A Search-Based Approach to Solve Pursuit-Evasion Games with Limited Visibility in Polygonal Environments

Robotics Track

Alberto Quattrini Li
University of South Carolina
Columbia, South Carolina, USA
albertoq@cse.sc.edu

Raffaele Fioratto
Politecnico di Milano
Milano, Italy
raffaele.fioratto@mail.polimi.it

Francesco Amigoni
Politecnico di Milano
Milano, Italy
francesco.amigoni@polimi.it

Volkan Isler
University of Minnesota
Minneapolis, Minnesota, USA
isler@cs.umn.edu

ABSTRACT

A pursuit-evasion game is a non-cooperative game in which a pursuer tries to detect or capture an adversarial evader. We study a pursuit-evasion game which takes place in a known polygonal environment. The goal of the pursuer is to capture the evader by moving onto its location. The players can observe each others' locations only if they can "see" each other – i.e., if the line segment connecting their locations lies entirely inside the polygonal environment. The complexity of representing the information available to the players at a given time makes solving pursuit-evasion games with visibility limitations difficult. We represent the state of the game using an efficient visibility-based decomposition of the environment paired with a more classical grid-based decomposition. The optimal players' strategies are computed using a min-max search algorithm improved with specific speedup techniques that preserve optimality. We show that our decomposition is complete for a rash evader, which hides from the pursuer and does not move from its hiding location when the pursuer is not visible. Simulations in realistic indoor environments and comparison with a Monte Carlo tree search algorithm validate our approach.

KEYWORDS

pursuit-evasion; limited visibility; min-max search

ACM Reference Format:
Alberto Quattrini Li, Raffaele Fioratto, Francesco Amigoni, and Volkan Isler. 2018. A Search-Based Approach to Solve Pursuit-Evasion Games with Limited Visibility in Polygonal Environments. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018*, IFAAMAS, 9 pages.

1 INTRODUCTION

Pursuit-evasion games model several robotics applications, such as surveillance and search and rescue [4]. Typically, these games involve two players, a *pursuer* and an *evader*. The pursuer (or sometimes a team of pursuers) tries to detect or capture the evader. Due to their practical relevance, there has been significant interest in studying and solving such games in complex environments and with realistic assumptions regarding players' sensing capabilities.

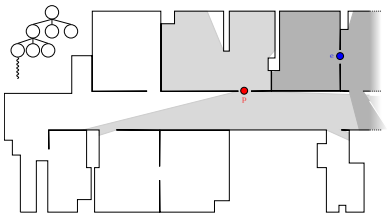

Figure 1: Instance of our setting showing the visibility regions for the pursuer (red dot) and the evader (blue dot) and the abstract representation of the min-max tree used to solve the problem.

The approaches to pursuit-evasion games can be divided into two main categories with respect to the environment: *discrete space*, where the environment is topologically represented as a graph (e.g., [3, 22]), and *continuous space*, where the game happens in a geometric space (e.g., [25]). Furthermore, various assumptions have been made about the capabilities of the players: evader arbitrarily faster than pursuer (e.g., [7]) vs. same bounded velocity for both players (e.g., [28]); full visibility for both players (e.g., [2]) vs. limited and full visibility for pursuer and evader, respectively (e.g., [6]). Moreover, different variants of the goal of the game have been considered. For example, the goal could be to physically capture the evader, namely the position of the evader should coincide with, or be within a certain distance from, the position of the pursuer (e.g., [2]). Other works, like that of [7], consider a game in which the goal is just to detect (find or see) the evader. Another possible goal is to maintain visibility of the evader over time (e.g., [20]).

In this paper, we focus on a pursuit-evasion game in simply-connected polygons (i.e., without internal "holes"), in which the two players have *a priori* knowledge about the environment, the same speed, and line-of-sight visibility: the two players can see each other only if the line segment connecting them lies completely inside the polygon (namely, their visibility is limited by obstacles). These assumptions are realistic when considering for example a practical scenario in which a pursuer and an evader move inside

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

an indoor environment just relying on their vision only. The goal of the pursuer is to capture an evader that tries to actively escape.

We formulate an adversarial search problem to calculate the *optimal* capture path (if any), measured according to the number of time steps. We solve this problem by using an algorithm based on min-max search (see Figure 1 for an example of our problem). The complexity of representing the information available to the players at a given time makes solving pursuit-evasion games with sensing limitations difficult if care is not taken in limiting the size of the state space. Considering a high-resolution regular grid partition, as usually done in robotics path planning, is not always viable due to the high number of states to consider (this is further justified in our simulations). Hence, we use a visibility-based partition of the environment, which is coarser than a uniform grid but captures all relevant visibility information. The partition is built by connecting with a line the mutually visible vertices of the polygon, thus embedding visibility information in the resulting decomposition. In this way, we significantly circumvent the problem of the high computing effort, as the number of cells resulting from the proposed partition is drastically reduced. We show that our discretization is complete for a *rash evader* model in which the evader can hide from the pursuer but does not move from one hiding location to another when the pursuer is not visible. We also present a collection of techniques in order to further reduce the computational effort. Simulations in realistic indoor environments show the effectiveness of our approach, also compared to a Monte Carlo tree search based approach.

