In grid worlds, each abstraction level reduces the number of states by a factor of at most 2 because the clusters are of size 2 in a 4-connected grid. CRA computes the approximate cover heuristic at level ℓ as this is $O(2^\ell)$ times faster than in the original space. Unfortunately, the greedy cover-maximizing algorithm will produce only an *abstract* action for each pursuer. To determine the proper ground-level action, CRA uses a standard refinement process (Sturtevant & Buro 2005): a path in the abstract space defines a *corridor*, which is a set of states whose abstract parents lie on the abstract path. We can then use A* search (Hart, Nilsson, & Raphael 1968) restricted to the states within the corridor to produce a path at the lower level of abstraction. The refinement is then repeated with that path until we return to the original space ($\ell = 0$).

5.2 Risk

As presented thus far, CRA has a notable weakness; when the pursuers are too few in number to completely surround the target (which occurs frequently in large, empty maps), their goal of minimizing the target's mobility (that is, increasing cover) prevents them from moving in for the capture. We can address this problem by allowing the pursuers to decide between keeping the target surrounded and "rushing in" for the capture, *i.e.,* to decide on how much of a *risk* each pursuer should take.

The PRA* algorithm (Sturtevant & Buro 2005) implements this "rush-in" policy by sending a pursuer directly to the target's current position. PRA* exploits abstraction to speed up a single-agent's search by searching in an abstract

space, then doing *partial* refinement (as explained in Section 5.1). This means that little work is lost when the target moves because the pursuer has not computed the path all the way to the target.

We introduce a risk parameter $w \in [0, 1]$ to balance the trade-off between surrounding and rushing. A CRA($w = 0$) pursuer takes only greedy actions (which optimize the cover size score), using the PRA* criteria only to break ties. A CRA($w = 1$) pursuer uses PRA* — *i.e.,* it ignores cover. This means that a $w = 0$ pursuer would hold back to limit the target's mobility, while a $w = 1$ pursuer would rush in for a capture. For $w \in (0, 1)$, CRA(w) first computes the destination state using cover, and then compares its cover with the cover of the destination state produced by PRA*. If the ratio of these two covers is at most $1 - w$, CRA(w) selects PRA*'s move; otherwise it uses the maximal cover move. For example, CRA($w = 0.5$) would select the PRA* move whenever the cover at that position is at most half the cover of the maximum cover move.

Figure 3 shows the CRA algorithm for one pursuer, p. The method *refine* takes a proposed abstract successor state and a level ℓ, and returns a ground level action, as explained previously. The time complexity per move of the algorithm is $O(bN)$, where b is the number of (abstract) successors and N is the number of (abstract) states.

```
CRA(p, w, ℓ) : action
1   s_max ← the abstract successor of max. cover
2   s_PRA* ← the abstract successor given by PRA*
3   c_max ← the cover of s_max
4   c_PRA* ← the cover of s_PRA*
5   if c_PRA*/c_max > 1 − w then
6     return refine(s_max, ℓ)
7   else
8     return refine(s_PRA*, ℓ)
9   end if
```

Figure 3: The CRA algorithm.

6 Empirical Evaluation

We developed a testbed that can use homemade maps as well as maps inspired by commercial video games such as Baldur's Gate (Bioware 1999). It supports asynchronous and the synchronous unit simulation as described in Section 2. We also developed an optimal solver that, given a target algorithm, finds the optimal policy for the pursuers. While this solver is not viable for practical applications due to its large computational cost, it is useful as a baseline when comparing the various algorithms.

In our experiments, we used small, homemade maps such as the one on Figure 1 [left], others with sizes ranging from 100 to 400 states, and two larger maps from commercial games (similar to those used in other studies, (Bulitko & Sturtevant 2006)). For the small maps, we averaged the results from 20 random starting locations; for the large maps, we used 100 random starting locations. Each simulation terminated whenever either any pursuer captured the target (*i.e.,* occupied the same position), or after 1000 simulation-time steps. We report the *interception rate* as the number of captures divided by the total number of runs.

