# State Abstraction for Real-time Moving Target Pursuit: A Pilot Study

**Vadim Bulitko** and **Nathan Sturtevant**
Department of Computing Science, University of Alberta,
Edmonton, Alberta, Canada T6G 2E8
{bulitko | nathanst}@cs.ualberta.ca

### Abstract

The pursuit of moving targets in real-time environments such as computer games and robotics presents several challenges to situated agents. *A priori* unknown state spaces and the need to interleave acting and planning limits the applicability of traditional search, learning, and adversarial game-tree search methods. In this paper we build on the previous idea of hierarchical state abstraction, showing how it can be effectively applied to each of the problems in moving target pursuit. First, we develop new algorithms for both chasing agents and target agents that automatically build and refine a hierarchical state abstraction. We then demonstrate the improvements in performance that the state abstraction affords when applied to incremental A* search, learning real-time heuristic search, and minimax adversarial search. Finally, we propose and evaluate a systematic and efficient method of using single-agent techniques in adversarial domains. This leads to a diverse family of new moving target pursuit algorithms which draw on recent advances in single-agent search. In a preliminary empirical evaluation, we demonstrate effects of state abstraction on search and learning in the agents.

**Keywords:** Search, Machine Learning, Computer Games, Game-Tree Search

## Introduction

Pursuit tasks are a common occurrence in many domains. In computer games, for instance, hostile agents often pursue human-controlled agents. Even in cooperative settings, however, there is often a need for computer-controlled agents to follow other agents in the world. Pursuit tasks also occur in real-world domains, such as robo-soccer. Many people have also envisioned asking robots to perform similar tasks for uses such as tracking down criminals. In this paper we present new work and methodology for acting and learning in competitive pursuit tasks.

## Previous Research

In order to make a distinction between the moving-target search (MTS) algorithm (Ishida & Korf 1991) and the general problem to which it applies, we propose that this subfield be called Moving Target Pursuit (MTP).

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

### Moving Target Search

Previous research on moving target pursuit falls into two categories. The moving target search (MTS) family of algorithms (Ishida & Korf 1991; 1995; Ishida 1998; Shimbo 1999) learn shortest path distances between pairs of states on a map. The original MTS algorithm was an adaptation of learning real-time A* (LRTA*) (Korf 1990) for a moving target. MTS was shown to be complete in turn-based settings when the target periodically skips moves. It is, however, subject to "thrashing" in heuristic depressions (local minima) and "loss of information" when the target moves (Melax 1993a; 1993b). In response to these problems, a version of MTS extended with commitment and deliberation was proposed (Ishida 1993). The commitment mechanism has a hand-tuned parameter which allows the pursuit agent to commit to a target's location and minimize the "loss of information". The deliberation module switches to off-line search when the pursuit agent detects a heuristic depression. This was meant to reduce the number of heuristic function updates needed to correct the heuristic function. Not only is a manually tuned parameter required, but improved performance is not guaranteed (Melax 1993a; 1993b).

A second group of algorithms is based on learning a map as opposed to a heuristic function. Trailblazer (Chimura & Tokoro 1994; Edelkamp 1998) and Trailblazer with abstraction (Sasaki, Chimura, & Tokoro 1995) were designed to build a map of an *a priori* unknown environment, and then used to plan a path towards the target.

This research does not address several challenges specific to moving target pursuit in computer game-like settings. First, such environments are asynchronous insomuch as the pursuit and target agents do not take turns but operate simultaneously. Second, the torus grid worlds used with MTS are not used in applications. Third, target agents (such as human-controlled agents) can be more sophisticated than the basic agents (e.g., random) used in the academic research. Fourth, commercially used maps are not random and exhibit various patterns that can be used to a target agent's advantage. Fifth, most empirical results reported in the literature are limited to first-trial measurements and offer few insights as to what various efficient learning strategies are. Sixth, the environments are assumed to be fully known *a priori*. Finally, the pursuit agents studied simply do not scale

up to large maps. In the preliminary study presented in this paper, we address all of the aforementioned extra challenges with the exception of human-controlled target agents).