2 RELATED WORK

The literature on pursuit-evasion games is vast. The term *visibility-based pursuit-evasion game* is used for games in which one or more pursuers with a visibility sensor try to detect an evader which is arbitrarily faster and whose position is unknown [17]. In contrast, the *lion-and-man game* is played by a pursuer and an evader that have the same speed and global visibility [25]. See [4] for an overview of recent results on these two games. Our problem can be considered as lion-and-man game with visibility constraints.

Detection of an evader with limited visibility is considered in [6]. Stiffler and O'Kane [26, 27] present an algorithm that computes a shortest path to detect the evader in a simply-connected polygon. Experimental results show the effectiveness of the approach. In this work, we consider the problem of not only detecting the evader, but also capturing it.

Regarding the lion-and-man game with limited visibility, Isler et al. [10] present a randomized strategy to solve the problem of detecting an evader by a single pursuer with line-of-sight visibility and the problem of capturing it by two pursuers in simply-connected polygons. The expected time to capture the evader is $O(nT_1^2 + T_2 \cdot (n^2 \ln n))$, where n is the number of vertices and T_1 and T_2 are the time it takes for the two pursuers to travel the diameter of the polygon (i.e., the distance between the two furthest points in the polygon), respectively. The authors also show a modification of the proposed strategy so that a single pursuer can capture the evader, but the expected capture time may significantly increase compared to the case with two pursuers.

The work of [21] deals with the lion-and-man problem, where both have equal speeds and the lion has a line-of-sight visibility. The authors show that the lion can capture the man in any monotone polygon using a deterministic strategy with a capture time of $O(n^7 D^{13})$, where n is the number of vertices of the polygon and D is the diameter of the polygon.

The study in [12] addresses how many pursuers are necessary to capture an evader in an environment with obstacles in the lion-and-man game assuming the visibility-based discrete-time model. In a general hole-free polygon of n vertices, the authors show that in the worst-case $\Theta(n^{1/2})$ pursuers are both necessary and sufficient. In an environment with h holes, the upper bound on the number of pursuers is $O(n^{1/2}h^{1/4})$ for $h \leq n^{2/3}$, and $O(n^{1/3}h^{1/2})$ otherwise.

This body of work provides theoretical guarantees on the problem; however, it does not answer the question: what is the optimal solution (path) to capture an evader in a given environment? In this paper, we devise a search-based approach to answer the question, which reasons on an efficient state-space representation informed by the geometry for a simply-connected polygon. Such an approach can be used tofi nd an optimal path to capture an evader in realistic indoor environments.

Wefi nally note that a method using a min-max search based approach to maintain visibility of an evader who actively avoids tracking with a team of pursuers on a graph is proposed in [18]. In contrast, we are considering a polygonal environment where the pursuer not only has to detect the evader but needs to physically capture it.

3 PROBLEM FORMULATION

In this paper, we study the following pursuit-evasion game between a pursuer and an evader. It takes place in a two-dimensional simply-connected[1] polygon $\mathscr{P}$, which is *a priori* known to both the players. We consider them as points in the plane. This is without loss of generality for translating robots, if we enlarge the polygon boundary to account for the real size of the two players, as usually done in path planning [16].

We assume that time t is discretized and players move in turns of a unit time step (e.g., 1 s). We denote $p(t)$ and $e(t)$ as the pursuer's and the evader's location at time step t, respectively. Both the pursuer and the evader have a maximum velocity v, which is the longest distance they can travel in a unit time step. In each turn, one of the players can move, in general to arbitrary locations according to its velocity. (Note that our formal model is turn-based, but in reality both players can move at the same time.)

Both players have line-of-sight visibility, namely, from a point $q \in \mathscr{P}$ they can see all the points that can be joined to q with a line segment that completely lies in $\mathscr{P}$. In this sense, their knowledge of the position of the other player is limited. We call $VR(q)$ the set of all points visible from a point $q \in \mathscr{P}$, the *visibility region*. This implies that at t the pursuer in position $p(t)$ and the evader in $e(t)$ can see each other if and only if $p(t) \in VR(e(t))$ (or, equivalently, if $e(t) \in VR(p(t))$). Given that both players have a limited visibility, at time step t, they have knowledge about the region cleared so far $CR(p(t))$ ($CR(e(t))$), a subset of $\mathscr{P}$. This includes the current

[1] A polygon is *simply-connected* if any simple closed curve inside the polygon can be shrunk to a point.