6.1 Target algorithms

Simple Flee (SF). The target randomly distributes 20 beacons on the map at the onset of a simulation. It then selects the beacon that is furthest from the pursuers and runs PRA* to that beacon. It then executes the path for five moves before selecting the furthest beacon again.
Dynamic Abstract Minimax (DAM). The target runs minimax starting at the topmost level of abstraction (Bulitko & Sturtevant 2006). If minimax returns a move in which the target evades capture, then that move is refined and used. If minimax determines that there is no escape, then the abstraction level is decreased and the process starts over.
Minimax. The target runs Minimax with alpha-beta pruning with cumulative distance as the evaluation function, to a depth of 5.
Greedy. The target runs the standard greedy algorithm guided either by cumulative Manhattan distance to the pursuers (named GrM on the plots) or by the target-cover heuristic (named GrC on the plots).

6.2 Pursuit algorithms

Perfect pursuit policy. It is computed off-line for each target algorithm using dynamic programming techniques. The exponential complexity of computation makes the approach tractable only for small maps.
Cover with Risk and Abstraction (CRA). We use $w = 0$ for all studies except our investigation of the effects of risk.
Moving target Search (MTS). As described in Section 3 with a degree of commitment and degree of deliberation both with a value of 5 for small maps, and 10 for large maps.
Greedy. The standard greedy algorithm guided either by cumulative distance to the pursuers (named GrM on the plots) or by cover heuristic (named GrC on the plots).
Minimax. The set of pursuers run Minimax with alpha-beta pruning with cumulative distance as evaluation function, to a depth of 5 ply.
Partial Refinement A* (PRA*). Each pursuer runs PRA* to the target's position, oblivious of the other pursuers.

6.3 Studies

Overall performance. Each of these first set of studies involved two pursuers (running one of the six pursuit algorithms), chasing a single target of equal speed (running one of the five target algorithms). The maps were small enough to allow us to compute an optimal pursuit policy, which leads to a successful capture under the available time, for each of these maps and for each starting position. The interception rate indicates the effectiveness of the pursuer algorithms; see Figure 4, where each bar is the mean over 300 problems (3 maps, 5 targets, 20 problems each). Note the two pursuers sometimes start in different locations (Figure 4 [left]) and sometimes in a single location (Figure 4 [right]). In either case, CRA dominates the competition. Additionally, pursuers that coordinate their pursuit (CRA and GrC) fare better when started in the same location than uncoordinated contestants (e.g., MTS and PRA*).

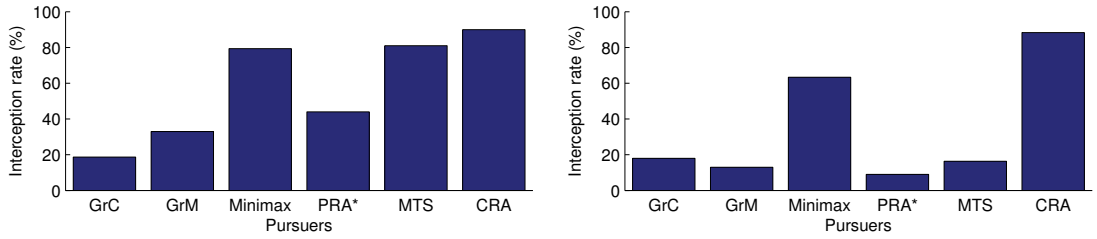

Figure 4: Interception ratio for dispersed (left) or aggregated (right) starting positions.

The interception time is another relevant measure, but as the algorithms do not achieve a 100% success rate, comparing the interception times for different problems is problematic. Also, comparing only the problems that all algorithms could solve biases the result towards easy problems. Figure 5 presents the average optimal time for successful runs along with the average algorithm time. These times show that the low-success algorithms are solving easy problems since the average *optimal* capture time is low for those problems. For instance, greedy with cover heuristic solved 66 problems whose average optimal interception time is only 8.29 steps.

Algorithm	Problems solved	Optimal time	Alg. time
Optimal	300	10.60	10.60
CRA	253	10.32	16.13
MTS	234	9.63	65.19
Minimax	211	9.58	37.60
PRA*	148	8.70	18.96
GrM	112	8.88	13.63
GrC	66	8.29	10.32

Figure 5: Number of problems solved by this algorithm, the average interception times by *optimal* pursuers, and average interception times by this algorithm.

