Generating Strategies for Multi-Agent Pursuit-Evasion Games in Partially Observable Euclidean Space

Eric Raboin[1], Ugur Kuter[2], and Dana S. Nau[1]

[1] Department of Computer Science
University of Maryland, College Park, MD 20742 USA
{eraboin, nau}@cs.umd.edu
[2] Smart Information Flow Technologies
211 North 1st Street, Minneapolis, MN 55401 USA
ukuter@sift.net

Abstract. We introduce a heuristic search technique for multi-agent pursuit-evasion games in partially observable Euclidean space where a team of tracker agents attempt to minimize their uncertainty about an evasive target agent. Agents' movement and observation capabilities are restricted by polygonal obstacles, while each agents' 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 show experimentally 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. Experimental results 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.

Keywords: visibility-based pursuit-evasion, multi-agent planning, game theory

1 Introduction

This paper introduces a strategy generation technique for multi-agent pursuit-evasion games 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.

Minimizing uncertainty about a target's location is different from the goal of most other pursuit-evasion formalisms. The tracker team must work both to

maintain visibility on the target, but also to move to strategic locations prior to visibility loss so that recovery will be possible. Past approaches that seek only to maintain visibility on the target for as long as possible [10, 11], or discover the location of a hidden target [15, 7], may not be 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 techniques that are based on deep combinatorial search.

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 [13]. 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.

In this paper we introduce the *Limited-communication Euclidean-space Lookahead* (*LEL*) heuristic, a method for evaluating tracker strategies in games where agents can move freely in two-dimensional Euclidean space and where there may be long periods of time when communication between agents is interrupted.

Our contributions include–

- An algorithm for computing the *LEL* heuristic 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.
- Experimental results showing that our algorithm quickly generates strategies for the continuous domain that are twice as effective at retaining visibility on the target when compared to a strategy that follows the shortest path to the target.

This paper is organized as follows: in the next section we provide a formal description of the multi-agent pursuit-evasion game addressed in this paper, and describe how to generate observation histories for each agent. In sections 3 and 4 we provide a definition for *LEL* and describe our algorithm. Section 5 discusses experimental results for our implementation, including the effect of interruptions in communication. Related work on similar pursuit-evasion games is discussed at the end of the paper.

2 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 goal of the tracker team is to minimize its uncertainty about the target's location by the

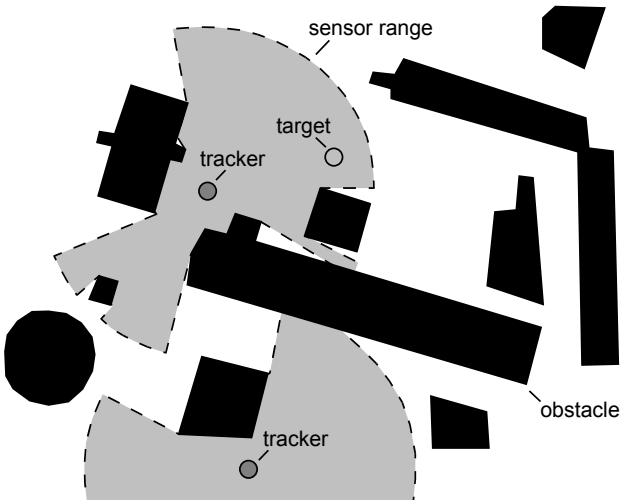

Fig. 1. Example pursuit scenario with two tracker agents and a single target agent. Shaded areas represent the region that can be observed by the tracker agents.

end of the game. Agents' movement and observation capabilities are formally defined below.

2.1 States and Histories

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 $f_i(t)$ denotes 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.

2.2 Reachability and Observability

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

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

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

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

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

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

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 figure 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 $a_{j \neq i}$. Agent a_i only knows the location of the other agents based on the initial state s_0 and its observation history, defined in section 2.3.

2.3 Observation Histories

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 $o \in \Omega$ appears in agent a_i's observation history at time t, then 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.