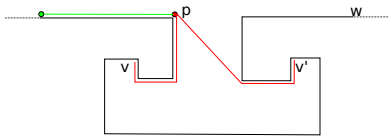

Figure 2: A situation for which, in the worst case, for the evader (red point) there is no gain in being general.

visibility region $VR(p(t))$ ($VR(e(t))$), but also regions that have been cleared at previous time steps and not possibly recontaminated (similarly to the events the work in [7] considers). Thus, we define $E(t) = \{e(t)|e(t) \in \mathscr{P} \wedge e(t) \notin CR(p(t))\}$ as the current knowledge of the pursuer about the possible positions of the evader, when it is not visible. Similarly, we have the same kind of knowledge for the evader over the current possible positions of pursuer: $P(t) = \{p(t)|p(t) \in \mathscr{P} \wedge p(t) \notin CR(e(t))\}$. Clearly, if the two players are visible to each other, we have singletons $E(t) = \{e(t)\}$ and $P(t) = \{p(t)\}$. Note that initially, at $t = 0$, $E(0)$ and $P(0)$ correspond to $\mathscr{P} \setminus VR(p(0))$ and $\mathscr{P} \setminus VR(e(0))$, respectively.

The pursuer wins the game when, at afi nite time t^* (that it tries to minimize), its location is the same as the evader's location (which tries to maximize t^*), namely, $p(t^*) = e(t^*)$. Otherwise, the evader wins the game if it can escape forever (or for a time T representing the duration of the game).

We call *general evader* one allowed to move at every turn: if the two players are not visible to each other, the evader can stay hidden an amount of time (which could be given by the distance between the last known position of the pursuer and the point where the evader disappeared), and then moves to an arbitrary location not in the current visibility region of the pursuer. Another evader model is called *rash evader* model, that allows us to efficiently solve the game in a complete fashion [11]. In this model, the evader can run and hide in an arbitrary hiding location which is not visible by the pursuer. However, the evader does not move from one hiding location to another unless the pursuer becomes visible.

Note that both evader models can be conveniently restricted to going and hiding behind vertices of $\mathscr{P}$. Selecting any other location, different from a vertex, is dominated by choosing one of the vertices of the polygon to hide; that is, for any other location, there exists a vertex where it is better for the evader to hide. The reason is that, given line-of-sight visibility, any other location would either make the traveled distance to reach another shadow area (created by a reflex vertex[2]) where to hide longer or the pursuer would detect the evader earlier. Such an insight motivates the efficient decomposition described in the next section.

In general, the rash model is not unreasonable. Suppose a scenario in which the evader goes around a corner and the pursuer loses sight of the evader. Would it make sense for the evader to hide behind some vertex v for a while and then move to another vertex w before the pursuer shows up (Figure 2)? Intuitively, since the pursuer is not visible throughout this time, the evader could go to w in thefi rst place without the risk of revealing itself during the transition. The rash model captures this intuition as it allows the evader to choose an arbitrary vertex to hide behind but does not allow further movement until the pursuer becomes visible. However, in the above scenario, could it be that there are two paths toward vertices v and v' and, after hiding behind v for some time t, the evader infers that the pursuer moved toward v' and decides to attempt an escape toward w? The answer is not clear, because the pursuer can prevent such inferences by waiting t steps before it commits to v or v'.

At this point, we do not know whether rash strategies are as powerful as general strategies for the evader. In most practical environments, the rash model seems to be as powerful as the general model. For example in cases similar to the one shown in Figure 2, we have the situation described above, in which the pursuer is following the evader along a corridor and then the evader hides in one of the two pockets in the room. By hiding there, as the evader does not have knowledge about the current position of the pursuer, if the evader tries to get again to the corridor, it risks to be seen and possibly captured by the pursuer. As such, in the following, we focus on an algorithm tofi nd the optimal solution to capture a rash evader.

4 SOLVING THE GAME

In this section, we show how our pursuit-evasion game can be formulated as an adversarial search problem with an efficient state-space representation using a visibility-based decomposition of the environment. We show that the decomposition is complete for a rash evader model. The game has two phases: *detection* and *capture*. In thefi rst phase, the pursuer tries to cover the environment with its sensors in order tofi nd the evader (note that this phase is necessary because of the limited visibility). In the second phase, the pursuer tries to reach the same position of the evader. Both phases are formulated as a search problem.

