Artificial Intelligence in Adversarial Real-Time Games: Papers from the 2012 AIIDE Workshop
AAAI Technical Report WS-12-15

Adversarial Planning for Multi-Agent Pursuit-Evasion Games in Partially Observable Euclidean Space

Eric Raboin
University of Maryland
Computer Science Dept., and
Inst. for Systems Research
College Park, MD, USA
eraboin@cs.umd.edu

Ugur Kuter
Smart Information Flow
Technologies (SIFT), LLC
211 N. 1st St., Suite 300
Minneapolis, MN, USA
ukuter@sift.net

Dana Nau
University of Maryland
Computer Science Dept., and
Inst. for Systems Research
College Park, MD, USA
nau@cs.umd.edu

S. K. Gupta
University of Maryland
Mechanical Engr. Dept., and
Inst. for Systems Research
College Park, MD, USA
skgupta@umd.edu

Abstract

We describe a heuristic search technique for multi-agent pursuit-evasion games in partially observable Euclidean space where a team of trackers attempt to minimize their uncertainty about an evasive target. Agents' movement and observation capabilities are restricted by polygonal obstacles, while each agent's knowledge of the other agents is limited to direct observation or periodic updates from team members.

Our polynomial-time algorithm is able to generate strategies for games in continuous two-dimensional Euclidean space, an improvement over past algorithms that were only applicable to simple gridworld domains. We demonstrate that our algorithm is tolerant of interruptions in communication between agents, continuing to generate good strategies despite long periods of time where agents are unable to communicate directly. Experiments also show that our technique generates effective strategies quickly, with decision times of less than a second for reasonably sized domains with six or more agents.

Introduction

Pursuit and evasion strategies are important in many video-game environments, and this paper concentrates on generating such strategies in continuous, partially observable Euclidean space. We provide a polynomial time algorithm capable of generating online strategies for a team of cooperative tracker agents that wish to pursue an evasive target. The goal of the tracker team is to minimize their uncertainty about the target's location by the end of a fixed time period. The domain may have arbitrarily shaped polygonal obstacles that limit movement as well as observability.

To minimize uncertainty about a target's location, the trackers must work both to maintain visibility on the target, but also to move to strategic locations prior to visibility loss so that recovery is possible. Past approaches aimed at maintaining visibility for as long as possible (Muppirala, Hutchinson, and Murrieta-Cid 2005; Murrieta et al. 2004), or discovering the location of a hidden target (Suzuki and Yamashita 1992; LaValle et al. 1997), is not suited for scenarios where the target frequently passes in and out of visibility. Since we want to generate strategies quickly, this also rules out many combinatorial search techniques.

Prior work on this problem included a game-tree search algorithm that could generate strategies for simple gridworld domains, where time was divided into discrete time steps and agents were only permitted to move in one of four cardinal directions (Raboin et al. 2010). That work also assumed that agents would be in constant communication, since it generated trajectories using a heuristic method that required knowing the location of every agent on the team.

We introduce the *Limited-communication Euclidean-space Lookahead* (*LEL*) heuristic for evaluating tracker strategies in 2-D Euclidean space, with sporadic communication among the trackers. Our contributions include—

- An algorithm for computing *LEL* in two-dimensional Euclidean space with polygonal obstacles, where communication between agents may be interrupted for long periods of time.
- An efficient method for computing the set of trajectories for each agent that are consistent with a trackers' *observation history*, which consists of direct observations and information shared periodically by other tracker agents.
- Complexity analysis showing that our algorithm for computing *LEL* runs in polynomial time with respect to the size of the domain and number of agents.
- Experiments showing that our algorithm quickly generates strategies in the continuous domain that are twice as effective at retaining visibility on the target, compared to a strategy that follows the shortest path to the target.

Formalism

We define a multi-agent, imperfect-information game where a single *target* agent a_0 is pursued by a team of n *tracker* agents $\{a_1, a_2, \ldots a_n\}$. The tracker team's goal is to minimize uncertainty about the target's location at the end of the game. The agents' capabilities are defined below.