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

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

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 figure 2.

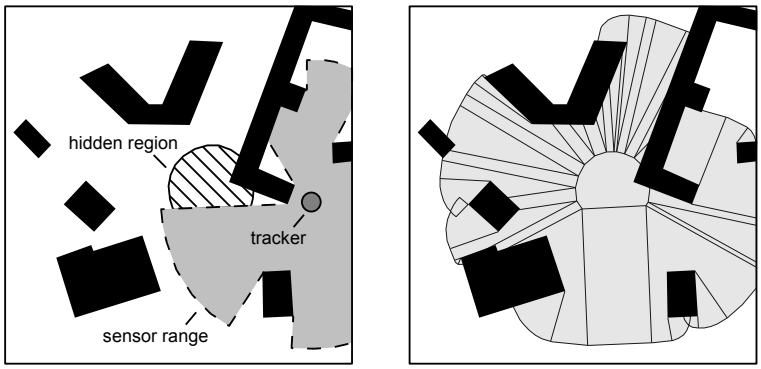

Fig. 2. (left) Example scenario where a single tracker has lost sight of a target. The hidden region is represented as a hatched area. (right) Set of polygons generated by expanding the boundary of the hidden region.

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

$$hidden_j(\mathcal{O}_i, f_i, t) = R_j^+(\mathcal{O}_i, t) \setminus V_i(f_i(t)) \tag{4}$$

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 $\langle a_j, hidden_j(\mathcal{O}_i, f_i, t), t\rangle$ is added to agent a_i's observation history.

Each tracker will receive periodic updates from the other agents on their team. An update from tracker a_j informs the other trackers of a_j's current location, as well as a_j's observation history $\mathcal{O}_j$. This can be merged with a_i's latest observations by computing

$$merge_k(\mathcal{O}_i, \mathcal{O}_j, t) = R_k^+(\mathcal{O}_i, t) \cap R_k^+(\mathcal{O}_j, t) \tag{5}$$

where $\langle a_0, merge_0(\mathcal{O}_i, \mathcal{O}_j, t), t\rangle$ 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 $I_i(t) \subseteq H$, which is 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

$$C(\mathcal{O}_i, f_j) = \langle a_j, L, t\rangle \in \mathcal{O}_i \rightarrow f_j(t) \in L.$$

As with states and histories, the set of all possible information sets at time $t > t_0$ is infinite.

In practice, we only require the most recent observation from each history. This is sufficient both to compute the *LEL* heursitic introduced in section 3 and to maintain an accurate *hidden* region for the target. Therefore, older observations may be discarded.

2.4 Strategies

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 $\boldsymbol{\sigma} = (\sigma_0, \sigma_1, \ldots \sigma_n)$ assigns a single pure strategy to each agent. Since the game is deterministic, each strategy profile $\boldsymbol{\sigma}$ should produce a unique history $h(\boldsymbol{\sigma})$ at the end of the game. The expected value of profile $\boldsymbol{\sigma}$ is

$$E(\boldsymbol{\sigma}) = u(h(\boldsymbol{\sigma}))$$

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,

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

The utility for the tracker team is $-E(\boldsymbol{\sigma})$, 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 $-E(\boldsymbol{\sigma})$ 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.

3 LEL Heuristic

This section introduces the *Limited-communication Euclidean-space Lookahead* (*LEL*) heuristic, which can be used to evaluate trajectories for the tracker team. *LEL* 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.

LEL is inspired by the *relaxed-lookahead* (*RLA*) heuristic, which evaluated trajectories for tracker agents in a *gridworld* version of the game [13]. Trajectories generated using *RLA* were limited to movements between grid locations in one of four cardinal directions. *RLA* also required continuous communication between agents, since all agents were required to know the exact location of the other agents on their team. *LEL* overcomes these limitations by estimating the size of the *hidden* region using only an agent's observation history, as described in section 3.1, and by using a very different algorithm to compute the reachability and visibility information for each of the agents, described in section 4.

