

Visibility Induction for Discretized Pursuit-Evasion Games

Ahmed Abdelkader and Hazem El-Alfy

Department of Engineering Mathematics and Physics

Faculty of Engineering, Alexandria University

Alexandria 21544, EGYPT

{*abdelkader,helalfy*}@*alexu.edu.eg*

Abstract

We study a two-player pursuit-evasion game, in which an agent moving amongst obstacles is to be maintained within “sight” of a pursuing robot. Using a discretization of the environment, our main contribution is to design an efficient algorithm that decides, given initial positions of both pursuer and evader, if the evader can take any moving strategy to go out of sight of the pursuer at any time instant. If that happens, we say that the evader *wins* the game. We analyze the algorithm, present several optimizations and show results for different environments. For situations where the evader cannot win, we compute, in addition, a pursuit strategy that keeps the evader within sight, for every strategy the evader can take. Finally, if it is determined that the evader wins, we compute its optimal escape trajectory and the corresponding optimal pursuit trajectory.

Introduction

We consider the problem of target tracking, that is planning the motion of a mobile robot (pursuer) as it tracks a target (evader) moving amongst obstacles. We use the term *target tracking* to mean *following* that target or, more precisely, maintaining its visibility. This terminology is common within the robotic planning community (Hsu, Lee, and Rong 2008) as opposed to the broader notion of tracking, known in the computer vision literature, which refers to the identification of paths of different targets. Several applications of that problem are suggested in the literature (Hsu, Lee, and Rong 2008; Bhattacharya and Hutchinson 2008; Murrieta-Cid et al. 2007). Those cover surveillance and security in sensitive or restricted areas, providing home care by watching over children or elderly people and monitoring the performance of human workers.

The studied problem is part of the more general *visibility-based pursuit-evasion problems* (LaValle 2006). In those problems, the task of a pursuer is to compute a path which guarantees finding an evader that might be hiding in an environment. Obviously, the evader might “sneak” between different hiding places as the pursuer is traveling making a single pursuer unable to solve the problem. The natural extension becomes that of finding the minimum number of pursuers needed to eliminate any hiding places for one or

more evaders under different constraints of knowledge of the environment and information about other players’ locations (Gerkey, Thrun, and Gordon 2004; Kolling and Carpin 2010; Klein and Suri 2011; Borie, Tovey, and Koenig 2011). Once the evader is found, the related problem that arises is that of maintaining its visibility which is known as *target tracking*. An early attempt at that problem is presented in (LaValle et al. 1997) for fully predictable targets (optimal tracking paths found offline) and partially predictable targets (best next step found online). Recently, there has been interest in a variant of that problem in which a pursuer, tracking an unpredictable evader, loses immediately if its view of the evader is obstructed by an obstacle (Bhattacharya and Hutchinson 2008; Murrieta-Cid et al. 2007). The problem of deciding which player wins for any pair of initial positions has been shown to be NP-complete (Murrieta-Cid et al. 2008).

In this paper, we are interested in the latter problem of deciding which player wins the pursuit-evasion game described above. We develop an algorithm, using a mesh discretization approach, to decide the outcome of the game. For any input pair of players’ positions, the algorithm decides whether the evader can win by moving in such a way to break the line of sight with the pursuer at any time instance or otherwise, the pursuer wins by maintaining the visibility of the evader throughout the game. Using a grid to discretize the surveyed environment has the advantages of being independent of the geometry and layout of the obstacles as well as being computationally feasible.

Grid discretization is often used in Artificial Intelligence to solve problems in *AI planning* (Ishida and Korf 1995). In AI, the focus is on solving the problem of finding and (or) catching the evader using algorithms that search the discretized state space along with heuristics to speed up the process. The problem of deciding if a solution exists is only approached theoretically and is often intractable. Recently, a polynomial time algorithm that decides which player wins and in how many steps has been developed (Hahn and MacGillivray 2006). Mathematical results from graph theory are used to present bounds on the time complexity of the problem that can be generalized, in theory, to the case of more players. A realization of such algorithms in real world problems is still not practical. In contrast, our approach presents a grid-based solution that can be applied in practice to a robotics path planning problem. It enriches the

literature by linking to existing research in that area, modeling realistic constraints of bounded speeds, different players' speeds and limited sensor range.