4.1 Formulation as a Search Problem

For a given state-space representation discretizing the environment, the pursuit-evasion game introduced in the previous section can be solved as a search problem. See, e.g., [24, Chapter 5] for further information on this classical approach.

In our case, the state $s(t)$ is a triple $((p(t), E(t)), (e(t), P(t)), cp)$ composed of the current positions $p(t)$ and $e(t)$ of pursuer and evader, of their current knowledge about the opponent $E(t)$ and $P(t)$, and of the current player $cp \in \{p, e\}$ that is playing at turn t.

For now, let us assume that the representation of the state space is powerful enough so that any subset of $\mathscr{P}$ can be represented. The general formulation of our search problem for solving the pursuit-evasion problem is as follows.

Initial state. The initial state $s(0) = ((p(0), E(0)), (e(0), P(0)), p)$ is given by the initial positions of the two players and their initial knowledge about the possible positions of the opponent.

Player. We assume that the pursuer starts the game, and so consequently at an even round the pursuer plays, whereas at an odd round the evader plays.

Actions. From a state $s(t) = ((p(t), E(t)), (e(t), P(t)), cp)$, applicable actions for a player cp are to move to any reachable point $q' \in \mathscr{P}$ (given its maximum speed) and to perceive the environment from the new point q'. A point q' is reachable from $p(t)$ or

[2] A vertex v of a polygon $\mathscr{P}$ is called *reflex vertex* if its internal angle is strictly greater than π.

$e(t)$ if there is a path between the two points that is safe (namely, inside the polygon and not colliding with the boundary of the environment). Player $\overline{cp} = \{p, e\} \setminus \{cp\}$ that does not play in $s(t)$ has no action, but can perceive.

Transition function. The new state of the game resulting from performing an applicable action during the player cp turn "cp moves to q'" is a new state $s(t+1)$, where the position of the player cp is updated to q' and also players' knowledge is updated. Note that the number of applicable actions (and of new states resulting from the transition function) depends on the state-space representation. If it is toofi ne (e.g., by using afi ne-grained grid over the environment) we get too many states; while, if it is too coarse, the players have too few applicable actions, making the representation not complete for the problem.

Terminal test. The terminal test checks whether (a) $p(t) = e(t)$ in a state $s(t) = ((p(t), \{e(t)\}), (e(t), \{p(t)\}), cp)$ or (b) there has been a loop in the path from the initial state to the current state or (c) $t > T$, where T is the maximum time of the game.

Utility. The utility (or payoff) function evaluates the terminal states $s(t)$. If the pursuer wins, it returns $g(s(t))$ (where $g(s(t))$ computes the number of time steps t to reach $s(t)$) and, if the evader wins (in the case of (b) or (c)), it returns $+\infty$.

A solution to the above search problem is afi nite sequence of states $S = \langle s(0), s(1), \ldots, s(t)\rangle$ such that $s(0)$ is the initial state and $s(t)$ is a state that satisfies the terminal test. An optimal solution is a solution S^* that is optimal in terms of the number of time steps t^* (that the pursuer tries to minimize, while the evader tries to maximize).

4.2 Solution of the Search Problem

Given the search problem defined above, the efficiency infi nding a solution is related to the representation of the state space. The key insight to efficiently solve the problem is that we distinguish two phases of the game: (a) when not in direct visibility, the pursuer needs to detect the evader; (b) once found, the pursuer needs to capture the evader. In each phase, we can use a different environment representation derived from a decomposition that partitions the polygon $\mathscr{P}$ in a set of disjoint cells C ($\bigcup_{c \in C} c = \mathscr{P}$) and returns a subset of points $D = \{d \mid d \text{ belongs to a } c \in C\}$. Moreover, for each phase, a search problem (of the type discussed in the previous section) is instantiated.

Given D, derived from a decomposition representing an environment $\mathscr{P}$, we have the following state representation $s_D(t) = ((p(t), E(t)), (e(t), P(t)), cp)$, where $p(t), e(t) \in D$. The actions are represented as follows. From a current point $q \in D$ that belongs to a cell $c \in C$, we have that possible next points $q' \in D$ are in neighbor cells c' that can be reaches with the current speed of the robot in one time step. Note that any representation, however, should guarantee completeness.

In robotics, it is common to tile $\mathscr{P}$ with afi nite regular grid of cells G^e such that each point $q \in \mathscr{P}$ belongs to a cell of the grid. Cells are identical squares with edge e and can be either free or occupied (by the boundaries of the environment). The position d encoded in a state is given by the center of a cell c and the next possible actions are the movements to the center of the neighbor cells.

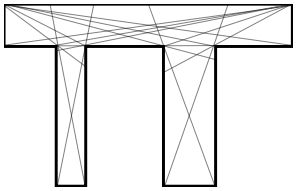

Figure 3: Polygonal decomposition of an environment.

