ABSTRACT

We describe a formalism and algorithms for game-tree search in partially-observable Euclidean space, and implementation and tests in a scenario where a multi-agent team, called tracking agents, pursues a target agent that wants to evade the tracking agents. Our contributions include—

- A formalism that combines geometric elements (agents’ locations and trajectories and observable regions, and obstacles that restrict mobility and observability) with game-theoretic elements (information sets, utility functions, and strategies).
- A recursive formula for information-set minimax values based on our formalism, and a implementation of the formula in a game-tree search algorithm.
- A heuristic evaluation function for use at the leaf nodes of the game-tree search. It works by doing a quick lookahead search of its own, in a relaxed version of the problem.
- Experimental results in 500 randomly generated trials. With the strategies generated by our heuristic, the tracking agents were more than twice as likely to know the target agent’s location at the end of the game than with the strategies generated by heuristics that compute estimates of the target’s possible locations.

Categories and Subject Descriptors
I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search

General Terms
Algorithms, Theory

Keywords
Visibility-based pursuit-evasion games, multi-agent planning, game tree search

Copyright © 2010, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.
tic search and optimization approaches do not always work due to
the uncertainty that arises from the lack of full observability of
the target and lack of perfect knowledge about the target’s objectives
and its strategy; 3) the pursuer agents must generate plans very
quickly and act on those plans since most pursuit scenarios occur
in real time.

Previous work on pursuit and evasion games has often simplified
these challenges by addressing one of two separate sub-problems:
generating pursuit strategies that maintain visibility on a target in
a perfect information domain [5, 24, 22], or pre-computing patrol
strategies that maximize the likelihood of finding an unseen moving
target [13, 38, 21]. Many of the techniques used to address imper-
fect information either omit game-theoretic considerations, or are
formalized for discrete rather than continuous domains [1, 4].

In this paper, we describe a formalism and algorithms for game-
tree search in partially-observable Euclidean space, to address each
of the challenges outlined above. Our contributions include—

- A new formalism for imperfect-information games in n-
dimensional Euclidean space. The formalism allows us to
specify continuous and discrete geometric characteristics, ob-
structed observability due to solid objects, and game-theoretic
information sets, utility functions, and strategies (Section 2).

- A recursive formula for computing minimax values at each

- A heuristic evaluation function for use at the leaf nodes of the
game-tree search. The heuristic function does a quick looka-

- Experimental results that show our heuristic’s effectiveness in

2.1 World Models

Let \( \mathbb{R}^n \) be an n-dimensional Euclidean space that contains obsta-
cles \( o_1, \ldots, o_k \) modeled as geometric solids.\(^1\) Let \( r = (r_1, \ldots, r_k) \)
be a set of agents controlled by an entity called the tracker,
and \( r_0 \) be a single agent controlled by the target. Let \( T =
(t_0, t_1, \ldots, t_{\text{end}-1}, t_{\text{end}}) \) be a finite sequence of time points
that may be either uniformly or non-uniformly spaced. A trajectory
for \( r_i \) is a function \( \ell_i \) that gives a location \( \ell_i(t) = (x_{i1}, \ldots, x_{in}) \)
f or \( r_i \) at every time point \( t \in [t_0, t_{\text{end}}] \), where \( t_0 \) and \( t_{\text{end}} \) are the starting
and ending times.\(^2\)

Observability and reachability play important roles in deciding
how to move during pursuit tasks. An agent \( r_i \)’s observed region at
location \( p \) is \( V_r(p) = \{ \text{all locations that } r \text{ can observe from } p \} \);
see Fig. 2 for an example. An agent’s reachability at location \( p \)

\[ \text{is defined as } R_r(p, \delta) = \{ \text{all locations } r \text{ can reach from point } p \text{ in time } \delta \} \],
where \( \delta \) is some time interval \( (t_i, t_i+\delta] \subseteq [t_0, t_{\text{end}}] \).
This function describes the movement capabilities of agent \( r \) in
the problem domain.