In the robotics motion planning area, two approaches are used. Using the terminology in (LaValle 2006), these are *combinatorial approaches* that find exact solutions through a continuous space and *sampling-based approaches* that divide the space (probabilistically or deterministically) into discrete regions and find paths through these regions. The simplest form of deterministically dividing the space (a.k.a. cell decomposition) is with a grid of fixed resolution. The main advantage of this approach is its simple implementation in contrast to combinatorial methods, many of which are impractical to implement. However, cell decomposition methods are *resolution complete* (unlike combinatorial methods), which means that they find a solution when one exists only if the resolution of the sampling grid is fine enough. Another common drawback with grid methods is that their complexity depends on the grid size (González-Baños, Hsu, and Latombe 2006).

The main contribution of this paper is the development of an algorithm that enhances recent results in AI planning and visibility-based pursuit-evasion to tackle the computationally prohibitive task of deciding the outcome of the pursuit-evasion game. This is a significant improvement over earlier depth-limited or heuristic-based approaches. The computed results are also used to find optimal player trajectories and optimize various objectives. The basic model (El-Alfy and Kabardy 2011) is formally studied to facilitate the derivation of several suggested optimizations. This, to our knowledge, provides first evidence of the feasibility of optimal decisions at such high grid-resolutions under varying speed ratios, different visibility constraints and regardless of obstacle geometries.

In the rest of this paper, we proceed with a more detailed literature review followed by a formal problem definition. Our approaches to solve the decision problem and compute tracking strategies are then presented and the results follow. Finally, we conclude and suggest future work.

Related Work

A large amount of literature has been devoted to pursuit-evasion games. In this section, we review closely related work, with a focus on attempts at deciding the outcome of the game. In the field of robotics, we survey recent work in the problem where one pursuer maintains the visibility of one evader, in an environment with obstacles. In artificial intelligence, we review the related problem of cops and robbers.

In the area of robotics motion planning, the main approach used is to decompose the environment into noncritical regions with critical curve boundaries, across which critical changes in occlusion and collision occur, then use a combinatorial motion planning method. Murrieta-Cid et al. (2007) model the pursuit-evasion game as a motion planning problem of a rod of variable length, creating a partitioning of the environment that depends on the geometry of obstacles. Later (2008), they present a convex partitioning that

is modeled as a *mutual visibility graph*. Their method alternates between an evader assumed to take the shortest step to escape, countered by a pursuer that computes a prevention-from-escape step, which produces a sequence of locally optimal paths. This leads to an interesting result: to decide which player wins, every feasible ordering of local paths has to be checked, concluding it is an NP-complete problem.

Bhattacharya et al. address the problem of maintaining the visibility of an escaping evader and show that it is completely *decidable* around one corner with infinite edges (Bhattacharya, Candido, and Hutchinson 2007). The authors then extend their work in (2008) to deal with more general environments with convex obstacles. They split the environment into decidable and non-decidable regions and approximate bounds on these regions. They also provide a sufficient condition for escape of the evader. Recently, they used differential games theory to analyze that problem under complete information, suggesting a formulation in which the pursuer maximizes the time for which it can track the evader while the evader minimizes it (Bhattacharya, Hutchinson, and Başar 2009; Bhattacharya and Hutchinson 2010). Computing equilibrium strategies gives necessary and sufficient conditions for tracking. They present results around a point obstacle, a corner and a hexagonal obstacle.

Within the AI planning community, the related problem of cops and robbers consists of one or more players (cops) trying to find and catch one or more evaders (robbers). The players perform alternating moves. This makes the game naturally discrete, often modeled as a graph with vertices representing the game's states. The mathematical foundations for solving these problems are surveyed in (Hahn 2007). A polynomial time optimal algorithm that determines whether the cops or the robbers win, and in how many steps, has only been recently given in (Hahn and MacGillivray 2006). Unfortunately, methods for computing optimal strategies have always been impractical to implement. For example, Moldenhauer and Sturtevant (2009a) compute optimal move policies offline in 2.5 hours per environment using an enhanced form of the algorithm in (Hahn and MacGillivray 2006). For that reason, several heuristics have been used in order to compute near optimal approximations in practical time. In (Moldenhauer and Sturtevant 2009b), different optimal strategies are studied on small maps while in (2009a), larger maps are used to evaluate less optimal strategies against optimal ones. One of the first practical implementations of the cops and robbers game is presented in (Ishida and Korf 1995) under the name of *moving-target search*. Since, that problem has been extensively studied using a variety of heuristics such as incremental heuristic search (Koenig, Likhachev, and Sun 2007) and *Cover* heuristic (Isaza et al. 2008), to name a few recent references.