### Abstraction and Path-Refinement Search

Abstract state representation has often been used to help solve problems more quickly. Heuristics, which are essential for efficient search, are built from abstracted versions of original problem domains. Pattern Databases (Culberson & Schaeffer 1996), for instance, are built by abstracting a problem domain and then solving all possible instances of the abstract problem. There are two common approaches which build and use abstractions dynamically during search. The first approach is to perform computations in abstract space in order to improve heuristics (Holte, Grajkowski, & Tanner 2005) for the original problem space. An alternate approach is to directly refine the results of search in abstract space into a result in the original problem space. Partial-Refinement A* (PRA*) (Sturtevant & Buro 2005) is an example of refinement search. PRA* dynamically chooses an abstract level for path-finding and then successively refines a partial path which can subsequently be executed. PRA* is not only faster than A* when computing full paths, but is also more flexible due to the partial refinement. Partial-refinement spreads the cost of planning evenly over execution, because only the immediately required path is refined. This is particularly useful in dynamic environments where the the changing world may force an agent to re-plan.

Path-Refinement Learning Real-Time Search (PR-LRTS) (Bulitko, Sturtevant, & Kazakevich 2005) also uses path-refinement. PR-LRTS performs learning in an abstracted space and then uses A* to refine and execute paths in the real environment. The use of state abstraction speeds up learning greatly as a single learned heuristic value for abstract state affects agent's behavior on all its children at the ground level. Thus, learning is effectively generalized through the abstract space.

In order to illustrate path refinement as it is done in PR-LRTS and PRA*, we consider a micro-example from PR-LRTS in Figure 1. Subfigure (1) shows the ground state space below and one level of abstraction above. The agent must plan a path from A to B located at the ground level. First, the abstract parents of A and B, A' and B', are located. Then a learning search with the lookahead of three plans three steps in the abstract space (2). A "corridor" at the ground level comprised of children of the abstract path is then built (3). A set of ground-level children of the end of the abstract path are shown as C (4). Finally, A* is run within the corridor to find a path from A to C (5).

### Adversarial Search

Minimax with alpha-beta pruning is the most common algorithm used for play in two-player zero-sum games, however this approach is not generally used in more complicated domains such as Real-Time Strategy (RTS) games, a popular genre of computer games, or in MTP. This is because the actions in these domains are of a different granularity than the strategic decisions of the game. Any single action taken alone is fairly insignificant. What is important is the joint effect achieved by many actions.

Imagine using a robotic arm to play chess where the available actions are to move actuators in the arm. In theory there is no difference between planning for chess pieces or movements of the robotic arm, but it is unfeasible to use an algorithm like minimax to control a robotic arm directly. The robotic arm should be controlled by a separate single-agent search task, independent of the strategy being used to play a game of chess. Planning for MTP and RTS games using minimax would be somewhat analogous to using minimax to control this robotic arm. In this example, if we appropriately abstract the robotic control to a degree which returns the game to the level of the more abstract game of chess, the problem would once again become feasible. Similarly, adversarial search is not feasible in MTP unless we appropriately abstract the problem domain.

## Problem Description

Path-finding in computer games and robotics is subject to real-time constraints and limited CPU budget. In this paper we use an open-source environment used in (Bulitko, Sturtevant, & Kazakevich 2005) that enables setting up moving target pursuit problems on commercial game maps. Formally, the moving target pursuit problem is defined by a tuple $(G, s_p, s_t, a_p, a_t, r)$ where $G = (S, E)$ is a weighted graph representing the topology of the map, $a_p$ is the pursuit agent chasing the target agent $a_t$. In order to make our presentation easier to illustrate, in this paper we the graph $(S, E)$ is generated from a two-dimensional grid world where each square is connected to eight neighbors. Some cells are blocked with walls. An agent can move in eight directions (four cardinal and four diagonal). The travel (execution) cost of a straight move is 1, while diagonal moves cost $\sqrt{2}$. In our experiments, target agents have full knowledge of the map while the pursuit agents have to explore and learn the map. They see the status (blocked/available) of all grid squares within $r$ moves around their current position. Note that these settings are not a limitation of the algorithms discussed in this paper which can be applied to general search problems with explicit or implicit search spaces.

Like computer games and physical environments, and unlike turn-based testbeds used in previous research, this testbed is asynchronous insomuch as agents can move while other agents are deliberating. Agents are not allowed to move and deliberate at the same time. This puts slower thinking agents at a disadvantage as a less sophisticated but quicker agent can make more moves because of lower thinking costs. We assume that each agent can observe the location of the other agent, but only during deliberation. As a consequence, the other agent may be able to take several moves between observations. Unlike some previous research, we do not require the target agent to move more slowly than the pursuit agent. Thus, a fast-thinking target making perfect moves may be able to run away for an unlimited amount of time.