We assume that each agent a_i is a holonomic point robot with a fixed maximum velocity v_i. Agents can be located anywhere in the region $C_{free} \subseteq \mathbb{R}^2$, defined as free space. The domain may have multiple obstacles, where each obstacle is a polygon in $\mathbb{R}^2$, and C_{free} is the set of locations not intersecting any obstacles. The game's state $s \in S$ is a set of locations for each agent, $\{l_0, l_1, \ldots l_n\}$, and a time t_k. Each game has an initial state s_0, and a time t_{end} indicating when the game ends. A game's history $h \in H$ at time $t_k \leq t_{end}$ is the set of trajectories followed by each agent $\{f_0, f_1, \ldots f_n\}$ from time t_0 until t_k, where $f_i(t)$ de-

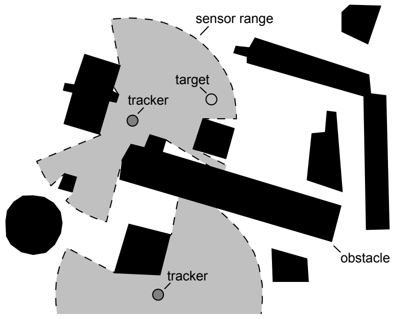

Figure 1: Example pursuit scenario with two tracker agents and a single target agent. Shaded areas represent the region that can be observed by the tracker agents.

notes the location of agent a_i at time t. Since agents can move freely in two-dimensional Euclidean space, the set of all states S, and set of all game histories H, are both infinite.

Agent a_i can travel from location l_j to location l_k only if a path exists from l_j to l_k, and every location in that path is contained in C_{free}. Thus, agent a_i's *reachability* function is

$$R_i(l_j,t) = \{l_k : \text{locations } \langle l_j, l_k\rangle \text{ are connected in } C_{free} \text{ by a path of length } d \leq t/v_i\}$$

which is the set of locations agent a_i can reach in time t starting from location l_j. This can be generalized to

$$R_i(L,t) = \{l_k : l_j \in L \wedge l_k \in R_i(l_j,t)\} \quad (1)$$

which is the set of locations agent a_i can reach in time t starting from anywhere in $L \subseteq \mathbb{R}^2$.

Agent a_i can observe location l_k from location l_j only if l_k is contained within the observable region $V_i(l_j)$. We define the observable region as the set of locations within a_i's sensor range, r_i, where the line-of-sight is not obstructed by an obstacle. Thus, agent a_i's *observability* function is

$$V_i(l_j) = \{l_k : \text{ locations } \langle l_j, l_k\rangle \text{ are connected in } C_{free} \text{ by a line segment of length } d \leq r_i\}$$

which is the set of locations observable to agent a_i while located at l_j. Observability can also be generalized as

$$V_i(L) = \{l_k : l_j \in L \wedge l_k \in V_i(l_j)\} \quad (2)$$

which is the set of locations agent a_i can observe while located somewhere in $L \subseteq \mathbb{R}^2$. An example state of the game that illustrates observability is shown in Fig. 1.

Agent a_i may recall its past location $f_i(t)$ for any time $t \leq t_k$, but it does not know the trajectory f_j followed by any other agent $a_{j \neq i}$. a_i only knows the location of the other agents in the initial state s_0 and its observation history.

During a game, agent a_i's *observation history* is a finite set of observations $\mathcal{O}_i = \{o_0, o_1, \ldots o_k\}$ where each observation is a tuple $\langle a_j, L, t\rangle$, meaning $f_j(t) \in L$, or "agent a_j is located in region L at time t." If observation $o \in \Omega$ is in agent a_i's observation history at time t, the information in o is available to a_i at any time $t' \geq t$. As with states and histories, the set of possible observations, Ω, is infinite.

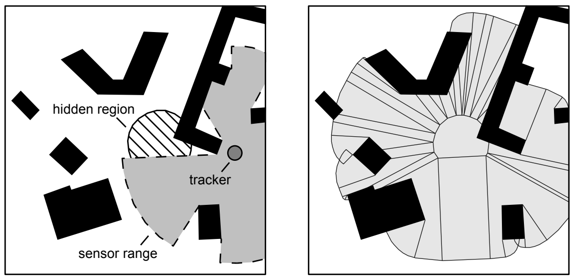

Figure 2: Left: example where a tracker has lost sight of a target. The hidden region is the hatched area. Right: polygons generated by expanding the hidden region's boundary.

Observations are made at discrete time intervals, such that the number of observations in a particular observation history remains finite. Since agents are free to move between observations, we define a set of rules for computing the possible trajectories followed by each agent that are consistent with prior observations.