Below, we provide a formal definition and algorithm for computing *LEL* in games where agents can move freely over two-dimensional Euclidean space, and where communication between agents can be disrupted. *LEL* requires that each agent has maintained an observation history according to the rules defined in section 2.3.

3.1 LEL Definition

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

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

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 tracker team. This is a subset of the actual *hidden* region defined in section 2.3. The value returned by the *LEL* heuristic is the average size of this region over a given time interval,

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

where t is the current time, and d determines how far into the future to compute the approximation. We will refer to d as the *prediction depth* of the heuristic, since we are estimating the average size of the hidden region out to time d and no further. In section 4 we present an algorithm to quickly evaluate *LEL*.

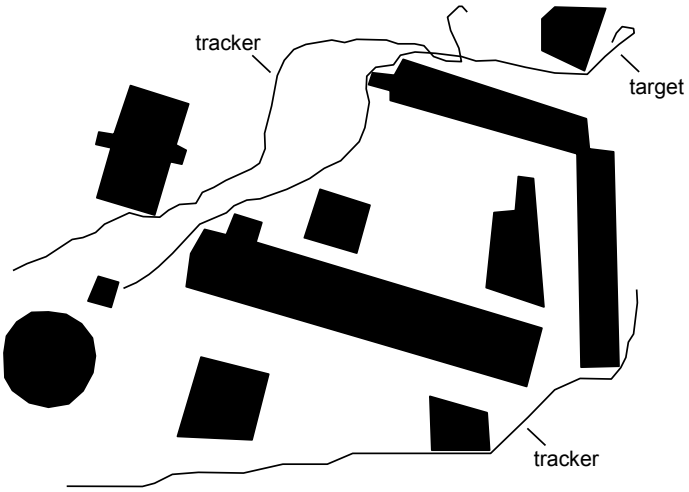

Fig. 3. Trajectories generated using the *LEL* heuristic in a domain with two tracker agents and an evasive target. The tracker agents start in the lower left and move past obstacles while attempting to surround the target from both sides.

3.2 Using LEL to Select Trajectories

To select a trajectory for agent a_i at time t, several candidate trajectories should be sampled using the *LEL* heuristic. Let l' be the location of a possible waypoint for agent a_i and let t_k be the desired arrival time. The value of the *LEL* heuristic for this candidate trajectory is $u_{lel}(\mathcal{O}_i + \langle a_i, \{l'\}, t_k \rangle, t_k, d)$. This value can be compared against other possible waypoints, and the tracker should select the waypoint which returns the smallest *LEL* value, since it 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,

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

which is equivalent to what the *LEL* heuristic 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.

In our experiments in section 5, a total of nine waypoints were evaluated by each agent per update: eight waypoints forming a uniform circle of radius r around the agent's current location, and a single waypoint to represent no movement. While more waypoints could have been evaluated, evaluating only

these nine was sufficient to tell the agent which direction to move. Since our algorithm is able to re-compute the *LEL* heuristic very quickly, we set the arrival time for each waypoint (determined by the radius r) to only a brief moment in the future.

The trajectories in figure 3 were generated by evaluating one extra waypoint per update, created by interpolating the two best waypoints from the original nine. If the *LEL* value for this new waypoint was better than the others, it was chosen instead. This had the advantage of generating slightly smoother trajectories by performing a small additional computation.

4 Algorithm

Our algorithm for *LEL* is divided into two stages: first we generate reachability and observability information for each of the agents in the domain, then we combine that information to compute the *LEL* heuristic. The agent must also have updated its observation history according to the rules described in section 2.3.

We assume that the boundaries of any region L are represented by a set of polygons in $\mathbb{R}^2$. These polygons can be disjoint and have holes in them, allowing a close linear approximation of most regions that will appear in the game. Obstacles in the domain are represented using these polygons, as is the *hidden* region that describes an agent's knowledge of the target.