Each trial starts with the pursuit agent $a_p$ starting at the state $s_p$ and the target agent at the state $s_t$. *Initial distance* is

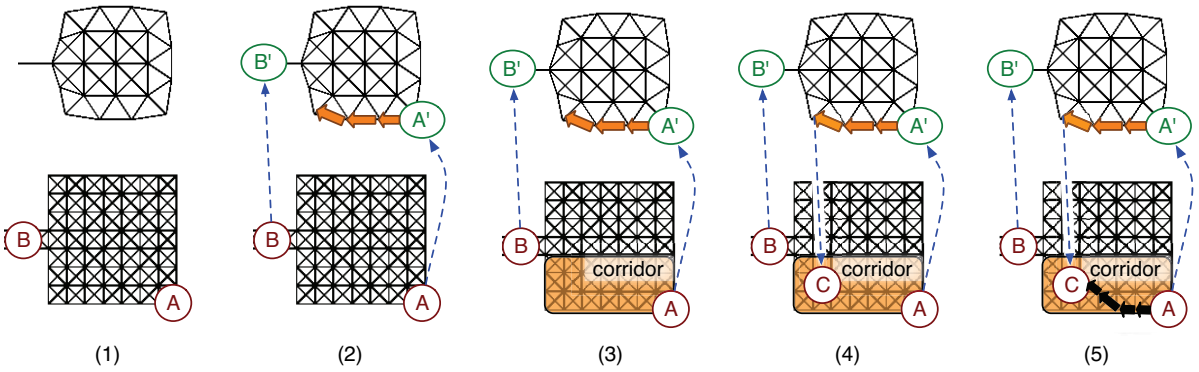


Figure 1: Path refinement with two levels of abstraction.

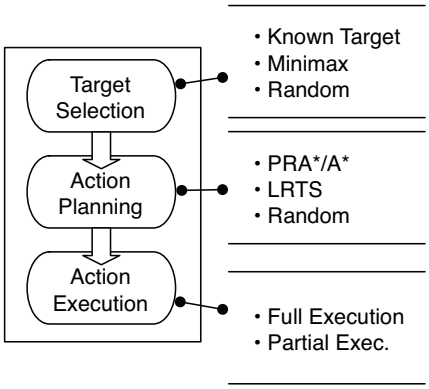


Figure 2: Unified agent design for both pursuit and target agents.

defined as the shortest path length between these two states. A trial continues until the pursuit agent catches the target agent. At this point, their locations are reset and a next trial begins. A trial is called final if the pursuit agent was able to catch the target agent without learning anything about the environment (i.e., without discovering anything new about the map and/or updating its heuristic values). A convergence run is a sequence of trials ending with the final trial. We report *convergence travel* as the total execution cost that a pursuit agent incurs on a convergence run. In this paper we do not report computational efforts expanded by the pursuit and target agents. Such data are less important in an asynchronous real-time simulation where computational efficiency is taken into account by the ability of a pursuit agent to catch its target. In the follow-up work we will experiment with other performance measures such as the travel distance after the learning process is complete.

## Agent Design

We use a unified design for all of our MTP agents, illustrated in Figure 2. This design consists of three stages. The first stage is 'target selection', which proposes a location to which the agent should travel. The second stage is 'action planning', in which the agent computes a path towards that location. In the final stage, 'action execution' the plan is either fully or partially executed before the process begins again.

For a pursuit agent which already knows the location of its target, the first stage can be quite simple, as such producing the location of the intended target. But, more complicated approaches are possible, such as predicting where the target is expected to travel, and taking the path which cuts the agent off. If a pursuit agent did not know the location of its target, this first module would be responsible for proposing locations to be explored.

In the case of an agent which is a target of another pursuit agent there are several more possible approaches for deciding where to travel next. Given enough computational resources, minimax could be applied to analyze the world completely. Except in exceedingly small worlds this approach is not computationally feasible. On the other end of the spectrum, an agent can randomly propose a neighboring node, which is exceedingly fast. A third approach is to suggest several possible target locations and decide between them using either heuristic information or other analysis.

Once an agent decides upon a target for travel in the world, it is a much simpler task to plan desired actions. A* will compute an optimal path from the current location to the target location, or PRA* can be used to calculate a partial path. Another approach is to use one of the family of real-time search algorithms such as LRTS (Bulitko & Lee 2006) for learning.