At any given time \( t_s \), the set of all time points at which the tracker
has observed the target’s location is

\[ \hat{T}_n(t_s) = \{ t \in [t_0, t_s] : \ell_n(t) \in V_r(\ell_i(t)) \cup \ldots \cup V_r(\ell_k(t)) \}, \]
where each \( V_r \) is \( r_i \)’s observed region. Hence the tracker has
observed a partial trajectory \( \ell_n : \hat{T}_n(t_s) \rightarrow \mathbb{R}^n \) for the target,
where \( \ell_n(t) = \ell(t) \) for every \( t \in \hat{T}_n(t_s) \). Given the tracker’s observa-
tions, the set of all possible locations for the target is

\[ L_n(t_s) = \{ \ell_n(t_s) : \ell_n \text{ is a trajectory for } r_0, \text{ and } \ell(t) = \ell_n(t) \text{ for every } t \in \hat{T}_n(t_s) \}. \]

Similarly, if the tracking agents’ trajectories are \( I = (I_1, \ldots, I_k) \) and
the target observes partial trajectories \( \hat{I} = (\ell_1, \ldots, \ell_k) \), then

\[ \hat{L}_n(t_s) = \{ \ell_n(t_s) : \ell_n \text{ is a trajectory for } r_0, \text{ and } \ell(t) = \hat{\ell}_n(t) \text{ for every } t \in \hat{T}_n(t_s) \}; \]

\[ \hat{L} = (\hat{L}_1, \ldots, \hat{L}_k). \]

2.2 Imperfect-Information Game Tree Search

In what follows, we assume the tracker wants to be as sure as
possible of the target’s location at time \( t_{\text{end}} \), hence wants to min-
imize the volume \( \text{Vol}(L_n(t_{\text{end}})) \) of the set \( \hat{L}_n(t_{\text{end}}) \) of possible

2An information set \([26, 27]\) is the game-theoretic analog of a be-

\(^1\)In principle, each \( o_i \) could be any regular, semi-analytic set [16].

\(^2\)For simplicity of presentation we use a fixed ending time here, but
in general it may vary. For example, it could be the time at which a
robotic agent runs out of power and can no longer function.
A mixed strategy is a probability distribution over pure strategies. Let $\sigma_0$ and $\sigma = (\sigma_1, \ldots, \sigma_k)$ be any mixed strategies for the target and tracker. Then the expected utility given $\sigma$ and $\sigma_0$ and $I$ is the expected value $E^\sigma(\sigma, \sigma_0, I) = E[\text{Vol}(L_0(t_{\text{end}})) \mid \sigma_0, \sigma, I]$. The Minimax Theorem [40] does not apply per se, but an approximation of it applies, i.e., there are mixed strategies $\sigma^*$ and $\sigma_0^*$, and a minimax value $E^*[I]$ that corresponds intuitively to the information set $I$’s optimal expected utility, such that

$$E^*[I] \approx \min_{\sigma_0} \max_{\sigma} E[\text{Vol}(L_0(t_{\text{end}})) \mid \sigma_0, \sigma, I] \approx \max_{\sigma} \min_{\sigma_0} E[\text{Vol}(L_0(t_{\text{end}})) \mid \sigma_0, \sigma, I].$$

From this we can show that for any information set $I_t(t_j)$ for the tracker,

$$E^*[I_t(t_j)] \approx \begin{cases} \text{Vol}(L_0(t_j)), & \text{if } t_j = t_{\text{end}}, \\ \min_{m_0} \max_{m} E^*[I_t(t_{j+1}) \mid I_t(t_j), m, m_0], & \text{otherwise}; \end{cases}$$

where $m$ and $m_0$ are probability distributions over the moves available to $r$ and $\tau_0$ in $I_t(t_j)$. Conceptually, Eq. (1) is an analog of the well-known recursive formula for minimax values in perfect-information games such as chess, but generalized to accommodate both geometry and imperfect information.

Eq. (1) can be modified to accommodate limited-depth lookahead and a heuristic evaluation function $h(I)$ that gives an estimate of $E^*[I]$. In particular, if we look ahead to some time point $d \leq t_{\text{end}}$, then

$$E^*[I_t(t_j)] \approx \begin{cases} \text{Vol}(I_r(t_j)), & \text{if } t_j = t_{\text{end}}, \\ h(I_r(t_j)), & \text{if } t_j = t_d < t_{\text{end}}, \\ \min_{m} \max_{m_0} E^*[I_r(t_{j+1}) \mid I_r(t_j), m, m_0], & \text{otherwise}; \end{cases}$$

The above equation provides the basis for the strategy-generation algorithm in the next section.