Problem Definition

This is a two-player game, with one pursuer and one evader modeled as points that can move in any planar direction (holonomic robots). Each player knows exactly both the position and the velocity of the other player. We consider two-dimensional environments containing obstacles that obstruct the view of the players. Obstacles have known ar-

bitrary geometries and locations. The assumption of complete information is used here to derive the outcome of the game, since if some player loses with complete information, it will always lose under other conditions. Both players have bounded speeds, move at different speeds and can maneuver to avoid obstacles. We will denote the maximum speed of the pursuer v_p , that of the evader v_e and their ratio $r = v_e/v_p$. Players are equipped with sensors that can “see” in all directions (omnidirectional) and as far as the environment boundaries or obstacles, whichever is closer. We will see later that we can model minimum and maximum ranges for sensors with a simple variation in our algorithm.

The pursuit-evasion game proceeds as follows. Initially, the pursuer and the evader are at positions from which they can see each other. It is common in the literature to define two players to be visible to one another if the line segment that joins them does not intersect any obstacle. The goal of the game is for the pursuer to maintain visibility of the evader at all times. The game ends immediately, if at any time, the pursuer loses sight of the evader. In that case, we say that the pursuer loses and the evader wins.

Deciding the Outcome of the Game

Deciding the game requires the construction of a binary function in two variables for the initial positions of both players. In order to study the progress of the game, we introduce a third parameter for the time index which is a discrete version of time in the continuous case. We call this function $Bad(p, e, i)$. When Bad evaluates to 1, it means there exists a strategy for an evader starting at e to go out of sight of a pursuer at p by the i^{th} time index. It is clear that $Bad(p, e, 0)$ corresponds directly to visibility constraints and is straight forward to compute. We seek an algorithm to determine the value of $Bad(p, e)$, with the time index dropped to indicate the end result of the game as time tends to infinity. In logical contexts, Bad is used as a predicate.

The Visibility Induction Algorithm

Fix a pursuer and evader at grid cells p and e and let i be the time index. If $Bad(p, e, 0) = 1$, then the evader is not initially visible to the pursuer and the game ends trivially with the pursuer losing. Now, consider the case where the evader manages to escape at step $i + 1$. To do so, the evader must move to a neighboring cell e' where no corresponding move exists for the pursuer to maintain visibility. In other words, all neighbors p' of the pursuer either cannot see the evader at e' or, otherwise, were shown unable to keep an evader at e' in sight up to step i i.e. $Bad(p', e', i) = 1$. This means that when $Bad(p, e, i + 1)$ is updated to 1, the outcome of the game has been decided for the configuration in question as a losing one, i.e. there is a strategy for the evader to escape the sight of the pursuer at some step $\geq i$, but not any sooner. We use $\mathcal{N}(c)$ for the set of neighboring cells a player at c can move to. With that, we have the following recurrence:

$$Bad(p, e, i + 1) = 1 \quad \forall (p, e) \exists e' \in \mathcal{N}(e) \text{ s.t.} \\ \forall p' \in \mathcal{N}(p) \quad Bad(p', e', i) = 1 \quad (1)$$

Equation (1) computes $Bad(p, e, i + 1)$ inductively by evaluating the necessary escape conditions as of the i^{th} step. For any given non-trivial initial configuration, the very first application of the inductive step would only yield a change for cases where the evader starts right next to an obstacle and is able to hide behind it immediately. That is because the $Bad(p, e, 0)$ is only 1 for initially obstructed cells. After many iterations of the induction over all cells, more pairs farther and farther from obstacles get marked as bad. The expansion of the bad region only stops at cells which the pursuer in question is able to track and the function stabilizes with $Bad(p, e) = 1$ if and only if the evader can win. This bears a discrete resemblance to integrating the *adjoint equations* backward in time from the termination situations as presented in (Bhattacharya and Hutchinson 2010).

Algorithm 1 performs *backward visibility induction* as defined in (1) to matrix M which is initialized with initial visibility constraints. To determine mutual visibility between cells, we use Bresenham’s line algorithm (Bresenham 1965) to draw a line connecting every two cells and see if it passes through any obstacle. This approach works for any geometry and is easy to implement. More sophisticated algorithms could be used but are hardly justified. Restrictions on visibility can be easily incorporated by modifying the initialization part at line 5. For example, for a limited sensing range R_{max} , we add **or** $D(p, e) > R_{max}$, where D is the distance. If the pursuer is not to come any closer than a minimum distance to the evader, a lower bound R_{min} can be added as well.