## Applying State Abstraction to MTP

The central contribution of this paper involves the question of how state abstraction can be used to improve MTP. Given a graph representation of an environment, there are many mechanisms through which state abstractions can be built. The idea is that, given an initial graph, a smaller graph which retains the general structure of the original graph is built. (Holte *et al.* 1996), for instance, abstracts nodes simply based on the neighbors within a fixed radius. (Botea, Müller, & Schaeffer 2004) overlay sectors onto a map to quickly abstract most of the space away. (Bulitko, Sturtevant, & Kazakevich 2005) abstract nodes based on local cliques in the graph. We use this approach, as their code and experimental framework is available for use. Instead of building

```
DAM
1  for each level of abstraction i = max_level ... 0
2     value, location =
         minimax(pursuit agent location, target agent location, i)
3     if value ≠ LOSS
4        return location
5     end if
6  end for
7  return CAPTURED
```

Figure 3: DAM pseudo-code.

a single abstract graph based on this method, a hierarchy of successively more abstract graphs are built, until eventually all connected components are abstracted into a single node.

We introduce here two novel uses of state-abstraction. First, we apply it to minimax search in the Dynamic Abstract Minimax algorithm, and then we apply it to Moving Target Search in the Path-Refinement Moving Target Search algorithm.

## Dynamic Abstract Minimax

Dynamic Abstract Minimax (DAM) is a strategy for the 'target selection' portion of an agent illustrated in Figure 2. The goal is to select an appropriate location in the world to which an agent can most safely flee. Given that there are 8 possible moves (including diagonals) from each state, even using alpha-beta pruning we will only be able to search a few ply before computational costs outweigh the benefits of any strategy produced. DAM does minimax analysis (using alpha-beta) in an abstract problem space, and uses those results to suggest an appropriate location to which the agent should flee.

In the state abstractions used, we have a hierarchy of abstractions. Given any two locations or nodes in the world we can successively find the parents of each node in the next level of abstraction, and eventually these nodes will share a parent. If agents are nearby, their first common parent in the abstraction will be low in the hierarchy. In general, the level in the abstraction at which two nodes share a parent will be correlated with the logarithm of the distance between the nodes. This is simple to show if we assume that each successive abstraction reduces the graph size by a constant factor greater than 1.

We could choose a fixed level of abstraction and always do an adversarial search at that level. But, there are problems with this approach. If we do not choose a sufficiently abstract level for planning, computational costs will still be large. If we choose too abstract a level for planning we will lose critical details about the map, especially when the pursuit agent gets close. Instead, we dynamically choose a level at which to do our abstract planning, and if that planning fails (i.e., we cannot avoid being captured), we re-plan in successively less abstracted levels. The pseudo-code for this process is in Figure 3.

The DAM algorithm works as follows: Given the location of the pursuit and target agents, the target agent performs a minimax search at the maximum level of abstract space. If the target agent cannot avoid being captured, the search is repeated at the next finer level of abstraction. This process

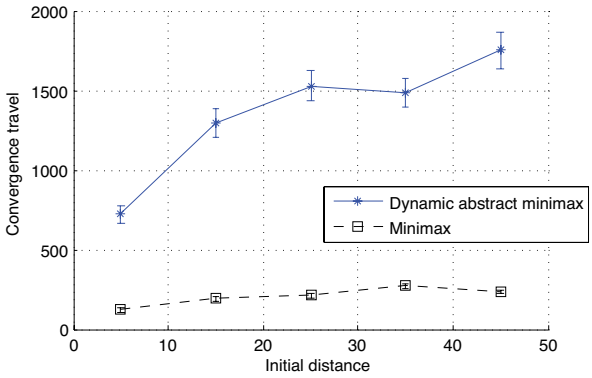


Figure 4: Advantages of dynamic abstraction in minimax search. Two target agents (DAM and minimax) are chased by PR MTS. DAM is substantially more difficult to catch. Each data point is averaged over 100 problems. Larger values are better.

repeats until the target agent can avoid capture. One way of thinking of this process is that it finds the best possible abstract map for search. An abstract map which results in immediate capture doesn't give enough information about the underlying map to be useful, while the full map is too detailed to extract high-level information. Thus, this process finds the most abstract map which is still useful for search.

The result of the iterative search process is an abstract leaf node; the abstract target to which the agent should travel. Given this node we choose a random child of this node in the actual map, and then use PRA* to compute a partial path to that node. After this path is executed, the process repeats.

We demonstrate the effectiveness of DAM over simple minimax search in Figure 4. This figure plots the time it takes for a PR MTS agent (described in the next section) to learn to catch a minimax and DAM agent. The DAM agent travels approximately five times farther than the simple minimax agent before being captured.