In this section, we provide an algorithm that quickly computes a numerical approximation of *LEL* by taking advantage of the Fast Marching Method [14], which can quickly compute shortest-path distances over two-dimensional Euclidean space. Since multiple calls to the *LEL* heuristic will result in redundant work, improvements can be made by caching some of the reachability information, a process that is discussed at the end of the section.

4.1 Mapping the Environment

Computing *LEL* requires a set of reachability functions $\{rdist_0, rdist_1, \ldots rdist_n\}$ where each function $rdist_j(l)$ 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 $R_j^+(O_i, t)$, which is the set of locations guaranteed to contain a_j at time t based on a_i's observation history.

Evaluating $rdist_i(l)$ explicitly can be done in logarithmic time using a shortest-path map [9], but our implementation achieves linear time complexity by using the Fast-Marching Method. The Fast-Marching Method 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 [14]. 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, $\{vdist_1, vdist_2, \ldots vdist_n\}$ 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. This is defined in terms of $rdist_i$ below

$$vdist_i(l) = \min_{l' \in V_{poly}(l)} rdist_i(l').$$

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, we can use a sampling technique to approximate the visibility distance, described below in algorithm 1.

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 represented as a polygon, and the points in L_{raster} form a two-dimensional grid, we are able to compute algorithm 1 efficiently by taking advantage of scan-line rasterization techniques. An example visibility map generated by this rasterization process is shown in figure 4.

4.2 Computing LEL

Given $\{vdist_1, vdist_2, \ldots vdist_n\}$ 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

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

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. The value for *LEL* computed by our algorithm is,

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

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.

An example of the *rdist* and *vdist* rasters generated by this technique are shown in figure 4. The per-pixel evaluation of u_{lel} is represented as a composite, shown in the fourth image of the figure. Shaded areas represent locations where the target is able to move before the tracker can guarantee visibility. If there are many such locations, then u_{lel} will return a high value, indicating that the state is poor for the tracker team.

Most of the work performed evaluating a single trajectory can be re-used to evaluate multiple trajectories at once. Since the same observation history is used throughout this process, the reachability and visibility information for most of the agents does not need to be recomputed. Evaluating m possible trajectories for agent a_i involves computing $vdist_j$ and $rdist_j$ only once per agent a_j, while $vdist_i$ is computed m times.

4.3 Caching Critical Points

If the travel distances from each location in the domain are cached in advance, then it is possible to simplify the evaluation of $rdist_i(l_j)$ and $vdist_i(l_j)$. Let L_{obs} be the set of vertex locations along the boundaries of the obstacles in the domain. Let the map $vcache(l_j, l_k)$ be the cached visibility distance from location $l_j \in L_{obs}$ to arbitrary location l_k. If agent a_i is at location l_i, the value of $vdist_i(l_j)$ for all $l_j \in L_{raster}$ can be computed as follows:

Algorithm 2 Compute $vdist_i$ using critical point caching.

for all $l_j \in L_{raster}$
  $vdist_i(l_j) = \infty$
for all $l_j \in (L_{sample} \cap V_{poly}(l_i))$
  for all $l_k \in (L_{raster} \cap V_{poly}(l_j))$
    $vdist_i(l_k) = \min(rdist_i(l_j), vdist_i(l_k))$
for all $l_j \in (L_{obs} \cap V_{poly}(l_i))$
  $d =$ Euclidean distance between l_i and l_j
  for all $l_k \in L_{raster}$
    $vdist_i(l_k) = \min(d + vcache(l_j, l_k), vdist_i(l_k))$

The advantage of using algorithm 2 is that it is no longer necessary to rasterize the individual visible regions $V_{poly}(l)$ for any location l that isn't in the