Given an observation history, an agent is able to determine the region where another agent may be located, even if that agent's actual location is not known. Given $\mathcal{O}_i$, the set of locations guaranteed to contain agent a_j at time t is

$$R^+_j(\mathcal{O}_i, t) = R_j(L, t - t') \quad (3)$$

where $\langle a_j, L, t'\rangle$ is the most recent observation in $\mathcal{O}_i$ describing agent a_j at some time $t' \leq t$. This expands the set of locations where a_j might be, as illustrated in Fig. 2.

If agent a_i happens to observe agent a_j at time t, meaning $f_j(t) \in V_i(f_i(t))$, then the tuple $\langle a_j, \{f_j(t)\}, t\rangle$ is added to agent a_i's observation history. If a_i does not directly observe a_j, the observation history is updated with a set of locations instead. Agent a_i can compute this *hidden* region as

$$\textit{hidden}_j(\mathcal{O}_i, f_i, t) = R^+_j(\mathcal{O}_i, t) \setminus V_i(f_i(t)) \quad (4)$$

which is the set of locations that a_j can reach by time t, minus the locations observed by agent a_i. If a_i does not directly observe a_j at time t, then $\langle a_j, \textit{hidden}_j(\mathcal{O}_i, f_i, t), t\rangle$ is added to agent a_i's observation history.

Each tracker receives periodic updates from the other agents on their team. An update from tracker a_j includes a_j's current location and a_j's observation history $\mathcal{O}_j$. This can be merged with a_i's latest observations by computing

$$\textit{merge}_k(\mathcal{O}_i, \mathcal{O}_j, t) = R^+_k(\mathcal{O}_i, t) \cap R^+_k(\mathcal{O}_j, t) \quad (5)$$

where $\langle a_0, \textit{merge}_0(\mathcal{O}_i, \mathcal{O}_j, t), t\rangle$ represents a_i and a_j's combined knowledge of the target at time t. This observation, and $\langle a_j, \{l_j\}, t\rangle$ are both added to tracker a_i's observation history as the result of the update.

Agent a_i's observation history $\mathcal{O}_i$ and past trajectory f_i map to an information set $I_i(t) \subseteq H$,i.e., the set of possible game histories given a_i's knowledge at time t. History h is in $I_i(t)$ if and only if $\langle f_i, \mathcal{O}_i\rangle$ is consistent with h. Formally, $I_i(t) = \{h : (f_i \in h) \wedge \forall_{f_j \in h} C(\mathcal{O}_i, f_j)\}$, where $C(\mathcal{O}_i, f_j)$ is the consistency relationship $C(\mathcal{O}_i, f_j) = \langle a_j, L, t\rangle \in \mathcal{O}_i \to f_j(t) \in L$. As with states and histories, the set of all possible information sets at time $t > t_0$ is infinite.

In practice, we only need the most recent observation in each history. This is sufficient both to compute *LEL* (see below) and to maintain an accurate *hidden* region for the target.

A *pure strategy* σ_i for agent a_i is a function mapping the agent's information set, $I_i(t)$ to the move it should perform at time t. Since changes to agent a_i's observation history occur only at regular time intervals, $\sigma_i(I_i(t))$ should specify a trajectory f for agent a_i to follow from time t until the next update occurs to $\mathcal{O}_i$. Trajectory f is feasible for a_i at time t if and only if $f(t)$ is equal to $f_i(t)$ and $\forall_{j,k}[(t \leq t_j \leq t_k) \rightarrow f(t_k) \in R_i(f(t_j), t_k - t_j)]$.

A strategy profile $\vec{\sigma} = (\sigma_0, \sigma_1, \ldots \sigma_n)$ assigns a single pure strategy to each agent. Since the game is deterministic, $\vec{\sigma}$ should produce a unique history $h(\vec{\sigma})$ at the end of the game. The expected value of profile $\vec{\sigma}$ is $E(\vec{\sigma}) = u(h(\vec{\sigma}))$ where $u(h)$ is the size of the region guaranteed to contain the target based on the trackers' observation histories at the end of a game with history h. This value can be computed given the observation history $\mathcal{O}_i(h)$ generated by history h,

$$u(h) = |\bigcap_{i=1}^{n} R_0^+(\mathcal{O}_i(h), t_{end})| \quad (6)$$

The utility for the tracker team is $-E(\vec{\sigma})$, meaning the highest possible utility is zero, which happens when the target is directly observable at the end of the game. We leave the objective function for the target undefined, but set out to maximize $-E(\vec{\sigma})$ under a *worst-case* assumption: i.e. we assume that the target will always pick a strategy that minimizes the trackers' utility. This is equivalent to playing a zero-sum game against an opponent that always chooses the *best-response* to the player's strategy.