3. ALGORITHM

This section develops an algorithm to perform a game-tree search with limited-depth lookahead in partially-observable multi-agent tracking problems. Since the number of possible trajectories is infinite in multi-agent tracking problems, and thus the reachability at any particular point may be infinite, we will assume that the problem space has been discretized, so that $R_t(p, \delta_t)$ is finite for all $p$. The geometric-modeling literature gives a variety of decomposition and tessellation techniques for accomplishing this.

A multi-agent tracking problem can be formally described by the tuple $(T, R, V, \ell_0(t_0), I(t_0))$, where $T$ is a set of time points, $R$ and $V$ describe reachability and observability in the problem domain, and $\ell_0(t_0)$ and $I(t_0)$ are the initial locations of the agents. We assume that each of the tracker’s agents share a common information set $I$, removing the need to compute separate game trees for each tracker agent. If one of the trackers spots the target, they will all have that information.

We now describe how to compute the minimax value for a multi-agent tracking problem. Algorithm 1 shows a high-level description of our game-tree search procedure. The goal of our algorithm is to produce a strategy $\sigma(I(t_j))$ that specifies the next move.

Algorithm 1 Depth-limited minimax search for a team of trackers following a single target.

$$\text{minimax}(I, t_j, t_0)$$

if $t_j = t_{\text{end}}$ then return $\text{Vol}(I)$

if $t_j = t_d$ then return $h(I)$

$\alpha = \infty$

for all $I' \in \Gamma(t_j)$ do

$\beta = -\infty$

for all $L_0 \in \Gamma(t_j)$ do

$I' = (I', L_0)$

$\beta = \max(\beta, \text{minmax}(I', t_{j+1}, t_d))$

$\alpha = \min(\alpha, \beta)$

return $\alpha$

for the tracker by using a depth-limited game tree search. Because of the high computational complexity of game-tree search in imperfect-information games, we have modified Eq. (2) to incorporate a “paranoid” model of the target’s behavior, i.e., an assumption that the target always knows the tracker’s location and strategy, and will choose the moves that are worst for the tracker.

In Section 2 we defined $\ell_0(t_i)$ as the location of the target at time $t_i$, and the set $I(t_i) = \{\ell(t_1), \ldots, \ell(t_i)\}$ as the location of the trackers. Additionally, we defined $L_0(t_i)$ as the set of possible target locations in the tracker’s information set. In order to perform a game-tree search, we must be able to compute $\ell_0(t_{i+1}), I(t_{i+1})$ and $L_0(t_{i+1})$ in terms of $t_i$.

The set of possible positions for agent $r$ at time $t_{i+1}$ can be determined by

$$\gamma_r(t_i) = R_t(\ell(t_i), \delta_i)$$

where $\delta_i = (t_i, t_{i+1})$. Regardless of the strategy used by the target, we can now guarantee that $\ell_0(t_{i+1}) \in \gamma_0(t_i)$. We can provide a similar definition for the set of tracker locations, such that $I(t_{i+1}) \in \Gamma(t_{i+1}) = \gamma_{\text{track}}(t_i) = \gamma_2(t_i) \times \ldots \times \gamma_k(t_i)$

Using $\gamma_0(t_i)$ and $\Gamma(t_{i+1})$ we can now plot the possible trajectories of the target and trackers. However, to compute the minimax value for a particular state, we will also need some way to determine $L(t_{i+1})$. To do this, we can start by generalizing reachability to operate on a set of locations

$$\mathcal{R}(L_0(t_i)) = \bigcup_{p \in L_0(t_i)} R_0(p, \delta_i).$$

If a tracker knows the exact location of the target at time $t_i$, then it follows from the above definition that $\mathcal{R}(L_0(t_i)) = \gamma_0(t_i)$. Otherwise, $\mathcal{R}(L_0(t_i))$ represents the “expanded” set of potential target locations at time $t_{i+1}$. In either case, only some of those locations are likely to remain hidden from the trackers.