In these experiments both agents used the same fixed lookahead. While the DAM unit will have higher computational costs, these are compensate for because the unit is penalized for the time it spends thinking. Thus, units that think more effectively move more slowly. Given these parameters, DAM units are only a constant factor slower than minimax units, because they will repeat the refinement process during planning at most a fixed number of times. Thus, these results show that the information gained by using the abstractions is worth the extra computational effort.

## Path Refinement Moving Target Search

The design presented in Figure 2 allows us to take advantage of recent single agent search and learning techniques. In particular, most of learning real-time search algorithms can be extended to work in both the pursuit and the target agents in a fairly straightforward fashion. Specifically, a single argument heuristic function is used to estimate distance to goal in LRTA* (Korf 1990), weighted LRTA* (Shimbo & Ishida 2003), SLA*T (Shue, Li, & Zamani 2001), Koenig's LRTA* (Koenig 2004), $\gamma$-Trap (Bu-

litko 2004), LRTA*(k) (Hernández & Meseguer 2005), LRTS (Bulitko & Lee 2006), and PR-LRTS (Bulitko, Sturtevant, & Kazakevich 2005). By using a hash table, we can efficiently *represent* estimated distances between any two points on the map as in the original MTS (Ishida & Korf 1991). Our contributions help overcome difficulties with *learning* these distances uncovered by the original MTS algorithm.

Namely, we build on a recent learning real-time heuristic search algorithm called Path Refinement Learning Real Time Search (PR-LRTS) that has been shown successful in static environments (Bulitko, Sturtevant, & Kazakevich 2005). We extend its heuristic function to two arguments representing current positions of the target and pursuit agents. The resulting algorithm, called Path Refinement Moving Target Search (PR MTS) uses the LRTS$(d, \gamma, T)$ learning algorithm (Bulitko & Lee 2006) in an abstract space to come up with an abstract path of $d$ actions. This path defines a corridor in the ground space $G$ (Figure 1). A* is then run within the corridor to find a partial path towards the goal.

PR MTS takes advantage of state abstraction in four ways. First, it is able to build and refine a hierarchy of state abstractions automatically as it explores an *a priori* unknown map. Second, by learning in a smaller abstracted space not only do fewer heuristic values need to be learned, but each update to the heuristic function is also effectively *generalized* onto a cluster on neighboring ground level states. Third, the target's position at higher level of abstractions does not change as rapidly as it does at the ground level. Thus, fewer updates to the abstracted heuristic function are needed. This phenomenon was first reported in a study of MTS extended with a hand-tuned commitment-to-goal mechanism (Ishida 1992). Finally, path planning at the ground levels is performed by an A* algorithm. This application of A* is computationally efficient for two reasons: (i) A* is run within a narrow corridor derived from a path at a higher level of abstraction, and (ii) A* plans only a short path to the area corresponding to the end of the abstract path. On short paths the initial heuristic is more likely to be accurate. The impact of the improvements is shown in Figure 5.

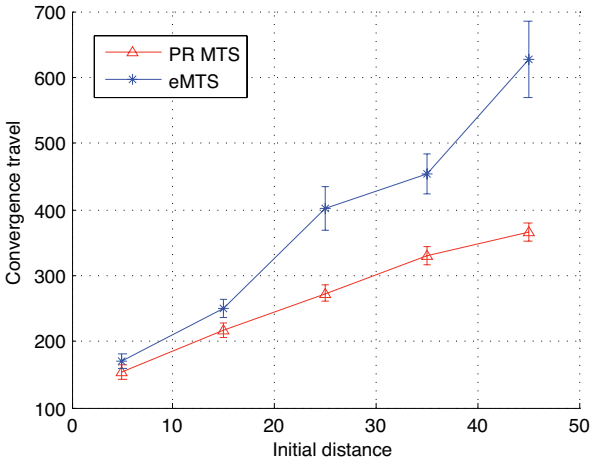


Figure 5: Effects of abstraction on convergence of moving target search. Two pursuit agents (extended MTS and path refinement MTS) are chasing a random agent. Path refinement MTS converges substantially faster. Each data point is averaged over 200 problems. Smaller values are better.