LEL Heuristic

LEL is based on the *RLA* heuristic introduced in (Raboin et al. 2010). It works by estimating how large the *hidden* region will be in the game's future if a tracker agent follows a particular trajectory. The *hidden* region is the set of locations where the target could be located based on the information provided in an agent's observation history. The size of this region at the end of the game is equivalent to the tracker team's utility, as defined in equation 6.

Given observation history $\mathcal{O}_i$, the target is guaranteed to be located somewhere in $R_0^+(\mathcal{O}_i, t)$ at time t. The region visible to the tracker team is bounded by

$$V^+(\mathcal{O}_i, t) = \bigcup_{j=1}^{n} V_j(R_j^+(\mathcal{O}_i, t)) \quad (7)$$

which contains every location that a tracker agent could observe at time t, given any trajectory consistent with $\mathcal{O}_i$. If the target is not visible at time t, then agent a_i can approximate the *hidden* region where it may be located by computing

$$R_0^+(\mathcal{O}_i, t) \setminus V^+(\mathcal{O}_i, t) \subseteq \mathit{hidden}_0(\mathcal{O}_i, f_i, t)$$

which is the set of locations that the target can reach by time t that are guaranteed to be unobservable by the trackers. This is a subset of the actual *hidden* region defined in the Formalism section. The value returned by the *LEL* heuristic is the average size of this region over a given time interval,

$$u_{lel}(\mathcal{O}_i, t, d) = \frac{1}{d}\sum_{k=t}^{t+d} \left|R_0^+(\mathcal{O}_i, k) \setminus V^+(\mathcal{O}_i, k)\right| \quad (8)$$

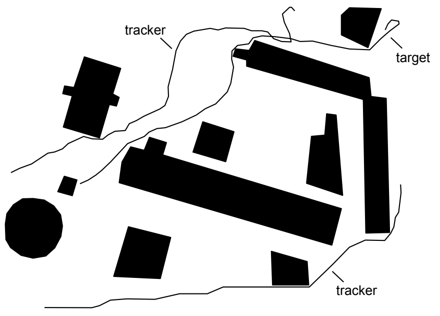

Figure 3: Trajectories generated using the *LEL* heuristic in a domain with two tracker agents and an evasive target. The tracker agents start in the lower left and move past obstacles while attempting to surround the target from both sides.

where t is the current time, and d, the *prediction depth*, is how far into the future to compute the approximation. In the next section is an algorithm to quickly evaluate *LEL*.

To select a trajectory for agent a_i at time t, several candidate trajectories should be sampled using the *LEL* heuristic. Let l' be a possible waypoint for agent a_i and t_k be the desired arrival time. The *LEL* value for this trajectory is $u_{lel}(\mathcal{O}_i + \langle a_i, \{l'\}, t_k\rangle, t_k, d)$. The waypoint with the smallest *LEL* value corresponds to a trajectory where the predicted size of the *hidden* region is the smallest.

If a tie occurs when selecting a trajectory for agent a_i, the tie can be broken by re-computing *LEL* using the location of just one tracker agent. To do this, substitute $V_i(R_i^+(\mathcal{O}_i, t))$ for $V^+(\mathcal{O}_i, k)$ in the heuristic and compute

$$u_{tb}(\mathcal{O}_i, t, d) = \frac{1}{d}\sum_{k=t}^{t+d} \left|R_0^+(\mathcal{O}_i, k) \setminus V_i(R_i^+(\mathcal{O}_i, k))\right| \quad (9)$$

which is what *LEL* would return if there were no other tracker agents on the team. This only needs to be computed in case of a tie, which happens when some subset of the tracker team is able to observe all of the locations that the target can reach. In this case, the tie-breaker ensures that the remaining tracker agents move into a reasonable position.

Algorithm

We assume that the boundary of any region L is a set of polygons in $\mathbb{R}^2$. These polygons can be disjoint and have holes, allowing a close approximation of most regions that will appear in the game (e.g., obstacles, and the *hidden* region).