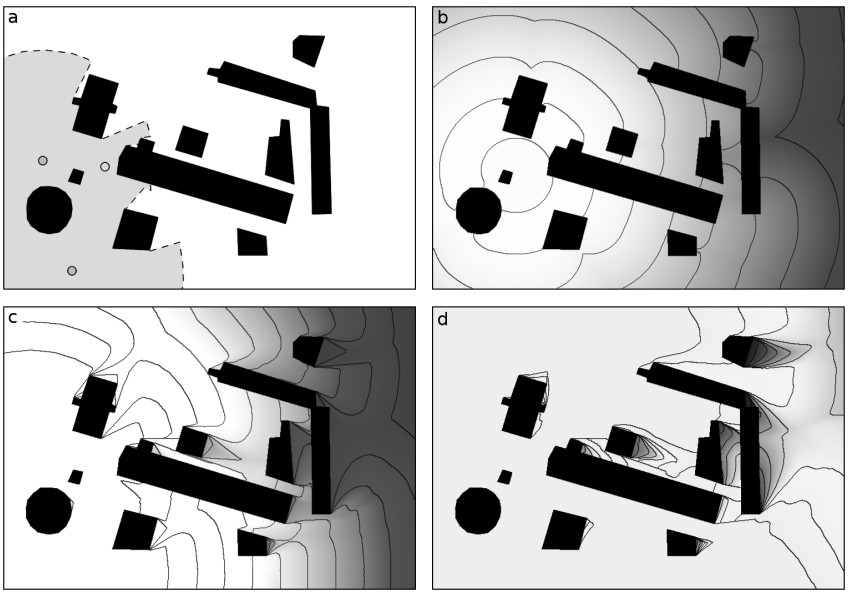

Fig. 4. Steps in the computing *LEL*: a) actual state of the game, b) travel distance from the target region, c) visibility distance from the trackers' locations, d) composite raster used by the *LEL* heuristic.

immediate line-of-sight of agent a_i. Instead, those distances can be determined based on the cached values in $vcache(l_j, l_k)$. As the agents move throughout the domain, the set $L_{obs} \cap V_{poly}(l_i)$ of vertices that are visible to the agent only changes gradually as new vertices become visible and old ones become hidden. Thus, $vcache(l_j, l_k)$ can be computed once when a vertex first becomes visible, and re-used indefinitely until the cache becomes too large or the information is no longer needed.

It's best to use caching when the average number of obstacle vertices visible to an agent is much smaller than the number of points in L_{sample}. If the visibility polygons $V_{poly}(l)$ are pre-computed for each point $l \in L_{sample}$, algorithm 2 has a computational complexity of $O((v + s)m)$ where v is the average number of vertices visible to the agent, s is the average number of points in $L_{sample} \cap V_{poly}(l_i)$, and m is the size of the raster. If v and s are held to be constant, then the complexity is linear in the size of the raster. In our experiments, we compute the distances $vcache(l_j, l_k)$ in advance for all $l_j \in L_{obs}$.

4.4 Computational Complexity

The complexity of *LEL* is $O(n(V+R+m)))$, where n is the number of trackers, R and V are the computational complexity of the algorithms used to generate $rdist$ and $vdist$, and m is the size of the raster. Once $rdist$ and $vdist$ are computed,

computing *LEL* is simply a matter of computing the sum of the time differences between rasters, which can be done in $O(nm)$.

The Fast Marching Method can generate *rdist* in linear time, $O(m)$, where m is the size of the matrix [17]. The algorithm for computing *vdist* has a a complexity of $O((v+s)m)$, as explained in section 4.3. As a consequence, the time required to compute *LEL* increases only polynomially as the size of the raster m or the number of agents per team n increases.

The complexity of other important algorithms are as follows: computing $V_{poly}(l)$ for any single location l is $O(|L_{obs}|)$, or linear in the number of vertices in the domain [5]. Similarly, the vertices visible from polygon L can be computed in $O(p \cdot |L_{obs}|)$, where p is the number of vertices in the boundary of L. Both of these are used in the computation of *vdist*, and can be cached in advance.

5 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.

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.

In addition to evaluating the *LEL* heuristic, 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 [13].