However, depending on e, the number of cells can be really high and this decomposition does not encode any visibility information.

Now, we present an alternative decomposition informed by the geometry of the environment, which allows us to largely reduce the number of possible cells (and so the number of possible actions and states). Specifically, we consider the cells that derive from drawing a line (that we call *inducing line*) between each pair of visible vertices of the polygon and that lies inside the environment (see Figure 3). These lines, by intersecting with other lines and with the boundary of the polygon, induce some cells. We refer to the resulting partition as *visibility-based decomposition.* By construction, this decomposition generates convex cells for which the following result holds.

Proposition4.1. *Let x and y be two points of a cell obtained by the visibility-based decomposition. Let z be a vertex of $\mathscr{P}$. Points x and z are mutually visible if and only if y and z are mutually visible.*

In our representation, when the players are not visible, we use the visibility-based decomposition and we keep track of only the entry and the exit points to and from a visibility-based decomposition cell by discretizing (sampling) the cell boundaries. Thus, the positions d encoded in the state are points along the edges of a cell c. We refer to the resulting state representation as the *detection-phase representation.* Note that at a time step t it is not necessary to reason on all of the cells, but just on those whose inducing lines are not entirely contained in the visibility region of a player. Moving to the inducing lines that lie in one's visibility region does not change the information set of the player. In the corresponding detection-phase search problem, the actions of the pursuer are given by the inducing lines of the cells in which the evader could hide (i.e., that are not yet cleared). Under the rash evader model, the evader does not move during the detection phase.

Instead, when the players are visible to each other, the current knowledge of a player about the opponent is the opponent's actual position. As the game evolves in a turn-based fashion, when determining the possible successor states, a player needs to take into account the set of points reachable from its current position in a time step, according to the velocity v. We call *capture-phase representation* a grid-based decomposition of the environment, where cells c are squares with a given edge length e imposed over the environment and points d are the center of the cells.

To summarize, the search can be divided in two phases: detection and capture. Thefi rst phase uses the visibility-based decomposition, while in the second one the grid decomposition is employed. We can prove that our decompositions preserve the completeness of a search algorithm that uses the corresponding state space.

Theorem4.2. *A strategy to capture the rash evader that hides behind a vertex exists if and only if a complete search algorithm (like min-max) canfi nd a solution in the state space of the detection-phase and capture-phase representations up to a given discretization.*

Proof. *Only if*: This direction is trivial because if the search algorithmfi nds a valid solution, then a capture strategy exists.

If: Now, suppose that there exists a strategy to capture the evader. We must show that a strategy still exists when we restrict the representation of the information available to the players to the detection-phase and capture-phase representations. Then the theorem follows from the completeness of min-max search. Notice that when the players see each other their information state is the opponent's location. In this case, afi ne grid decomposition is used (capture-phase representation) and the solution is complete up to a given resolution level.

When the players are not visible, under the rash model, the pursuer's information can be represented as the set of possible vertices the evader can be hiding behind (namely, the vertices in the areas not yet cleared by pursuer). The evader, when hiding, does not move. What affects its strategy is where the pursuer enters its line of sight. By Proposition 4.1, this event corresponds to crossing a cell boundary in the visibility-based decomposition. Therefore, the detection-phase representation captures it up to discretization of the entrance point. (The idea is that points d that are sampled on the inducing lines deriving from the visibility-based decomposition should not leave enough room to let the evader escape. In particular, the discretization depends on the amount of distance the players can cover in one time step.) Hence, in all cases, the successor states needed to ensure a solution are captured by the detection-phase and capture-phase representations. □

Our visibility-based decomposition of the environment is often used in work that deals with visibility constraints, such as in art gallery or watchman problems. However, the analysis usually performed in those works does not directly apply to our problem. For example, in the art gallery problem [15], it is required that all points are statically covered by guards placed in the environment, and it is not required that an evader is captured by a pursuer that dynamically covers only a portion of the environment.

We use a min-max approach with branch-and-bound as search algorithm, for which we provide a sketch about how it operates. Starting from the initial state, the terminal test is applied. If it is true, then the game ends. Otherwise, the successor states are recursively found in a depth-first way according to the current state and player turn until a leaf node is encountered. When it is encountered, the utility function introduced in the previous section is applied to compute the values for the players, that are propagated upward to the non-leaf nodes. Other branches are then evaluated and possibly pruned if their current value is worse than the one already found (either for the pursuer or the evader). The worst-case computational complexity of our min-max-search-based approach is exponential in the number of time steps needed to reach a terminal state. However, as the results of the next section show, our approach can solve pursuit-evasion games for realistic indoor environments in reasonable time.