Intuitively, if the target is not observable at time $t_i$, then it must hold that $L_0(t_i) \cap V_r(\ell(t_i, t_i)) = \emptyset$ for each tracker. For simplicity, we will generalize observability for the set of all tracker locations

$$\mathcal{V}(I(t_i)) = \bigcup_{r=1}^k V_r(\ell(t_i)).$$

Thus, we can make the following determination:

$$L_0(t_{i+1}) = \begin{cases} \mathcal{R}(L_0(t_i)) \setminus \mathcal{V}(I(t_{i+1})), & \text{if } \ell_0(t_{i+1}) \notin \mathcal{V}(I(t_{i+1})) \\
L_0(t_{i+1}), & \text{otherwise} \end{cases}$$

We can now update $L_0(t_i)$ for any set of moves selected from $\gamma_0(t_i)$ and $\Gamma(t_{i+1})$. This is sufficient for performing a game-tree search, but some additional work is required to produce a proper strategy.
3.1 Reasoning about a hidden target

Consider any time point \( t_i \) at which the target is hidden (i.e., not observable by the tracker). Recall from Section 2 that the tracker’s strategy \( \sigma \) is a function of the tracker’s information set \( I(t_i) \), which includes (among other things) the set \( L_0(t_i) \) of all possible target locations that are consistent with the tracker’s observations up to time \( t_i \).

At time \( t_i+1 \), the target can stay hidden by remaining somewhere in \( L_{\text{hide}}(t_i+1) = R(L_0(t_i)) \setminus V(t_i+1) \), or it can reveal itself at some location in \( L_{\text{reveal}}(t_i+1) = R(L_0(t_i)) \cap V(t_i+1) \). There are no other possible locations for the target to be at \( t_i+1 \), and the exact location is determined by the target’s strategy. Thus in the game tree, the set of possible “next positions” for the target is

\[
\hat{L}_{\text{target}}(t_i) = \{ p \in L_{\text{reveal}}(t_i+1) \} \cup \{ L_{\text{hide}}(t_i+1) \}
\]

We can now compactly represent the possible moves of the target and tracker using the sets \( \hat{L}_{\text{target}}(t_i) \) and \( L_{\text{track}}(t_i) \) respectively. This also provides a “worst-case” target model that depends only on the tracker’s information set, eliminating a large amount of redundant work from the tree.

The minimax value can be computed recursively (see Algorithm 1) using a heuristic evaluation function \( h(I) \) given some finite search depth \( t_i \). The algorithm’s performance can be improved by incorporating alpha-beta pruning, or other traditional game-tree pruning techniques.

4. HEURISTIC FUNCTIONS

In Algorithm 1’s second if-then statement, the algorithm requires a heuristic evaluation function \( h(I) \) to apply to the leaf nodes of its search. This section defines three different heuristic functions, the \( RS \), \( MD \), and \( RLA \) heuristics, which we will evaluate experimentally later in Section 5.

4.1 Region Size (RS)

The size of \( L_0(t) \) will vary throughout the game, but when it becomes large, it often indicates that the target is in danger of being lost. A simple heuristic is to just compute the current size of \( L_0(t) \), and use that to approximate the minimax value:

\[
h_{RS}(I(t_i)) = |L_0(t_i)|
\]

This region size heuristic is a good approximation of the minimax value only towards the end of the game. Since the heuristic does not use any look-ahead, there is no guarantee that at time \( t_{j+1} \) the evaluation won’t rapidly shift in favor of the target or tracker.

4.2 Maximum Distance (MD)

An even more obvious heuristic is to just measure the distance between each of the trackers and the target. If we compute the trackers’ \( A^* \) distance to each point in \( L_0(t_j) \), we can define

\[
h_{MD}(I(t_j)) = \frac{1}{k} \sum_{p \in L_0(t_j)} \max_{\ell \in L_0(t_j)} |d(\ell, p)|
\]

This heuristic computes the average of the maximum distance that the target could be from each of the trackers at time \( t_j \). When used as part of a single- ply minimax search, each tracker will independently move in the direction of the target, treating it as though it’s as far away as possible.

4.3 Relaxed Lookahead (RLA)

Both of the above heuristics reason about the target’s behavior locally — i.e., by performing a one-step look-ahead — but do not look ahead farther to predict long-term future consequences of the target’s strategies in the game. Because of the geometric properties of a multi-agent tracking problem, it is possible to estimate the size of \( L_0 \) several rounds in advance, without needing to perform a full game-tree search. This allows us the benefit of lookahead, without the requiring an expensive computational effort.