Computing *LEL* requires a set of reachability functions $\{\mathit{rdist}_0, \mathit{rdist}_1, \ldots \mathit{rdist}_n\}$ where each function $\mathit{rdist}_j(l)$ returns the Euclidean shortest-path distance from agent a_j's location at time t to location l. If a_i does not know the location of agent a_j, then a_i must compute the shortest-path distance from $R_j^+(O_i, t)$, the set of locations guaranteed to contain a_j at time t based on a_i's observation history.

Evaluating $\mathit{rdist}_i(l)$ can be done in logarithmic time using a shortest-path map (Mitchell 1991), but our implementation achieves linear time complexity by using the Fast-Marching Method. This technique is able to compute the Euclidean shortest-path distances for a set of locations in a Cartesian grid, evaluating all possible trajectories, including trajectories that do not pass through the grid points (Sethian 1995). Thus, we can provide a close numerical approximation of *LEL* by evaluating the set of locations L_{raster}, where L_{raster} is a two-dimensional grid of width w and height h. The Fast-Marching Method is able to correctly compute $rdist_i(l)$ for all $l \in L_{raster}$ in time $O(m)$, where $m = w \cdot h$.

Computing *LEL* also requires the visibility functions, $\{vdist_1, vdist_2, \dots vdist_n\}$ where each $vdist_j(l)$ returns the shortest-path distance from agent a_j's location at time t to the nearest location that can observe l, such that $vdist_i(l) = \min_{l' \in V_{poly}(l)} rdist_i(l')$, where $V_{poly}(l)$ is a polygon containing the set of locations visible from l.

Evaluating $vdist_i(l)$ is considerably more challenging than evaluating $rdist_i(l)$, since it requires computing the minimum distance over an set of locations in $V_{poly}(l)$. Rather than computing this explicitly, Algorithm 1 uses a sampling technique to approximate the visibility distance:

Algorithm 1 Approximate agent a_i's visibility map $vdist_i$.

```
$L_{sample}$ = finite subset of $C_{free}$
for all $l \in L_{raster}$
  $vdist_i(l) = \infty$
for all $l \in L_{sample}$
  for all $l' \in (L_{raster} \cap V_{poly}(l))$
    $vdist_i(l') = \min(rdist_i(l), vdist_i(l'))$
```

Since each visible region $V_{poly}(l)$ is polygonal and the points in L_{raster} form a two-dimensional grid, we can compute algorithm 1 efficiently using scan-line rasterization.

Given $\{vdist_1, vdist_2, \dots vdist_n\}$ and $rdist_0$, we can compute the difference in time between when the target can first reach a location and when it can first be seen by one of the trackers. This is evaluated as follows

$$\Delta(x, y) = \min_i \left(\frac{1}{v_i} \cdot vdist_i[x, y] - \frac{1}{v_0} \cdot rdist_0[x, y] \right)$$

where $rdist[x, y]$ and $vdist[x, y]$ correspond to the approximation of $vdist$ and $rdist$ at location $\langle x, y \rangle$ computed in the previous section. Then, the value for *LEL* is

$$u_{lel} = \frac{1}{wh} \sum_{x=0}^{w} \sum_{y=0}^{h} \max(0, \min(\Delta(x, y), d)) \quad (10)$$

To connect this algorithm to the definition of *LEL* in equation 8, note that the size of an arbitrary polygonal region can be approximated by counting how many points in L_{raster} are contained by the polygon. The quality of the approximation depends on the size of the raster, but with any sufficiently large raster we can approximate the size of the hidden region, $R_0^+(\mathcal{O}_i, t) \setminus V^+(\mathcal{O}_i, t)$, and use that to compute *LEL*. However, rather than computing this region at each time step as is done in equation 8, we simply determine when each point in L_{raster} is first intersected by $R_0^+(\mathcal{O}_i, t)$ and $V^+(\mathcal{O}_i, t)$, then use the difference in time to determine how long the point was contained in $R_0^+(\mathcal{O}_i, t) \setminus V^+(\mathcal{O}_i, t)$. That is what is done in equation 10 using our algorithm, allowing us to leverage the Fast-Marching Method and avoid performing costly set operations over complex polygonal regions.

Experiments

To evaluate the algorithm presented in this paper, we performed a series of experiments on randomly generated domains with two-dimensional polygonal obstacles. The starting location for each tracker agent was chosen at random, while the starting location for the target was set to a random location within the trackers' observable region. To make the game more challenging, in all of our experiments we set the target's velocity to be 10% faster than any of the tracker agents, meaning the target could out-run tracker agents.