Performance improvement with the number of agents. The pursuers that try to minimize the target's mobility have a clear advantage when there are many agents. As an illustration, consider a round table map (Figure 1 [left]). A pursuit algorithm that does not coordinate its pursuers will not take advantage of additional pursuers; this is especially problematic if the pursuers all start in the same location. This is clearly true for PRA*; Figure 6 shows that its performance curve is flat. On the other hand, GrM, which greedily selects *joint* moves using cumulative Manhattan distance, shows moderate improvement. CRA demonstrates significant improvement with the less abstracted version CRA($\ell = 1$), slowly reaching better performance than the more abstracted CRA($\ell = 3$). It is clear that, for these graphs, having more than 10 pursuers gives no further improvement in terms of success rate. In terms of capture time, both CRA($\ell = 1$) and CRA($\ell = 3$) converge after 200 steps with no further improvement as the map is saturated at this point.

Effects of abstraction. Increasing CRA's level of abstraction leads to the use of the cover heuristic in a smaller abstract space. Consequently, as the level of abstraction goes up, the pursuers will more frequently find themselves in the same abstract state as the target. Once that occurs, CRA invokes PRA* which uses the Manhattan distance heuristic and ignores the cover considerations entirely. Thus, CRA effectively becomes PRA* more frequently with higher levels of abstraction. Figure 7 considers 10 pursuers, each moving with a speed of 30% of the target's.

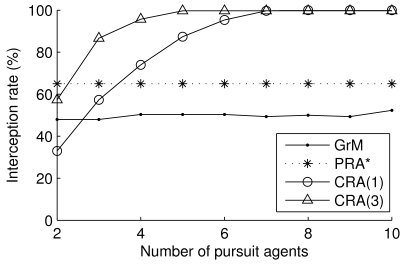

Figure 6: Interception rate improvement with additional pursuers. The results are averaged over three different target algorithms: SF, GrC and GrM. Each pursuer moves at 30% the speed of the target.

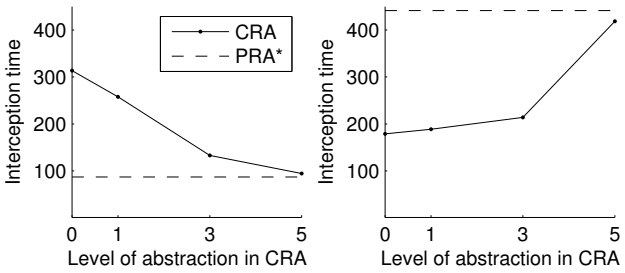

Figure 7: Effects of abstraction on CRA performance when the pursuers start out dispersed (left) or aggregated (right).

Effects of risk. We ran two studies to illustrate the effect of the risk parameter. The first analyzed the effects of map size with respect to a fixed target algorithm, DAM. The second analyzed the effects of different target algorithms over a fixed map. Again, we used two pursuers and one target, all having the same speed. We report the success rate of 100 runs with random starting positions.

For the first study we used five empty maps and two round-table maps of different sizes with random start states. Figure 8 [left] shows that a small value of risk works best for most of these problems. For the empty maps, all risk values between 0.1 and 0.6 have success rate of essentially 100%. For round-table maps the behavior is not as clear-cut, but the largest success rate is achieved near risk 0.1 for both maps. Notice that the success rate drops in all cases when the risk

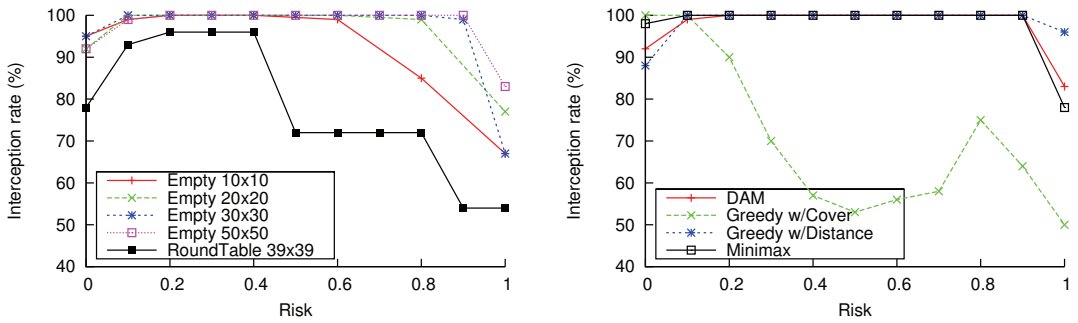

Figure 8: The success rate for different risk values for several maps (left) and for different target algorithms (right).

is 1 (which is equivalent to running PRA*).