While there may be a number of ways to estimate the size of \( L_0 \) several rounds in advance, we will extend the definition of \( L_0(t_j) \) to include all points reachable by the target \( d \) rounds into the future, excluding those which could be seen by a tracker under any potential circumstance:

\[
L_0(t_j,d) = \{ \ell_0(t_{j+d}) \cup L_0(t_{j+d}) \} \setminus V(I(t_{j+d}) \cap \hat{R}L_{\text{track}}(t_{j+d}))
\]

This relaxation of the problem serves as a very rough approximation of the value of \( L_0(t_{j+d}) \). Using this approximation, we can define a relaxed lookahead heuristic,

\[
h_{RLA}(I(t_j)) = \frac{1}{d+1} \sum_{\ell \in L_0(t_j)} |L_0(t_{j+d})|,
\]

which estimates the average size of \( L_0(t_{j+d}) \) over the next \( d \) rounds. Note there is a special case where \( h_{RLA}(I(t_j)) = h_{RS}(I(t_j)) \).

A computable version of this heuristic is shown in Algorithm 2, which has a runtime complexity that is polynomial in the total number of vertices searched. This algorithm can be accelerated by caching the function \( d(p_1, p_2) \) and a similar function \( v(p_1, p_2) \) to compute the shortest distance from \( p_1 \) that a tracker needs to travel to see \( p_2 \). Each require only a polynomial amount of space to store, but can reduce the computational overhead significantly.

In our implementation of this heuristic, we evaluate each location explicitly, such that \( h_{RLA}(I(t_j)) \) is the sum of

\[
\Delta(t,p, I(t_j)) = \min \max [v(\ell, I(t_j)), d(\ell, p)]
\]

summed over all points \( p \) that the target can reach in \( d \) time steps. To make this computation equivalent to Algorithm 2, we need to restrict the values of \( \Delta(t,p, I(t_j)) \) to the range \( [0,d] \). By doing this, we attribute equal weight to any target locations that are unobservable for \( d \) or more time steps, which may include locations that are not observable at all.

A proof is omitted, but this formulation allows for more efficient use of the cached values, while still producing the same result. In practice, it is also useful to incorporate a tie-breaker in both the region size and relaxed lookahead heuristics: if either heuristic results in a tie, we use the max distance heuristic to determine which is better. This is particularly useful if any of the trackers are isolated, and cannot directly affect the size of \( L_0 \) in the near future.

\footnote{This can be accelerated by computing the \( A^* \) distances in advance, or by caching \( d(p_1, p_2) \) during runtime.}
5. EXPERIMENTAL RESULTS

To evaluate the heuristics presented earlier, we ran a series of experiments in the pursuit-evasion domain illustrated in Fig. 3. Strategies for the tracker agents were computed using the depth-limited minimax search shown in Algorithm 1 with the RS, MD and RLA heuristics. We used a similar algorithm to compute three different approximations of the “worst-case” target model described in Section 3.1: our anti-RS, anti-MD and anti-RLA target models approximate the worst-response to the trackers’ action using a depth-limited minimax search with the corresponding heuristic function.

Results for each experiment were averaged over a set of 500 independent trials, each using randomly-generated locations for a target and two trackers. Each trial lasted for 50 times steps, and performance was measured by the size of \( L_0 \) at the end of the game. In our scenario generation process, we did not consider scenarios that are likely to be unsolvable (i.e., scenarios in which the target was not observable by at least one tracker agent initially), and scenarios that are likely to be trivial (i.e., scenarios in which the target and the tracker agents were positioned too closely).

The following table gives the average running times of game-tree searches using each heuristic, averaged over the same set of trials as in Figs. 5 and 6.

<table>
<thead>
<tr>
<th></th>
<th>2-ply</th>
<th>4-ply</th>
<th>6-ply</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLA</td>
<td>30ms</td>
<td>144ms</td>
<td>2361ms</td>
</tr>
<tr>
<td>RS</td>
<td>15ms</td>
<td>74ms</td>
<td>1069ms</td>
</tr>
<tr>
<td>MD</td>
<td>15ms</td>
<td>78ms</td>
<td>1306ms</td>
</tr>
</tbody>
</table>

Although searching deeper into the game-tree produces strategies that are slightly more successful, the most dramatic results come from changing the heuristic, which requires considerably less computational effort for an equivalent gain in performance. In particular, let us compare a 2-ply search with RLA to a 4-ply or 6-ply search using RS or MD. Of these five searches, the above table shows that the 2-ply RLA search has the smallest running time, and Figs. 5 and 6 show that it also has the best performance.