Algorithm 1: Decides the game for a given map.

Input : A map of the environment.

Output: The Bad function encoded as a bit matrix.

Data: Two $N \times N$ binary matrices M and M' .

```

1 begin
2   Discretize the map into a uniform grid of  $N$  cells.
   // Visibility initialization
3   Initialize  $M$  and  $M'$  to 0.
4   foreach  $(p, e) \in grid \times grid$  do
5     | if  $e$  not visible to  $p$  then  $M[p, e] = 1$ 
6   end
   // Induction loop
7   while  $M' \neq M$  do
8     |  $M' = M$ 
9     | foreach  $(p, e) \in grid \times grid$  do
10    | | if  $\exists e' \in \mathcal{N}(e)$  s.t.  $\forall p' \in \mathcal{N}(p)$   $M'[p', e'] = 1$ 
11    | | then  $M[p, e] = 1$ 
12    | end
13  end
14  return  $M$ 

```

Proof of Correctness

We start by showing that the algorithm always terminates. At the end of each iteration, either $M' = M$ and the loop exits or more cells get marked as bad, which stops when

all cells are marked, leaving $M' = M$. Next, we use this induction:

1. By line 6, M contains the decision at step 0 as enforced by the visibility constraints computed in line 5.
2. After the i^{th} iteration of the loop at line 9, $M[p, e] = 1$ iff the evader has an escape strategy e' where the pursuer has no corresponding strategy p' with $M[p', e'] = 0$.

Complexity Analysis

The algorithm uses $\mathcal{O}(N^2)$ storage for the output and temporary matrices M and M' . Initializing the matrices takes $\mathcal{O}(N^2\sqrt{N})$ time if naive line drawing is used for each pair, which is linear in the length of the line. The inner loop at line 9 processes $\mathcal{O}(N^2)$ pairs each costing $\mathcal{O}(\kappa^2)$ where $\kappa = \max(|\mathcal{N}(p)|, |\mathcal{N}(e)|)$. Per the preceding discussion, at the i^{th} iteration, the algorithm decides the game for all escape paths of length $(i+1)$. Let L be the length of the longest minimal escape trajectory for the given environment. Obviously, the induction loop at line 7 is executed $\mathcal{O}(L)$ times. We can see that L depends on the largest open area in the environment and also the speed ratio r , with equal speeds being the worst case, where the distance between the players may not change, as the game ends earlier otherwise. A worst case scenario is an equally fast evader starting very close to the pursuer. For such an evader to win, it would need to move along with the pursuer to the closest obstacle where it can break visibility. We conclude that $L = \mathcal{O}(N)$ and would typically be smaller in practice. With that, the visibility induction algorithm is $\mathcal{O}(\kappa^2 N^3)$.

Theorem 1. (Visibility Induction) *Algorithm 1 decides the discretized game for a general environment in $\mathcal{O}(\kappa^2 N^3)$.*

Proof. By the discussion above, the proof follows. \square

Practicalities and Optimizations

We present several enhancements to the visibility induction algorithm and the speedups they yield. Our time measurements are performed using test maps of 100×100 cells for speed ratios $[1, \frac{4}{5}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{5}]$. All run times are averaged over 10 runs.

Memory Savings As binary matrices, both M and M' need only 1 bit per entry. It is also obvious we need only store bits for valid states, which reduces N to the number of free cells. This allows N to exceed 60,000 using less than 1GB of memory, which enables processing at resolutions around 250×250 cells.

Parallelization Observe that the loop at line 9 reads from matrix M and writes to M' . This means that M' updates are *embarrassingly parallel*. In our C++ implementation, we used the cross-platform OpenMP library to exploit this property. By adding a single line of code, we were able to harness the multiprocessing capabilities commonly available today. This allowed a 36% average speedup.

Level-0 Caching It is evident most of the computations are dedicated to evaluating the escape conditions as presented in (1). It is crucial to skip any unnecessary evaluations that yield no updates.

Lemma 2. $Bad(p, e, i) \implies Bad(p, e, j) \forall j > i$

Proof. For $Bad(p, e, i+1)$ in (1), put $e' = e$. \square

Corollary 3. $\neg Bad(p, e, i) \implies \neg Bad(p, e, j) \forall j < i$

Lemma 2 allows skipping pairs that have already been decided by a simple addition to the condition in line 10.