## Empirical Evaluation

We compared learning algorithms on 1,000 problems of varying difficulty randomly generated over 10 synthetic maps shown in Figure 6. The maps had between 400 and 1,349 reachable states. The initial distance between the start states of the target and pursuit agents ranged from 0 to 50 falling into five buckets: $(0, 10], (10, 20], \ldots$. There were 200 problems in each bucket (20 from each of the 10 maps). The following four pursuit agents were used:

1. **eMTS:** we used an extended version of the basic MTS algorithm (Ishida & Korf 1991). The extensions adapted from LRTS (Bulitko & Lee 2006) included: (i) a deeper lookahead of 10 ply, (ii) heuristic weight of 0.1, and (iii) a more aggressive max-of-min learning rule. These control parameters were tuned in a pilot study of 500 smaller problems. With these parameters, eMTS pursuing a random agent converged on average 34 times faster than the original MTS (i.e., with $d = 1, \gamma = 1.0$). Taking into account the undirected nature of our search graph $G$ (i.e., distance from A to B is the same as distance from B to A), we made our heuristic function symmetric so that updates to $h(a_p, a_t)$ were synchronized with updates to $h(a_t, a_p)$. Finally, unlike the original MTS, we did not update pursuit agent's heuristic function when the target moved. This is because in an asynchronous environment we cannot always observe our opponents movements.
2. **PR MTS** used the eMTS algorithm described above at the third abstract level. A* running in a corridor was used at the ground level to refine its path. This particular configuration was chosen via trial and error. Automated selection of abstraction level in PR MTS is a subject of future research.
3. **A*** at the ground level was used with the free space assumption (Koenig, Tovey, & Smirnov 2003). Such a pursuit agent would plan a complete path to the target's current position assuming that unknown areas of the map are not blocked. It would then follow the path until its end or until hitting a newly discovered wall. In either case, it would replan for the new target's position. Once the map is known, A* may perform poorly because it travels to where the target is currently located, although with random targets this isn't a huge problem.
4. **PRA*** with four-step partial refinement was taken directly from (Sturtevant & Buro 2005). This unit also moves towards the current location of the target, but re-plans after executing the refined path.

Each of these four pursuit agents was run against the following four target agents:

1. **random** agent moves on a straight line. At every move, there is a 20% chance of its changing its direction randomly. The change also occurs upon a collision with a

blocked grid cell;

2. **randomer** agent changes its direction randomly every move. This is the random agent of (Ishida & Korf 1991);
3. **minimax** agent uses a 5-ply minimax search at the ground level to decide on the escape move;
4. **DAM** agent uses dynamic abstract minimax with a 5-ply lookahead as described earlier in the paper;

### Effects of State Abstraction on Learning Agents

Table 2 presents the convergence execution cost for the eight combinations: four targets and two learning pursuit agents. Bold numbers indicate the best performance. Even though eMTS converged an order of magnitude faster than the original MTS in a pilot study, it was unable to converge in a reasonable amount of time while chasing a DAM agent.

Table 1: Learning agents: convergence execution cost on 1000 problems on 10 synthetic maps.

| Target/Pursuer | eMTS | PR MTS | Speed-up |
|---|---|---|---|
| randomer | 701 $\pm$ 182 | **110** $\pm$ **2** | 527 % |
| random | 380 $\pm$ 16 | **268** $\pm$ **6** | 42 % |
| minimax | 241 $\pm$ 38 | **214** $\pm$ **7** | 13 % |
| DAM | - | **1148** $\pm$ **31** | - |

For learning pursuit agents, path refinement with clique abstraction reduces the convergence travel distance by 13% to 527% and affords convergence for the most difficult target (DAM) agent. We believe that there are several mechanisms t work. First, there are fewer abstract states to learn heuristic values of. Second, each heuristic value learned at an abstract level influences pursuit agent behavior on *several* ground states (children of the abstract state). Third, target remains in each abstract state for a longer time, allowing more learning to happen before its position changes. This effect can also achieved with the commitment in MTS (Ishida 1993).

### Effects of State Abstraction on Search Agents

Table 2 presents the convergence execution cost for the eight combinations: four targets and two search pursuit agents. Bold numbers indicate the best performance.

Table 2: Search agents: convergence execution cost on 1000 problems on 10 synthetic maps.

| Target/Pursuer | A* | PRA* | Speed-up |
|---|---|---|---|
| randomer | 107 $\pm$ 2 | **102** $\pm$ **2** | 4.9 % |
| random | 217 $\pm$ 5 | **193** $\pm$ **4** | 12.4 % |
| minimax | **329** $\pm$ **14** | 355 $\pm$ 16 | – 7.3 % |
| DAM | **474** $\pm$ **15** | 649 $\pm$ 16 | – 26.9 % |

Abstraction has a mixed effect on search agents. While it reduces convergence travel when chasing simple targets (randomer and random), it actually has an adverse affect when pursuing more sophisticated targets that actually attempt to avoid the pursuit agent. We speculate that this may be due to the fact that abstraction can occasionally introduce suboptimality and additional overhead. A closer investigation into this phenomenon will commence shortly.