Example strategy execution. Fig. 7 shows an example of the trajectories generated by our implementation. This is a simple problem with two tracking agents using the \( h_{RLA}^{40} \) heuristic against the corresponding anti-RLA target model.

Notice that, from the beginning, the two trackers split up and attempt to block the target from separate directions. Tracker \( r_1 \), which takes the longer route, has no immediate benefit from moving away from the target. It is only because of the deep lookahead provided by \( h_{RLA}^{40} \) that this occurs. Had the tracker taken a more greedy approach, such as with the max distance heuristic, then it would have missed the opportunity to block an escape route.

In addition to splitting up, tracker \( r_2 \) waits at a key location from time \( t_{18} \) to \( t_{31} \), blocking two paths. This behavior is not specific to RLA, but also appears using the region size heuristic. If the agent leaves that location prematurely, it risks increasing the size of \( L_0 \) from either one direction or the other. In other words, by being impatient, it would allow the target to escape back the way it came.

In practice, not all scenarios play out as well as this one. However, the combination of a short game-tree search, with a long-term predictive lookahead, help explain the performance benefits detailed in the previous sections.

\( ^{7} \)We define a ploy as a single move by either player.

\( ^{8} \)The experiments were run using Java VM version 1.6, on a 2.4GHz processor with 1GB of available memory.

\( ^{9} \)For this example, the values of \( \Delta t \) used to compute the RLA heuristic were restricted to the range \([0,1]\).
Figure 5: Average region size $V(\hat{L}_0(t_{\text{end}}))$ after 50 rounds for RS, RLA, and MD tracker agents versus (a) anti-RS, (b) anti-RLA, and (c) anti-MD target models in the gridworld domain. Each bar is an average of more than 500 trials.

Figure 6: Tracking success rate for RS, RLA, and MD tracker agents versus (a) anti-RS, (b) anti-RLA, and (c) anti-MD target models in the gridworld domain. Each bar is an average of more than 500 trials.

Figure 7: (a) Strategies generated by our algorithm on a simple tracking problem; note how trackers $r_1$ and $r_2$ move to block two escape routes. (b) The tracker’s observable regions $V_1(t_{18})$ and $V_2(t_{18})$ and its set $\hat{L}_0(t_{18})$ of possible target locations, at time $t_{18}$. The circles show the tracking agents’ maximum sensor distances.

6. RELATED WORK

Much of the existing work on visibility-based pursuit-evasion games centers on robot patrolling, or hider-seeker games, where the objective is to find an unseen target moving within some enclosed domain. Versions of this problem exist for both continuous [37, 21, 13, 23, 38] and discrete domains [28, 1, 4, 17] resulting in a variety of different approaches. Distinct from the work in this paper, patrol strategies do not typically specify a pursuit strategy once the target has been found, and thus are often pre-computed.

In a continuous domain, patrol strategies can be computed by tesselating the environment into cells, and modeling the pursuer as a visibility ray or cone [37]. Various algorithms exist to compute the route a pursuer should patrol to detect an evader, or to determine that no solution is possible [21]. In most cases, the problem is simplified by assuming the target has unbounded speed [13], or by approximating its movement [38]. As a result, game-theoretical elements often play a less significant role, although some techniques rely on them more heavily than others [23].

Graph-based versions of the hider-seeker game have existed for some time [28], but have not always modeled observability explicitly. Discrete versions of the visibility-based problem have been addressed using techniques from game-theory, including equilibrium concepts, and non-deterministic strategies [1, 4]. A variant of this problem, where the pursuer is free to observe an arbitrary sub-
set of the domain each round, has been shown to be NP-hard [17]. However, this work is again focused primarily on computing patrol strategies, with the additional limitation of only considering games which are finite.