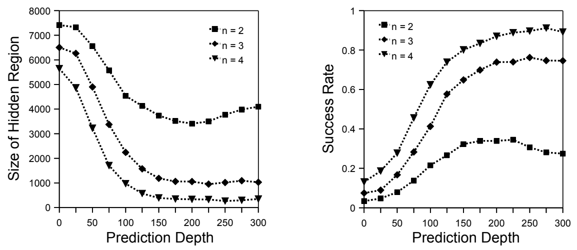

Figure 4: *LEL* performance at various prediction depths, with n trackers per team. Left: *hidden* region size at game's end. Right: proportion of games where target was visible at game's end.

All figures discussed in this section show results from an average of 500 randomly generated trials. For each trial, the score for the tracker team was determined by the size of the *hidden* region at the end of a fixed time period.

We also evaluated the *max-distance* (*MD*) heuristic, a simple hand-coded rule that instructs the tracker team to follow the "shortest-path" to the target. If the target is not visible, the *MD* heuristic will assume the target is as far away as possible and follow the shortest-path to that location. This heuristic provides a baseline comparison for judging the quality of the strategies produced by *LEL*, and has been used for a similar purposed in the past (Raboin et al. 2010).

To generate obstacles for our experiments we used a randomized version of Kruskal's algorithm to create a maze (Kruskal Jr. 1956). We then randomly removed half the walls from the maze to increase the domain's connectivity. The result was a continuous domain with many obstacles for the target to hide behind, but with very few dead-ends. Each trial in our experiments used a different set of randomly generated obstacles and starting locations.

When generating strategies for the target, we assumed that the target always knew the exact location of the tracker team. We used the *LEL* heuristic with a fixed prediction depth to select a trajectory for the target that would minimize the tracker team's utility. Targets using this *worst-case* strategy are much harder to track than targets which simply maximize their distance from the trackers (Raboin et al. 2010).

Tracker success. Fig. 4 shows the average success rate for a team of n trackers using *LEL* with different prediction depths, where "success" means the target is visible at the end of the game. Higher prediction depths decreased hidden region's size at the end of the game, and increased the likelihood of success. With two agents, the heuristic's performance decreased slightly at prediction depths above 200, indicating that the quality of the heuristic function's prediction likely declines beyond a certain depth.

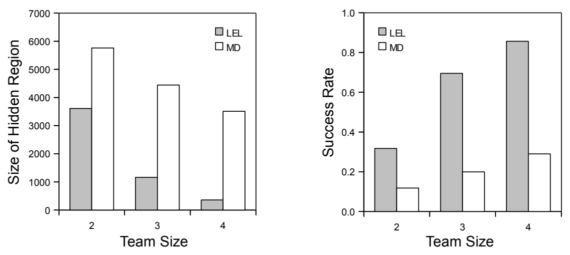

Figure 5: *LEL* and *MD* performance at different team sizes. Left: Average (over 500 games) of the *hidden* region size (in pixels on a 100x100 raster) at end of game. Right: proportion of games where the target was visible at game's end.

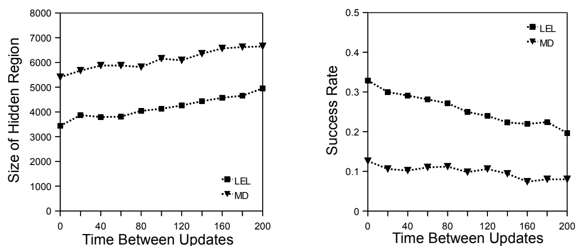

Figure 6: Results for *LEL* and *MD* with interrupted communication. As the time between updates increases, agents are able to communicate less frequently. Left: average size of the *hidden* region at the end of the game. Right: proportion of games where the target was visible at the end.

Fig. 5 shows that teams using *LEL* were over twice as effective as teams using the *MD* heuristic. On average, a 3-agent team using *LEL* was more successful than a 4-agent team using *MD*. In other words, teams using the *LEL* heuristic performed better than teams using the *MD* heuristic, even though the teams using the *MD* heuristic had more agents.

Interrupted communication. Figs. 6 and 7 show the effect of interrupting communication among the trackers. They could communicate only periodically, to report their current location, the target's location in their observation history, and no other information. Hence agents did not know where the other agents on their team were located, only the location provided during the most recent update.