Level-1 Caching As Lemma 4 suggests, we need only re-evaluate the conditions for those players who witnessed a change at the previous iteration. This can be applied independently to pursuers and evaders leading to a 45% average speedup. When applied to both we reached 50%. This comes at an additional $\mathcal{O}(N)$ storage, which is negligible compared to the M matrix.

Lemma 4. (Synchronized Neighborhoods)

$$\begin{aligned} \neg Bad(p, e, i) \wedge Bad(p, e, i+1) &\implies \\ &\exists (p^*, e^*) \in \mathcal{N}(p) \times \mathcal{N}(e) \text{ s.t.} \\ &\neg Bad(p^*, e^*, i-1) \wedge Bad(p^*, e^*, i) \end{aligned}$$

Proof.

$$\begin{aligned} \neg Bad(p, e, i) &\implies \neg Bad(p, e, i-1) \quad (\text{by C.3}) \\ &\implies \forall e' \in \mathcal{N}(e) \exists p^* \in \mathcal{N}(p) \text{ s.t.} \\ &\quad \neg Bad(p^*, e', i-1) \end{aligned} \quad (2)$$

$$\begin{aligned} Bad(p, e, i+1) &\implies \exists e^* \in \mathcal{N}(e) \text{ s.t. } \forall p' \in \mathcal{N}(p) \\ &\quad Bad(p', e^*, i) \end{aligned} \quad (3)$$

By (2) and (3), the existence of (p^*, e^*) is established. \square

Level-2 Caching Strict application of Lemma 4 results in re-evaluating the conditions only for pairs who witness *related* changes. Keeping track of that comes at a higher storage cost of $\mathcal{O}(N^2)$, which is equivalent to the M matrix. Adding the level-2 cache resulted in a 52% average speedup. With that, we reach a new complexity result. Note that caching under parallelization is particularly tricky and requires careful update and invalidation mechanisms.

Lemma 5. *Level-2 caching makes the induction loop $\mathcal{O}(\kappa^4 N^2)$.*

Proof. By only processing a pair (p, e) having a related update in both $\mathcal{N}(p)$ and $\mathcal{N}(e)$, no pair gets processed more than $|\mathcal{N}(p)| \times |\mathcal{N}(e)| = \mathcal{O}(\kappa^2)$ times. As the total number of pairs is $\mathcal{O}(N^2)$ and processing a single pair takes $\mathcal{O}(\kappa^2)$, this amounts to $\mathcal{O}(\kappa^4 N^2)$. \square

More Memory Savings It is possible to do without the auxiliary M' matrix. Instead of copying values before the inner loop and doing all checks on old values, we can use the M matrix for both checks and updates. If some entry $M[p, e]$ is not updated, the behavior would be the same. On the other hand, if $M[p, e]$ got updated, the algorithm would use a newer value instead of waiting for the next iteration which results in a minor speedup as a side-effect.

Applying all the enhancements discussed in this section led to a 64% average speedup on our test sample. We notice that for the medium sized square grids we consider, initialization of matrices is above quadratic by a small factor which is dominated by the number of iterations L . Furthermore, as κ is typically limited (players have bounded speeds) and can be considered constant for a given realization, it may be ignored in comparison to N as the algorithm approaches $\mathcal{O}(N^2)$.

For the typical case of a limited sensing region of size R , the algorithm need only consider that many evaders. This effectively replaces one N in all the above expressions and allows processing at much higher resolutions.

Discrete Tracking Strategies

Because the *Bad* function decides the game for all pairs including all combinations of the neighboring cells for both players, it can be used beyond determining the winner for trajectory planning. As a zero-sum game, there will only be one winner; and a valid trajectory must maintain this property. Further objectives can be defined as needed and an optimal trajectory can then be chosen. In particular, we are interested in optimal escape trajectories for a winning evader minimizing the visibility time. Other objectives relating to the distance between players, the distance traveled or speed of maneuvering can also be used.

Extensions to Guaranteed Tracking

The computed *Bad* function can be used directly in basic trajectory planning for the winning player. A valid trajectory must maintain the winning state by only moving via cells with guaranteed win regardless of the strategy followed by the opponent. A higher level plan may then choose any of the valid neighbors, which are guaranteed to exist for the winner. To unify the notation used below, we define:

$$\begin{aligned} Lose(a, b) = & (Pursuer(a) \wedge Bad(a, b)) \vee \\ & (Evader(a) \wedge \neg Bad(b, a)) \end{aligned} \quad (4)$$

Algorithm 2 first discards invalid neighbors a winner must not move to, then chooses one of the remaining neighbors. Typically, a distance function is used for tie breaking. For example, a pursuer would generally prefer to move closer to the evader and keep it away from bad cells.