For the second study we compared the different target algorithms over an empty 50×50 grid. Figure 8 [right] shows that using small values of risk is beneficial here as well — again risk values between 0.1 and 0.6 lead to the highest success rate. The low success rate when chasing Greedy with cover (GrC) suggests that the cover heuristic is very effective for *targets* as well.

These two studies suggest that a risk value of 0.1 can be a good choice for many problems. With the ability to tune the risk parameter, the results shown above may be further improved, possibly making CRA an even more superior winner over the other pursuit algorithms.

7 Future work

Our analyses above assume that *all* of the pursuit agents are using the same algorithms and same parameters; it would be interesting to mix agents that use different algorithms, or the same algorithm with different parameters for each agent — perhaps having several risk $w = 0$ pursuers surround a target while a different risk $w = 1$ pursuer goes in for the capture.

We analyzed how multiple *pursuers* can cooperate; another interesting problem is multiple-*target* cooperation. For example, the targets might decide to spread out to make it more difficult for the pursuers to surround them. Or perhaps the faster targets might be able to protect the slower targets, as seen in nature. Finally, we could consider using *cover* on different types of problems, such as *surrounding* instead of capturing.

Another interesting problem is dealing with unknown environments and targets which are not always visible. For the pursuers to be able to surround the target, they must first explore the environment in a distributed way. Similarly, if the target position is not known at all times, a different approach may be required.

8 Conclusions

This paper has discussed the challenges of designing an effective way for multiple pursuers to capture a single moving target. To address problems with existing approaches, (including run-time inefficiency and lack of coordination), we presented a new algorithm, CRA, that uses "cover" to overcome these limitations, and incorporates abstractions and risk to work effectively. We conducted a number of empirical studies, comparing CRA against five other classic and state-of-the-art pursuit algorithms chasing four types of targets. We computed the optimal policy for several small grids to serve as a yardstick in our comparisons, and demonstrated that CRA works effectively over a wide range of situations.

References

Bioware. 1999. Baldur's Gate. http://www.bioware.com/games/baldur_gate.

Bulitko, V., and Sturtevant, N. 2006. State abstraction for real-time moving target pursuit: A pilot study. In *AAAI Workshop: Learning For Search*, 72–79.

Bulitko, V.; Luštrek, M.; Schaeffer, J.; Björnsson, Y.; and Sigmundarson, S. 2008. Dynamic Control in Real-Time Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)* 32:419 – 452.

Gas Powered Games. 2007. Supreme Commanders. http://www.supremecommander.com.

Goldenberg, M.; Kovarksy, A.; Wu, X.; and Schaeffer, J. 2003. Multiple agents moving target search. *IJCAI*.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE SSC* 4.

Ishida, T., and Korf, R. E. 1991. Moving target search. In *IJCAI*.

Ishida, T. 1993. A moving target search algorithm and its performance improvement. *J. Japanese Soc. Art. Int.* 8.

Ishida, T. 1997. *Real-Time Search for Learning Autonomous Agents*. Kluwer Academic Publishers.

Koenig, S., and Likhachev, M. 2002. D* Lite. In *AAAI*.

Koenig, S., and Likhachev, M. 2005. Adaptive A*. In *AAMAS*.

Koenig, S.; Likhachev, M.; and Sun, X. 2007. Speeding up moving-target search. In *AAMAS*.

Korf, R. E. 1990. Real-time heuristic search. *AIJ* 42(2-3):189–211.

Melax, S. 1993. New approaches to moving target search. In *AAAI Fall Symposium*.

Relic Entertainment. 2005. Company of heroes. http://original.companyofheroesgam.com.

Sturtevant, N., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *AAAI Workshop: Learning For Search*.