To improve more the efficiency of our approach, we introduce some speedup techniques that are expected to reduce the computational effort, preserving the quality of the solutions.

Actions within a time step. Determination of applicable actions distinguishes between two cases: if line-of-sight is not established, only the pursuer perform actions. We can consider only those that lead the pursuer to the adjacent cells derived from the visibility-based decomposition. As such, the time step would be scaled accordingly and added to the current utility, as the rash evader is not performing any action. If the two players can see each other, actions leading to the furthest grid cells that are reachable within a time step are considered, according to the velocity of the players. (This general behavior is combined with a number of special cases, not detailed here, like when the evader moves to a close grid cell and hides behind a vertex.) Differently from considering just actions going to the adjacent grid cells, this reduces the depth of the branches that the search algorithm evaluates at the cost of slightly increasing the branching factor. The optimality is still preserved, as intuitively both pursuer and evader want to get closer or escape further, respectively.

Dominated actions. When the players are mutually visible, we consider a subset of possible actions available to the evader: we discard the actions that would lead to points q where the pursuer is located or that it can reach in one time step (in case no action is available, the evader stays put in its location). The completeness and the optimality are preserved, as if the evader moves to a point that is guarded by the pursuer, then at the next time step it would be captured.

Duplicated states. When selecting a state to expand, duplicates that have been already expanded can be discarded: considering the next player move, a state $s(t) = ((p(t), E(t)), (e(t), P(t)), cp)$ can be safely discarded if $\exists s(t') = ((p(t'), E(t')), (e(t'), P(t')), cp)$ already expanded ($t' < t$) such that $p(t) = p(t')$, $e(t) = e(t')$, and, in the case of the pursuer, $E(t') \subseteq E(t)$ and $g(s(t')) > g(s(t))$, while in the case of the evader $P(t') \subseteq P(t)$ and $g(s(t')) < g(s(t))$ (where $g(s(t))$ is the cost to reach the state $s(t)$ from the initial state $s(0)$). The algorithm is still complete and optimal as duplicated states will eventually lead to the same states already expanded, and those states with a cost greater for the pursuer (lesser for the evader) than the cost of at least one already expanded (duplicated) state are not considered.

Alpha-beta pruning with heuristic. Every node in the search tree carries two values, α and β, representing the best values found so far in a branch for the pursuer and evader, respectively. The pruning condition for a particular branch is triggered whenever $\alpha \geq \beta$, namely when it is discovered that a branch leads to a solution that is worse or equal to a solution already found. Pearl [23] demonstrates that α-β pruning preserves optimality. In the best-case scenario, the worst-case computational complexity can be reduced to $O(b^{\sqrt{d}})$, where b and d are the branching factor and tree depth, respectively. However, the complexity highly depends on the order in which the successor states are expanded. The approach in this work is to sort new states in increasing order of heuristic value when pursuer has to move and vice versa for the evader. This is done in order to select as early as possible the "best" move from the point of view of the moving player according to a heuristic

function $h(s(t)) = d(p(t), e(t))/v$, where $d(\cdot, \cdot)$ is a function that calculates the distance between the two players. The heuristic value encoded in a state can then be used to estimate the depth of the solution. The idea is to imagine that the pursuer is always able to move at full speed while the evader remains on its position the whole time. This estimate is added to the current time step t. If the result is greater than or equal to the time (cost) of the current best candidate solution, the branch is pruned. This operation is safe because this is a best-case scenario with respect to the pursuer (that is also unlikely to happen).

Lemma4.3. *The heuristic function $h(s(t)) = d(p(t), e(t))/v$ is a lower bound on the optimal capture time t^* for the pursuer.*

Proof. Suppose that at time step t the pursuer makes an estimate and completes its turn by making a move. Now is the turn of the evader, and 4 different situations regarding the distance $d(p(t), e(t))$ can occur:

(1) it stays the same, therefore the best-case estimated depth of the solution from the point of view of the pursuer becomes $1 + h(s(t+1)) > h(s(t))$,
(2) it increases, leading to the same case as in the previous one,
(3) it decreases and the pursuer manages to capture its adversary; however, this cannot happen since the evader would discard this action,
(4) it decreases and the pursuer is not able to win, since the evader manages to escape; thus the depth of the solution is certainly higher than the one originally estimated.

As such, $h(s(t))$ is a lower bound on the optimal capture time t^* for the pursuer. □

With such a lemma, we can prove that a branch can be safely pruned according to $h(s(t))$.