Algorithm 2: Generic trajectory planning for winners.

```

Input:  $Bad(., .)$ , current state ( $player, opponent$ ).
1 begin
2    $\mathcal{N}^* = \{\}$ 
3   foreach  $n \in \mathcal{N}(player)$  do
4     if  $\neg Lose(n, n') \forall n' \in \mathcal{N}(opponent)$  then
5        $\mathcal{N}^* = \mathcal{N}^* \cup n$ 
6     end
7   end
8   Move to any neighbor in  $\mathcal{N}^*$ .
9 end

```

Optimal Trajectory Planning

If the pursuer can keep the evader in sight, there is not much the evader can do as far as we are concerned. On the other hand, if the evader can win the game, it is particularly important to minimize the time taken to break the line of sight to the pursuer. The losing pursuer must also maximize that time by not making suboptimal moves that allow the evader to escape faster. To compute these optimal trajectories, we modify Algorithm 1 to store the time index i , at which the game got decided, into $M[p, e]$. In lines 5 and 10, we use 0 and i , respectively, instead of just 1, and only make the assignment once for the smallest i . The matrices are initialized to ∞ to indicate the absence of an escape strategy for the evader and the condition in line 10 is modified accordingly. We call the enhanced *Bad* function $J(p, e)$ as it gives the time left for visibility, which corresponds to the value of the game as in (Bhattacharya and Hutchinson 2010).

Theorem 6. (Time-Optimal Trajectories) $J(p, e)$ gives the time left before visibility is broken, assuming both players move optimally.

Proof. Trivially, $J(p, e) = \infty \implies \neg Bad(p, e)$ and the evader has no escape strategy. When $Bad(p, e)$ is marked at the i^{th} iteration for a given pair, an escape trajectory becomes available to the evader at e' . No such escape trajectory could be found up to step $i - 1$. By definition of the escape cell e' and J :

$$\forall p' \in \mathcal{N}(p) \quad Bad(p', e', i - 1) \implies J(p', e') < J(p, e)$$

By following $e' \rightarrow e'' \rightarrow \dots \rightarrow e^{(k)}$, $J(p^{(k)}, e^{(k)})$ is guaranteed to reach 0 at cell $e^{(k)}$ where visibility is broken. From Lemma 4, $\exists p^* \in \mathcal{N}(p)$ s.t. $J(p, e) = J(p^*, e') + 1$. By repeatedly selecting p^* , the pursuer can force k to attain its maximum value i.e. $J(p, e)$. \square

At each step, the players will be moving to neighbors p' and e' , maximizing $J(p', e)$ and minimizing $J(p, e')$, respectively. The obtained $J(p, e)$ can be processed further to have these neighbors precomputed into separate functions S_p and S_e , encoding the trajectories for each player. However, as storing the time index i increases the required storage, the algorithm can be modified to compute S_p and S_e directly. This allows reducing the storage by describing the neighbor relative to the current position of the player. The neighborhoods can be indexed unambiguously which allows the precomputed S functions to store just as many bits as necessary per entry, which is as small as $\lceil \log_2 \kappa \rceil$. We omit the modified algorithms due to the limited space.

Experimental Results

We implemented our algorithms in C++ and performed experiments on an Intel Core i7 CPU running at 2.67GHz with 4GB of RAM. We experiment with manually created environments containing obstacles of various forms. Initial players positions are randomly selected satisfying certain visibility constraints and the evader's paths are automatically generated as discussed in the above section on tracking strategies. We discretized the maps of the used environments with

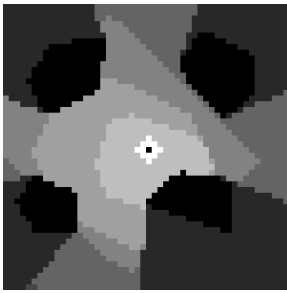


Figure 1: Decision map (evader) - 50x50 map.

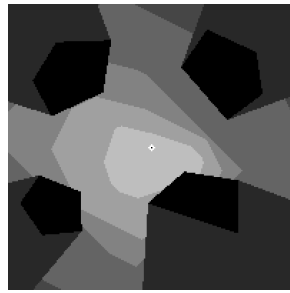


Figure 2: Decision map (evader) - 200x200 map.