Substantial work exists on visual tracking of a moving target both with and without obstacles [22, 20, 24], however these techniques do not typically consider imperfect information, and cannot extensively model scenarios where visibility has been lost on the target. Some special cases have been investigated in which the observer keeps a given distance from the evader [24] or in which the observer has bounded velocity [25]. Reinforcement Learning (RL) [36] has also been adapted for this task. For example, [18] studies two cases, where the target is adversarial and non-adversarial, and proposed a RL-based learning rule in order to generate actions for the pursuers depending on the rewards they will get based on both their and the evaders movements in the world. One drawback is that the model proposed by this paper is discrete, i.e., it does not handle continuous environments as the approaches mentioned above.

In robotics, many different approaches have been proposed for planning autonomous robot operations for pursuit and evasion scenarios. A widely-known tool is differential game theory [3], which extends sequential game theory to continuous time. Based on this theory, a pursuit and evasion game is modeled using differential equations with continuous variables that describe the dynamics of the game (e.g., the pursuers’ and evaders’ speeds and the time it takes for a capture). Nash Equilibria have been shown to exist for certain differential game variants of the pursuit-evasion problem [5]. However, differential models are often limited to games where the target remains visible, due to the challenge posed by imperfect information. Furthermore, solving a differential equation system requires significant amounts of computational time and resources in general, and many have argued that it is not feasible for planning actions in real time [23, 39, 34, 12].

Game-tree search has been highly successful in finite perfect-information zero-sum games such as chess [8, 7] and checkers [31], where the minimax theorem [40] applies. The basic idea is to compute approximations of minimax values using a depth-limited version of the minimax algorithm [33]. Numerous techniques have been developed to speed up the computation or make the approximations more accurate; some of the best-known techniques include alpha-beta pruning [19], transposition tables [8], and quiescence pruning [32].

Much work has been done during the past decade to extend game tree search to imperfect-information games such as bridge [35, 15], kriegspiel [27, 30], and poker [14, 6, 9]. One of the biggest problems is computational complexity, which is exponentially worse than in perfect-information games [29]. Some of the more popular approaches for alleviating this complexity involve doing the game tree search over a simplified game that has a smaller search space [6, 14], aggregating sufficiently similar states into groups that are treated as if they were identical [15, 35], and Monte Carlo rollout techniques that search a stochastic sample of the states in the game tree [27].

7. CONCLUSIONS

We have described a formalism that combines geometric and game-theoretic reasoning in imperfect-information multi-agent pursuit games. Based on this formalism, we have presented a recursive formula and search algorithm that are similar to game-tree search in perfect-information games such as chess, but generalized to accommodate both geometry and imperfect information. Our results show how our algorithm can be used to compute strategies that are considerably more sophisticated than simple strategies that just chase after the target.

Our experimental results show how the interplay between the geometric and game-theoretic aspects of our problem domain can be used to compute powerful game-tree-search heuristics. In particular, our RLA heuristic function works by creating a relaxed version of the problem that can be searched very quickly, and doing its own lookahead search in this relaxed problem. The relaxation involves analyzing the tracking agents’ observability ranges at a particular future time point t, ignoring the dependencies among the trackers’ actions, and considering all possible courses of action for the trackers a tracker could take up to time t. In our experimental results on 500 randomly generated problems, the RLA heuristic performed twice as well as two other heuristics: the MD heuristic, which measures the distance from the target, and the RS heuristic, which evaluates the level of uncertainty about the target’s location. The difference in performance was so great that RLA at 2-ply performed better than MD and RS at 4-ply and 6-ply.

Future work. In this paper, we reduced the search algorithm’s time complexity by making a “paranoid” assumption about the target’s behavior. This model has worked quite well in perfect-information games such as chess, but tends to produce strategies that are overly cautious, posing challenges both theoretically [11, 10] and experimentally [27] for imperfect-information games. Consequently, two important tasks will be (1) to develop and test several alternative models of the target’s behavior, and (2) to investigate other ways of reducing the search complexity.

Adapting the algorithm to support mixed strategies would improve the effectiveness of the algorithm, but at the cost of a significantly increased running time. An exploration of this tradeoff could help discover when the paranoid assumption does and does not do well.

Acknowledgments.

This work was supported in part by AFOSR grant FA95500610405, ARO grant W911NF0920072, NGA grant HM1582-10-1-0011, and in part by NSF grant CMMI-0727380. The opinions expressed in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

8. REFERENCES