To generate obstacles for our experiments we used a randomized version of Kruskal's algorithm to create a maze [6]. 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 agent, 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* target strategy are much harder to track than targets which simply maximize their distance from the trackers [13], so this is what we have chosen for our experiments.

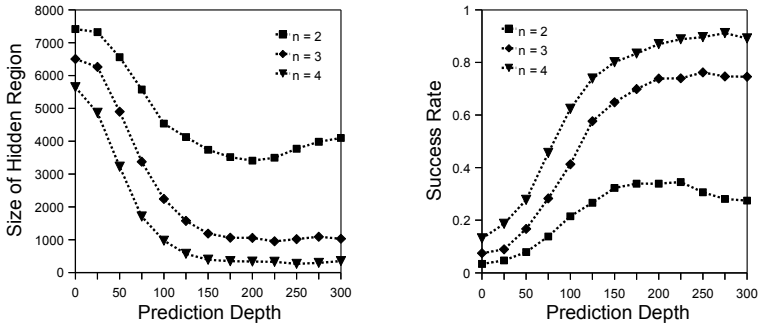

Fig. 5. Performance of *LEL* at different prediction depths given n trackers per team. (left) Average size of the *hidden* region after 500 random games. (right) Proportion of games where the target was visible to at least one tracker at the end of the game.

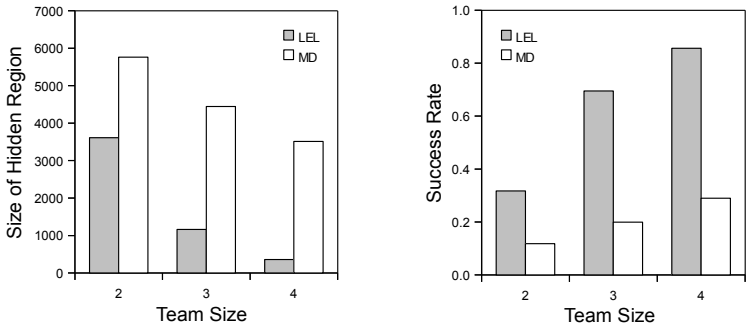

Fig. 6. Comparison of *LEL* and *MD* for different team sizes. (left) Average size of the *hidden* region at the end of 500 randomly generated games. (right) Proportion of games where the target was visible to at least one of the trackers at the end of the game.

5.1 Tracker Success

Figure 5 shows the average success rate for a team of n trackers using *LEL* with different prediction depths, where "success" was determined by whether or not the target was visible at the end of the game. Increasing the prediction depth decreased the size of the hidden region at the end of the game and also increased the likelihood that trackers would retain visibility on the target. In the case with two agents, the performance of the heuristic decreased slightly as the prediction depth increased beyond 200, indicating that the quality of the prediction made by the heuristic likely declines beyond a certain depth.

Figure 6 shows that teams using *LEL* were over twice as effective at tracking the target than those who used the *MD* heuristic. On average, a team of three agents using the *LEL* heuristic were more successful at tracking the target than a team of four agents using the *MD* heuristic. In other words, teams using the

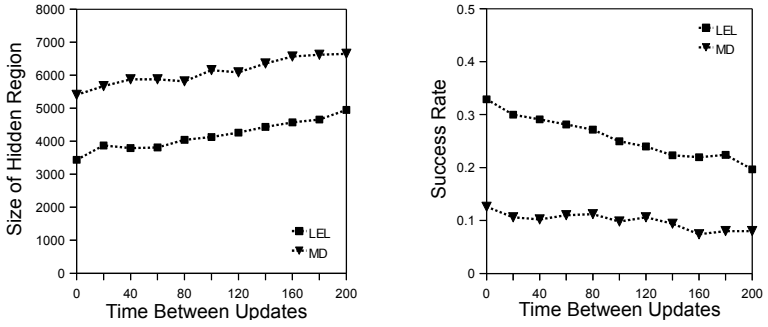

Fig. 7. Results for *LEL* and *MD* when communication is interrupted. As the time between updates increases, agents are able to communicate less frequently. (left) Average size of the *hidden* region at the end of the game. (right) Proportion of games where the target was visible at the end.