Figure 3: Decision map (pursuer) around a corner.

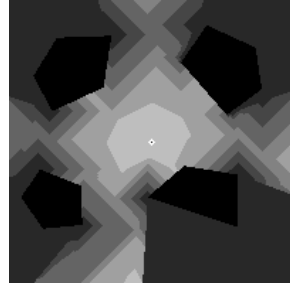


Figure 4: Decision map (pursuer) for polygonal obstacles.

regular grids of sizes ranging from 50×50 to 400×400 cells and used 4-connected and 8-connected neighborhoods. With such fine granularities, we anticipate moving to real world environments.

The effect of varying the grid size on the resolution of decision maps is shown in figures 1 and 2 for speed ratios $[1, \frac{4}{5}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{5}]$. The boundaries of the nested convex regions are such that if the two players fall inside a region, the pursuer can track the evader indefinitely, while if one player is inside and the other outside, the evader can escape. The darker the gray shade, the smaller the speed ratio. Obstacles are in black and the player mentioned in each figure is the black dot roughly centered inside all nested regions. Our approach works independently of obstacle shapes and layouts as shown in figures 3 to 5. Modeling visibility constraints (e.g. limited sensor range) affects the decision regions as in figure 6. All previous results are computed for 4-connected neighborhoods. Figures 7 and 8 contrast 4-connected to 8-connected neighborhood maps for a speed ratio of 0.5. Finally, we show two tracking scenarios in figures 9 and 10.

As we vary the grid size, runtime is affected as shown in table 1. It fits a quadratic model in N (for a fixed neighborhood size) as by the discussion following Lemma 5.

Conclusions

We addressed the problem of maintaining an unobstructed view of an evader moving amongst obstacles by a pursuer. In



Figure 5: Decision map (evader) for a circular obstacle.

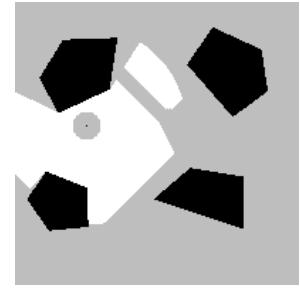


Figure 6: Decision map (evader) with restricted visibility.

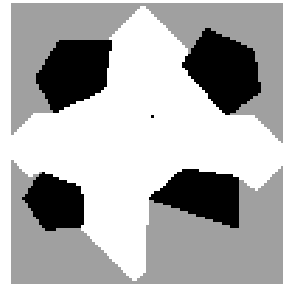


Figure 7: Decision map (evader) for a 4-connected neighborhood.

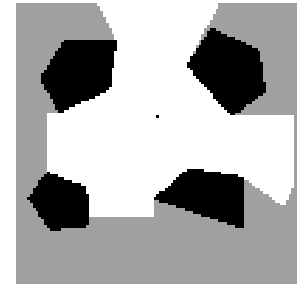


Figure 8: Decision map (evader) for an 8-connected neighborhood.

particular, we developed an algorithm that decides the outcome of the game for any pair of initial positions of the players. By employing a space/time discretization approach, the solution becomes feasible in polynomial time, is independent of the geometry of the environment and does not require the use of heuristics. We give a detailed analysis for the correctness of the algorithm, derive its space and time complexities and present and verify several approaches to reduce its time and memory demands. We extended our algorithm to compute tracking or escape trajectories for both players in real time and verified their optimality. We tested our method on different tracking scenarios and environments.

We are currently working on a realization of the proposed

Table 1: Average runtime for Algorithm 1 vs. grid size

Grid size	Runtime (hh:mm:ss)
50×50	00:00:01
60×60	00:00:02
75×75	00:00:06
100×100	00:00:23
120×120	00:00:52
150×150	00:02:27
200×200	00:09:24
300×300	01:06:18
400×400	06:47:57

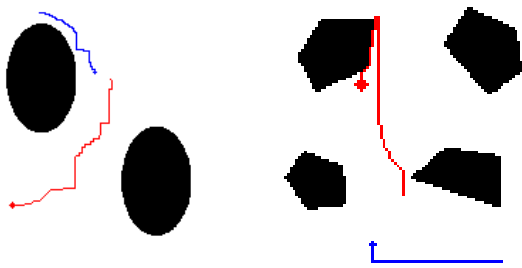


Figure 9: Example of a winning evader (blue; top). Figure 10: Example of a winning pursuer (red; top).

method using real robots equipped with sensors. To model the real world more accurately, we consider hexagonal mesh discretizations, with several interesting properties, and study decision errors for a given resolution. We already implemented limited range sensors and an extension to a limited field of view is systematic. Other constraints on players motion can be considered such as restricted areas where the pursuer is not allowed to go into. Regaining lost visibility or, alternately, allowing for blind interruptions are interesting extensions with a slight relaxation to the hard visibility constraint. Finally, we envision efficient ways to extend this approach to the case of more players.