Theorem4.4. *The current evaluated branch can be pruned if its current cost $g(s(t))$ summed to $h(s(t))$ is greater than or equal to the cost of the best candidate solution, preserving the optimality.*

Proof. Suppose that a current evaluated branch with $g(s(t)) + h(s(t))$ greater than or equal to the current cost of the best candidate solution is expanded. Given that $h(s(t))$ is a lower bound according to Lemma 4.3, after some iterations, the value would be worse than the best solution found so far. As such, α-β pruning will discard that branch, proving the theorem. □

5 SIMULATION RESULTS

It is not trivial tofi nd methods against which our approach can be fairly compared. For example, some employ multiple pursuers and some work on graphs [8, 14]. In particular, methods working on graphs are not easily applicable to our problem, as metric information about space is lost. Hence, to assess the validity of our approach in solving pursuit-evasion games with limited visibility, we compare our proposed enhanced version of the min-max algorithm with an implementation of Monte-Carlo Tree Search (MCTS) proposed in [5]. The main idea of MCTS is to select the most promising strategy based on the outcome of a high number of *playouts*, consisting in simulating the game until the end by selecting moves at random. In every node there are two values that are updated during execution: one is the ratio between the number of wins and the number of playouts simulated from there; the other one indicates how many times the node has been visited. The algorithm, briefly described here, is composed of four steps that are repeated as long as there is time available for a single iteration.

(1) **Selection**. Recursively apply a selection criteria from the root down to a leaf node, which usually has a low number of simulations played from it and further investigation by the algorithm is necessary. The selection criteria we use is derived from the Multi-Armed Bandit theory [13] and selects the child node i that maximizes the quantity $\frac{R_i}{n_i} + C\sqrt{\frac{\ln n_p}{n_i}}$, where R_i is the algebraic summation of the outcomes of previous playouts that start either from node i or one of its descendants, n_i is the number of visits performed on node i, C is a tunable exploration parameter, by default equal to $\sqrt{2}$ but in general selected empirically based on the problem, and n_p is the number of visits of i's parent node p.

(2) **Expansion.** The selected node is expanded using specific criteria; for instance, the pursuer might prefer to generate a node that allows reducing the distance to its adversary quicker.

(3) **Simulation.** Starting from the newly expanded node, a game is simulated, selecting moves at random. Two situations can occur: a goal node is reached, and either +1 or -1 is returned depending on the player that has turn respectively wins or loses (plus the number of time steps), or the maximum allowed simulation depth is reached, then a tie situation occurs and 0 is returned.

(4) **Backpropagation.** The result of the simulation is propagated up to the root, updating the values of nodes along the path.

Our proposed method and the Monte-Carlo Tree Search approach are run with simulated experiments in several indoor environments available in Radish repository [9]. As such environments are mainly represented with grid maps, we manually converted them to simple polygons. We report here four representative environments (boundaries of environments are considered as obstacles): thefi rst one (*albert-b-laser*) has a unique short corridor with rooms attached to it (Figure 4), the second one (*utk-claxton*) has long corridors (Figure 5), the third one (*fr079*) has a unique long corridor but a complex structure of rooms such that some of them are reachable only through other rooms directly attached to the corridor (Figure 6), and the last one (*ubremen-cartesium*) has a simpler structure characterized by a large open space and some small rooms (Figure 7). The size of the environments is approximately 30 m × 24 m, 140 m × 80 m, 38 m × 13 m, and 41 m × 11 m, respectively (note that the size of the environments has been estimated by considering 10 cm the size of the grid cell, as this information is not reported in the datasets: this seems to be reasonable in terms of average size of rooms). Such environments have been selected as a representative sample of realistic indoor environments without holes.

Table 1 shows the number of cells that derive from imposing a high-resolution grid on these environments and from partitioning the environments with the visibility-based decomposition. It can be noted that the number of cells is greatly abated in all cases, but especially for the second environment, because of the presence of less rooms.

We call a combination of initial positions (that cover the whole environment; see Figures 4–7) and environment a *setting*. For each setting, we run our approach and the MCTS method on a computer

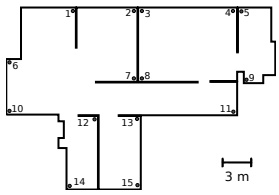

Figure 4: Indoor environment *albert-b-laser* (points indicate the initial positions of the players, also later).

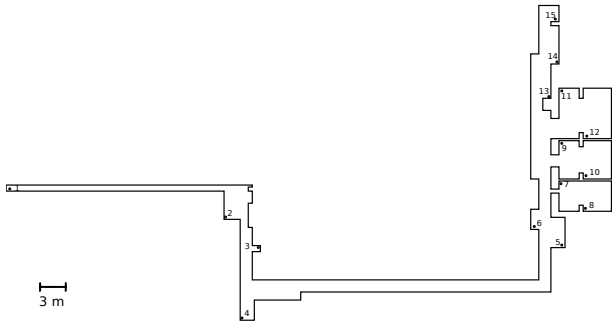

Figure 5: Indoor environment *utk-claxton*.