### Scalability to commercial maps

In order to investigate scalability and compare learning and search algorithms directly, we compared PR MTS and PRA* on 10 problems of varying difficulty randomly generated on a 6,176-state map from a commercial role-playing game and a 28,783-state map from a real-time strategy game (Figure 7). The initial distance between the start states of the target and pursuit agents ranged from 9.41 to 48.94 falling into five buckets: $(0, 10], (10, 20], \ldots$. There were two problems in each bucket (one from each map). Table 3 presents the convergence execution cost for a selection of the agents.

Table 3: Scalability: convergence execution cost on 10 problems on two commercial maps.

| Target/Pursuer | PR MTS | PRA* |
|---|---|---|
| randomer | 155 $\pm$ 38 | **145** $\pm$ **34** |
| random | 250 $\pm$ 38 | **201** $\pm$ **51** |
| DAM | 2161 $\pm$ 567 | **1936** $\pm$ **467** |

On the ten problems, PRA* outperformed PR MTS for every target agent. This may be because the dynamic abstraction level section mechanism of PRA* handles larger maps better than the fixed use of abstraction at the fixed level three in PR MTS. We are planning to verify this hypothesis via implementing a dynamic abstraction level selection mechanism within PR MTS. Note that all abstraction-enhanced pursuit agents are able to cope with maps of larger size than ever previously reported in the literature on moving target pursuit. Future work will investigate the practical limits of scalability by using even larger maps.

## Future Work and Conclusions

Future research will investigate dynamic abstraction level selection for learning agents, generalization of learned heuristic values across levels of abstraction, and dynamic lookahead depth selection. This is expected to eliminate the need to hand tune these parameters within PR MTS. We will also consider opponent modeling for target units.

In summary, this paper made several contributions. First, it demonstrated how automatically built hierarchical state abstraction can improve performance of all three components of moving target pursuit: search, adversarial search, and learning. Second, we proposed a systematic design of both pursuit and target agents which takes advantage of recent single-agent algorithms. Finally, we evaluated four target and four pursuit agents in an asynchronous environments similar to that of commercial real-time strategy and role playing games. The abstraction-enhanced algorithms scaled up to large maps from commercial games.

## Acknowledgments

PR MTS was originally suggested by Vadim Bulitko in September 2005. It was then implemented by Artit Visatemongkolchai, Jieshan Lu, and Jing Wu. We do not use their implementation in this paper. We appreciate funding by the Informatics Circle of Research Excellence and the National Science and Engineering Research Council and proof reading by Emilie Kohler.

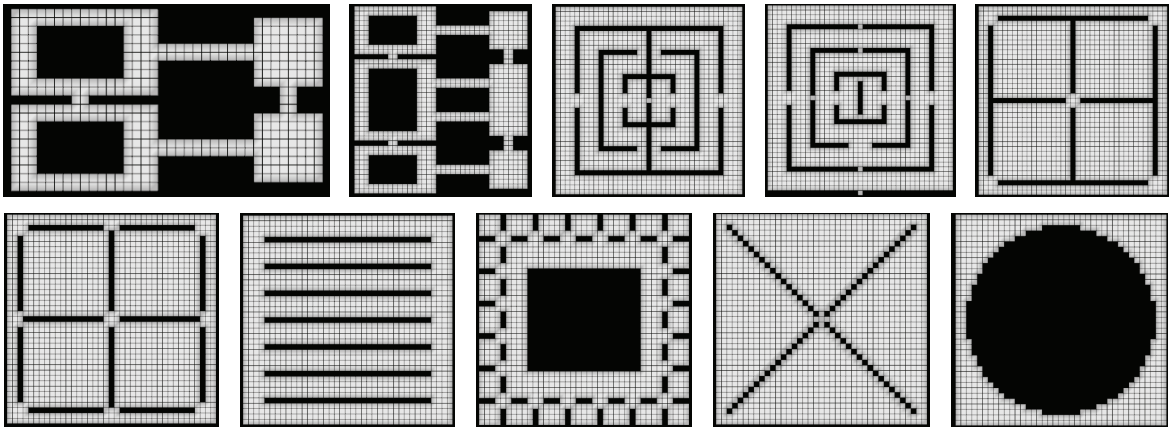

Figure 6: The 10 synthetic maps used in the first and second experiments.