References

Bhattacharya, S., and Hutchinson, S. 2008. Approximation schemes for two-player pursuit evasion games with visibility constraints. In *Proceedings of Robotics: Science and Systems IV*.

Bhattacharya, S., and Hutchinson, S. 2010. On the existence of nash equilibrium for a two player pursuit-evasion game with visibility constraints. *The International Journal of Robotics Research* 29(7):831–839.

Bhattacharya, S.; Candido, S.; and Hutchinson, S. 2007. Motion strategies for surveillance. In *Proceedings of Robotics: Science and Systems III*.

Bhattacharya, S.; Hutchinson, S.; and Başar, T. 2009. Game-theoretic analysis of a visibility based pursuit-evasion game in the presence of obstacles. In *Proc. American Control Conference (ACC'09)*.

Borie, R.; Tovey, C.; and Koenig, S. 2011. Algorithms and complexity results for graph-based pursuit evasion. *Autonomous Robots* 31(4):317–332.

Bresenham, J. 1965. Algorithm for computer control of a digital plotter. *IBM Systems Journal* 4(1):25–30.

El-Alfy, H., and Kabardy, A. 2011. A new approach for the two-player pursuit-evasion game. In *Proc. 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI'11)*, 396–397. Incheon, Korea: IEEE.

Gerkey, B. P.; Thrun, S.; and Gordon, G. 2004. Visibility-based pursuit-evasion with limited field of view. 20–27.

González-Baños, H. H.; Hsu, D.; and Latombe, J.-C. 2006. Motion planning: Recent developments. *Autonomous Mo-*

bile Robots: Sensing, Control, Decision-Making and Applications (CRC Press) 373–416.

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

Hahn, G. 2007. Cops, robbers and graphs. *Tatra Mountains Mathematical Publications* 36:163–176.

Hsu, D.; Lee, W. S.; and Rong, N. 2008. A point-based POMDP planner for target tracking. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'08)*, 2644–2650.

Isaza, A.; Lu, J.; Bulitko, V.; and Greiner, R. 2008. A cover-based approach to multi-agent moving target pursuit. In *Artificial Intelligence and Interactive Entertainment Conference (AIIDE)*.

Ishida, T., and Korf, R. E. 1995. Moving-target search: a real-time search for changing goals. *Pattern Analysis and Machine Intelligence, IEEE Transactions on (TPAMI)* 17(6):609–619.

Klein, K., and Suri, S. 2011. Complete information pursuit evasion in polygonal environments. In *Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI'11)*, 1120–1125.

Koenig, S.; Likhachev, M.; and Sun, X. 2007. Speeding up moving-target search. In *Proceedings of the 6th international joint conference on Autonomous agents and multi-agent systems (AAMAS'07)*, 1–8. Honolulu, HI, USA: ACM.

Kolling, A., and Carpin, S. 2010. Multi-robot pursuit-evasion without maps. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'10)*, 3045–3051.

LaValle, S. M.; González-Baños, H. H.; Becker, C.; and Latombe, J.-C. 1997. Motion strategies for maintaining visibility of a moving target. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'97)*, volume 1, 731–736.

LaValle, S. M. 2006. *Planning Algorithms*. New York, NY, USA: Cambridge University Press.

Moldenhauer, C., and Sturtevant, N. R. 2009a. Evaluating strategies for running from the cops. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, 584–589.

Moldenhauer, C., and Sturtevant, N. R. 2009b. Optimal solutions for moving target search. In *Proceedings of The 8th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS'09) - Volume 2*, 1249–1250.

Murrieta-Cid, R.; Muppurala, T.; Sarmiento, A.; Bhattacharya, S.; and Hutchinson, S. 2007. Surveillance strategies for a pursuer with finite sensor range. *The International Journal of Robotics Research* 26(3):233–253.

Murrieta-Cid, R.; Monroy, R.; Hutchinson, S.; and Laumond, J.-P. 2008. A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'08)*, 2657–2664.