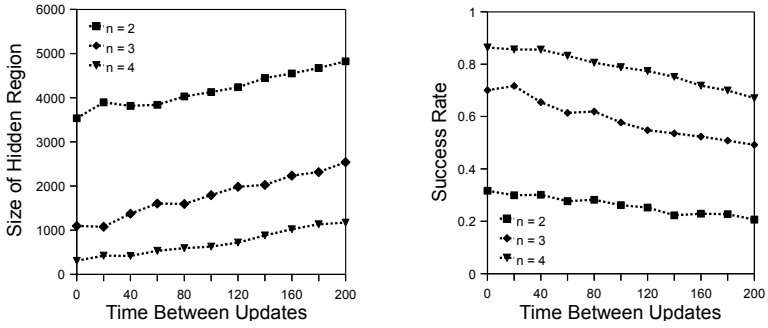

Fig. 8. Results for teams of n agents using the *LEL* heuristic when communication is interrupted. (left) Average size of the *hidden* region at the end of the game. (right) Proportion of games where the target was visible at the end of the game.

LEL heuristic performed better than teams using the *MD* heuristic, even though the teams using the *MD* heuristic had more agents.

5.2 Interrupted Communication

Figures 7 and 8 show the effect of interrupting communication between the tracker agents. In these experiments, agents were permitted to communicate only periodically, during which time they exchanged information about their current location and any knowledge of the target's location in their observation history. Outside these periodic exchanges, no additional information was passed between agents; this means 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

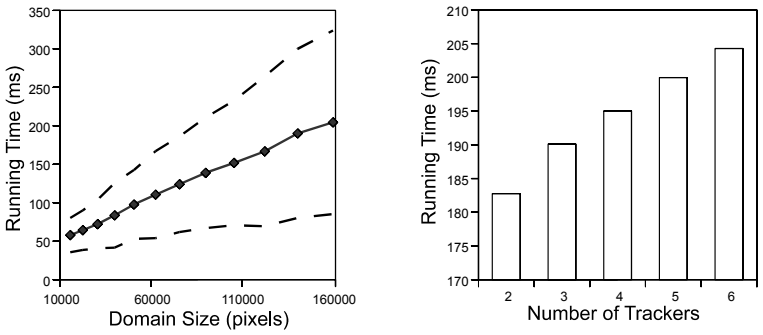

Fig. 9. (left) Running time in milliseconds for an agent to select its next move using the *LEL* heuristic. The dashed lines show one standard deviation from the mean. (right) Running time for *LEL* using a 400x400 raster given different team sizes.

still performed surprisingly well when using the *LEL* heuristic, even when updates were spaced apart by large amounts of time. To put these results into context, 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.

5.3 Running-Time

Figure 9 shows the average CPU time for a single agent to decide which trajectory to follow using the *LEL* heuristic[3]. In these experiments, *LEL* heuristic was evaluated nine separate times per decision, once per waypoint as discussed in section 3.2.

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.

5.4 Discussion

Agents that used *LEL* to select waypoints exhibited very different behavior when compared to agents that simply followed the shortest path to the target. Using *LEL*, typically one tracker would follow the target closely, while the remaining

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

agents positioned themselves somewhere in the domain that would corner the target. Figure 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 figure 5, increasing the prediction depth of *LEL* 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 [13].

6 Related Work

Introduced in earlier work, the *relaxed look-ahead* (*RLA*) heuristic was capable of evaluating strategies for a simple *gridworld* game similar to the problem explored in this paper [13]. *RLA* was not able to generate strategies 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.