As expected, when the period of time between updates was increased, the trackers became less successful at tracking the target. However, the tracker team still performed surprisingly well when using the *LEL* heuristic, even when updates were spaced apart by large amounts of time. When agents were permitted to communicate as frequently as possible they exchanged information 500 times per game, compared to only 5 times per game when communication was at a minimum. Despite this significant reduction in the frequency with which agents could communicate, agents that generated strategies using *LEL* still out-performed agents that used the *MD* heuristic, even if agents using the *MD* heuristic were allowed constant communication.

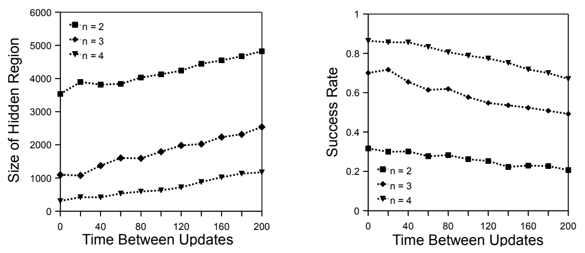

Figure 7: Results for teams of n agents using the *LEL* heuristic with interrupted communication. Left: average size of the *hidden* region at the end of the game. Right: proportion of games where the target was visible at the end of the game.

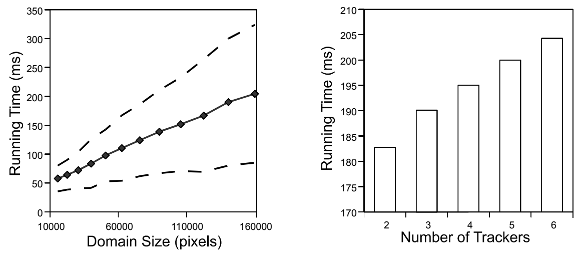

Figure 8: Left: running time in milliseconds for an agent to select its next move using *LEL*. The dashed lines show one standard deviation from the mean. Right: running time for *LEL* using a 400x400 raster, with various team sizes.

Running time. Fig. 8 shows the average CPU time for a single agent to decide which trajectory to follow using the *LEL* heuristic.[1] In these experiments, *LEL* heuristic was evaluated nine separate times per decision, once per waypoint as discussed in the LEL Heuristic section.

The relationship between the average CPU time per decision and the size of the domain (both the number of obstacles and the size of the raster used to compute *LEL*) was approximately linear. The relationship between team size and average CPU time per agent was also approximately linear. For the largest games we evaluated, with six tracker agents and between 700 and 750 obstacle vertices, the average decision time per agent was under half a second.

Discussion. Agents that used *LEL* to select waypoints exhibited very different behavior compared those that followed the shortest path to the target. Using *LEL*, typically one tracker would follow the target closely, while the remaining agents positioned themselves somewhere in the domain that would corner the target. Fig. 3 provides an example of this: tracker a follows the target directly, while tracker b moves along the southern end of the domain to intercept the target if it passes behind any of the obstacles. This kind of "division of labor" is seen often when *LEL* is used, even though each tracker selects its own trajectory independently.

Apparent in Fig. 4, increasing the prediction depth of *LEL*

[1]All experiments were performed using a 2.40 GHz Intel Xeon processor running Java Virtual Machine 6.

is subject to diminishing returns, eventually providing no additional benefit, and in some cases actually hurting performance. This is likely due to the fact that *LEL* will at some point evaluate all the locations in the domain, after which no additional information is provided by searching deeper. This result, and the rate of improvement when compared to *MD*, are both analogous to what was seen when *RLA* was used in the *gridworld* domain (Raboin et al. 2010).

Related Work

LEL can be viewed as an extension of the *RLA* heuristic in(Raboin et al. 2010), which could evaluate strategies for a simple *gridworld* game similar to the problem explored in this paper. *RLA* did not work in continuous Euclidean space, nor was it able to generate strategies when there were interruptions in communication between agents. In addition to *RLA*, there are numerous strategy generation algorithms for related visibility-based pursuit-evasion games, with varying degrees of similarity the game defined in this paper. We summarize some of these approaches below.