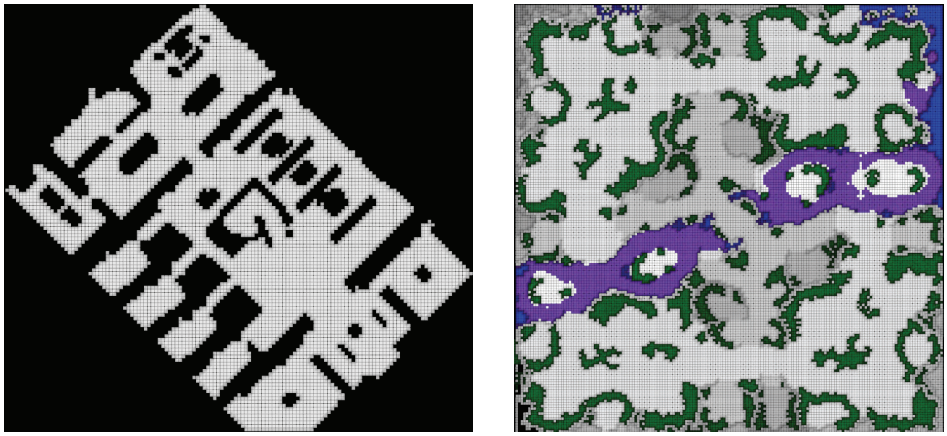

Figure 7: The two maps from commercial games used in the third experiment.

## References


Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development* 1(1):7–28.

Bulitko, V., and Lee, G. 2006. Learning in real time search: A unifying framework. *Journal of Artificial Intelligence Research* 25:119 – 157.

Bulitko, V.; Sturtevant, N.; and Kazakevich, M. 2005. Speeding up learning in real-time search via automatic state abstraction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1349 – 1354.

Bulitko, V. 2004. Learning for adaptive real-time search. Technical Report http: // arxiv. org / abs / cs.AI / 0407016, Computer Science Research Repository (CoRR).

Chimura, F., and Tokoro, M. 1994. The Trailblazer search: A new method for searching and capturing moving targets. In *Proceedings of the National Conference on Artificial Intelligence*, 1347–1352.

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Canadian Conference on AI*, 402–416.

Edelkamp, S. 1998. Updating shortest paths. In *Proceedings of the European Conference on Artificial Intelligence*, 655–659.

Hernández, C., and Meseguer, P. 2005. LRTA*(k). In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*.

Holte, R.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85(1-2):321–361.

Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. *LNAI 3607, Springer* 121–133.

Ishida, T., and Korf, R. 1991. Moving target search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 204–210.

Ishida, T., and Korf, R. 1995. A realtime search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17(6):609–619.

Ishida, T. 1992. Moving target search with intelligence. In *Proceedings of the National Conference on Artificial Intelligence*, 525–532.

Ishida, T. 1993. A moving target search algorithm and its performance improvement. *Journal of Japanese Society for Artificial Intelligence* 8(6):760–768.

Ishida, T. 1998. Real-time search for autonomous agents and multiagent systems. *Autonomous Agents and Multi-Agent Systems* 1(2):139–167.

Koenig, S.; Tovey, C.; and Smirnov, Y. 2003. Performance bounds for planning in unknown terrain. *Artificial Intelligence* 147:253–279.

Koenig, S. 2004. A comparison of fast search methods for real-time situated agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, 864 – 871.

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

Melax, S. 1993a. New approaches to moving target search. Master's thesis, Department of Computing Science, University of Alberta, Edmonton.

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

Sasaki, T.; Chimura, F.; and Tokoro, M. 1995. The Trailblazer search with a hierarchical abstract map. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 259–265.

Shimbo, M., and Ishida, T. 2003. Controlling the learning process of real-time heuristic search. *Artificial Intelligence* 146(1):1–41.

Shimbo, M. 1999. *Real-Time Search with Nonstandard Heuristics*. Ph.D. Dissertation, Department of Social Informatics, Kyoto University, Kyoto (Japan).

Shue, L.-Y.; Li, S.-T.; and Zamani, R. 2001. An intelligent heuristic algorithm for project scheduling problems. In *Proceedings of the 32nd Annual Meeting of the Decision Sciences Institute*.

Sturtevant, N., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1392–1397.