A considerable amount of 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 [12], and versions of this problem exist in both continuous [15, 7, 3, 8] and discrete domains [1, 2, 4]. This problem has been simplified in the past by assuming the target has unbounded speed [3], or by approximating its movement [16]. There are also several approaches to the problem of maintaining visibility on the target [10, 11], however this is a different problem from finding a target that is not visible already.

7 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.

7.1 Future Work

To evaluate how effective the *LEL* heuristic was at generating strategies when communication was interrupted, we ran a series of experiments where only periodic communication was allowed between agents, with updates occurring at a fixed time interval. In a real-world scenario, interruptions to communication might occur at random, or they might even be intentional. Transmitting information between agents can potentially reveal the location of the tracker team to the target that they are trying to follow. Since *LEL* is able to tolerate some interruption in communication, minimizing the amount of communication between tracker agents might provide an advantage against an eavesdropping opponent. Determining when it is actually necessary to communicate is a question worth exploring in the future.

The implementation evaluated in this paper performs rasterization in software using a software-emulated depth buffer. This process could be accelerated by performing 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. In a real-world scenario, the target may not know where the tracker agents are located, and it might even be possible to predict the movement of the target based on an opponent model. Future work could explore different ways to incorporate opponent modeling into *LEL*, which may or may not improve the success rate of tracker teams against various opponents.

8 Acknowledgements

This work was supported, in part, by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037, and by a UMIACS New Research Frontiers Award. The views expressed are those of the authors and do not reflect the official policy or position of the funders.

References

1. F. Amigoni, N. Basilico, and N. Gatti. Finding the optimal strategies in robotic patrolling with adversaries in topologically-represented environments. In *Proc. International Conference on Robotics and Automation (ICRA)*, pages 819–824, Kobe, Japan, May 12-17 2009.

2. N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proc. International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 57–64, Richland, SC, 2009.
3. B. P. Gerkey, S. Thrun, and G. J. Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research*, 25(4):299–315, 2006.
4. E. Halvorson, V. Conitzer, and R. Parr. Multi-step multi-sensor hider-seeker games. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 159–166, 2009.
5. S.-H. Kim, J.-H. Park, S.-H. Choi, S. Y. Shin, and K.-Y. Chwa. An optimal algorithm for finding the edge visibility polygon under limited visibility. *Information Processing Letters*, 53(6):359 – 365, 1995.
6. J. Kruskal, Joseph B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):pp. 48–50, 1956.
7. S. LaValle, D. Lin, L. Guibas, J. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *Proc. International Conference on Robotics and Automation (ICRA)*, pages 737–742, 1997.
8. Y. Meng. Multi-robot searching using game-theory based approach. *International Journal of Advanced Robotic Systems*, 5(4):341 –350, 2008.
9. J. S. B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3:83–105, 1991.
10. T. Muppirala, S. Hutchinson, and R. Murrieta-Cid. Optimal motion strategies based on critical events to maintain visibility of a moving target. In *Proc. International Conference on Robotics and Automation (ICRA)*, pages 3826–3831, 2005.
11. R. Murrieta, A. Sarmiento, S. Bhattacharya, and S. Hutchinson. Maintaining visibility of a moving target at a fixed distance: The case of observer bounded speed. In *Proc. International Conference on Robotics and Automation (ICRA)*, pages 479–484, 2004.
12. T. D. Parsons. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, pages 426–441, 1976.
13. E. Raboin, D. S. Nau, U. Kuter, S. K. Gupta, and P. Svec. Strategy generation in multi-agent imperfect-information pursuit games. *Proc. International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 947–954, 2010.
14. J. A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci*, pages 1591–1595, 1995.
15. I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21:863–888, 1992.
16. B. Tovar and S. M. LaValle. Visibility-based pursuit-evasion with bounded speed. *International Journal of Robotics Research*, 27:1350–1360, 2008.
17. L. Yatziv, A. Bartesaghi, and G. Sapiro. O(n) implementation of the fast marching algorithm. *Journal of Computational Physics*, 212(2):393 – 399, 2006.