Much work on pursuit-evasion games has focused on robot patrolling, or hider-seeker games, where the objective of the tracker is to find an unseen target within some enclosed domain. Graph-based versions of the hider-seeker game have existed for some time (Parsons 1976), and versions of this problem exist in both continuous (Suzuki and Yamashita 1992; LaValle et al. 1997; Gerkey, Thrun, and Gordon 2006; Meng 2008) and discrete domains (Amigoni, Basilico, and Gatti 2009; Basilico, Gatti, and Amigoni 2009; Halvorson, Conitzer, and Parr 2009). This problem has been simplified in the past by assuming the target has unbounded speed (Gerkey, Thrun, and Gordon 2006), or by approximating its movement (Tovar and LaValle 2008). There are also several approaches to the problem of maintaining visibility on the target (Muppirala, Hutchinson, and Murrieta-Cid 2005; Murrieta et al. 2004), but this is a different problem from finding a target that is not visible already.

Conclusion

We presented a formalism and algorithm for generating strategies in multi-agent pursuit-evasion games that occur in partially observable Euclidean space where communication between agents can be interrupted. Our algorithm, using a heuristic method known as *LEL*, is able to generate strategies for a team of tracker agents that are trying to minimize their uncertainty about the location of an evasive target. We have presented experimental results showing that *LEL* was more than twice as likely to maintain visibility on the target when compared to a simple hand-coded strategy that followed the shortest path to the target. We also presented experimental results showing that *LEL* is tolerant of interruptions in communication, continuing to perform well even when communication between agents is infrequent.

Our implementation does rasterization in software using a software-emulated depth buffer. This could be accelerated by doing rasterization in hardware using GPU resources. Modern graphics hardware is designed to perform these operations very quickly, so any implementation that takes advantage of this will most likely show a significant speed-up when compared to the running time of our implementation.

While the worst-case target strategy used in this paper is helpful for determining the minimum performance of tracker strategies, there is no reason to assume that the target will always exhibit worst-case behavior, because the target may not know the tracker agents' locations. Future work could investigate whether *LEL* can be enhanced by using opponent modeling to predict the target's movement, and whether this improves tracker teams' success rates.

Acknowledgments. This work was supported in part by ONR grant N000141210430 and a UMIACS New Research Frontiers Award. The information in this paper does not necessarily reflect the position or policy of the funders.

References

Amigoni, F.; Basilico, N.; and Gatti, N. 2009. Finding the optimal strategies in robotic patrolling with adversaries in topologically-represented environments. In *ICRA-09*.

Basilico, N.; Gatti, N.; and Amigoni, F. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS-09*.

Gerkey, B.; Thrun, S.; and Gordon, G. 2006. Visibility-based pursuit-evasion with limited field of view. *Int. J. Robot. Res.* 25(4):299–315.

Halvorson, E.; Conitzer, V.; and Parr, R. 2009. Multi-step multi-sensor hider-seeker games. In *IJCAI-09*.

Kruskal Jr., J. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.* 7(1):pp. 48–50.

LaValle, S.; Lin, D.; Guibas, L.; Latombe, J.; and Motwani, R. 1997. Finding an unpredictable target in a workspace with obstacles. In *ICRA-97*.

Meng, Y. 2008. Multi-robot searching using game-theory based approach. *Int. J. Adv. Robot. Sys.* 5(4):341 –350.

Mitchell, J. S. B. 1991. A new algorithm for shortest paths among obstacles in the plane. *AMAI* 3:83–105.

Muppirala, T.; Hutchinson, S.; and Murrieta-Cid, R. 2005. Optimal motion strategies based on critical events to maintain visibility of a moving target. In *ICRA-05*.

Murrieta, R.; Sarmiento, A.; Bhattacharya, S.; and Hutchinson, S. 2004. Maintaining visibility of a moving target at a fixed distance: The case of observer bounded speed. In *ICRA-04*.

Parsons, T. D. 1976. Pursuit-evasion in a graph. *Theory and Applications of Graphs* 426–441.

Raboin, E.; Nau, D. S.; Kuter, U.; Gupta, S. K.; and Svec, P. 2010. Strategy generation in multi-agent imperfect-information pursuit games. In *AAMAS-10*.

Sethian, J. A. 1995. A fast marching level set method for monotonically advancing fronts. In *PNAS*, 1591–1595.

Suzuki, I., and Yamashita, M. 1992. Searching for a mobile intruder in a polygonal region. *SIAM J. Comp.* 21:863–888.

Tovar, B., and LaValle, S. 2008. Visibility-based pursuit-evasion with bounded speed. *Int. J. Rob. Res.* 27:1350–1360.