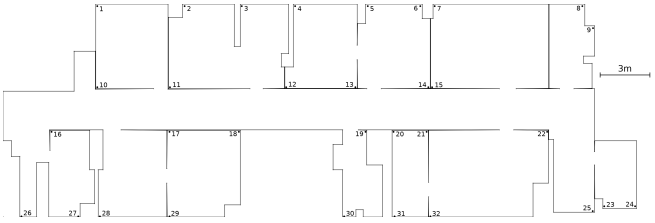

Figure 6: Indoor environment *fr079*.

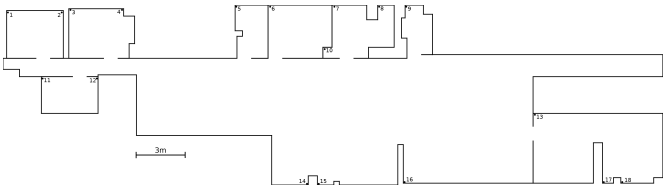

Figure 7: Indoor environment *ubremen-cartesium*.

equipped with a 3.50 GHz i7-4710HQ processor and 8 GB RAM to find the optimal pursuit strategy that guarantees capturing the evader. We have our own implementation of the simulator and the search algorithm in C++ (the state space is generated online), as no generic graph library, like BGL [19], is easily adaptable to our case. We use CGAL library [1] for handling simple polygons and arrangements deriving from the partition of the environment (we use the "exact_predicates_exact_construction_kernel", namely rational arithmetic).

We set a timeout of 5 hours for each run. For all of the runs that find a solution, we report the average and standard deviation of the number of time steps t^*. We consider two maximum velocities for the players: $v = 0.1$ m/s and v that depends on the environment, namely in *albert-b-laser*, *fr079*, and *ubremen-cartesium* $v = 0.5$ m/s, while in *utk-claxton* $v = 1$ m/s.

Environment	Number of grid cells	Number of visibility-based cells
albert-b-laser	71628	4604
utk-claxton	117625	2964
fr079	70000	17950
ubremen-cartesium	96118	48669

Table 1: Number of grid cells (where the size of the cell edge is 10 cm) and number of cells deriving from the visibility-based decomposition.

Map	Speed = 0.1 m/s		Speed > 0.1 m/s	
	Number of time steps	Computing time (s)	Number of time steps	Computing time (s)
albert-b-laser	57.2 (44.0)	172.8 (328.4)	42.7 (31.0)	136.7 (1520.0)
utk-claxton	213.9 (155.3)	568.8 (1121.7)	144.6 (93.2)	210.7 (465.7)
fr079	324.2 (159.9)	786.4 (1256.7)	281.2 (182.2)	231.3 (173.4)
ubremen-cartesium	306 (149.6)	942 (2066.4)	248.5 (164.2)	223.7 (301.7)

Table 2: Results (average and standard deviation over the results for each setting) for the indoor environments under the rash evader model in which both players use min-max.

Map	Low simulation depth and timeout		High simulation depth and timeout	
	Number of time steps	Computing time (s)	Number of time steps	Computing time (s)
albert-b-laser	40.0 (51.7)	926.4 (948.4)	50.1 (34.0)	1543.7 (658.0)
utk-claxton	130.2 (124.3)	1202.1 (526.4)	140.8 (137.3)	1879.2 (538.4)
fr079	250.9 (125.7)	1531.2 (452.0)	270.1 (366.7)	1944.3 (652.2)
ubremen-cartesium	225.9 (149.5)	1286.3 (2064.8)	237.1 (149.6)	1543.4 (2077.5)

Table 3: Results (average and standard deviation over the samples for each map) in both configurations regarding the relevant parameters where both players use MCTS for computing their strategy.

Map	Low simulation depth and timeout		High simulation depth and timeout	
	Number of time steps	Computing time (s)	Number of time steps	Computing time (s)
albert-b-laser	91.3 (44.0)	1858.7 (1893.1)	74.7 (51.4)	4969.0 (2907.5)
utk-claxton	173.3 (101.6)	2135.5 (2036.5)	152.9 (97.2)	2363.9 (1988.8)
fr079	301.3 (186.3)	2221.9 (1715.8)	290.6 (153.6)	3447.7 (1615.6)
ubremen-cartesium	292.2 (170.3)	3004.6 (2457.6)	261.5 (157.2)	5305.1 (2314.4)

Table 4: Results (average and standard deviation over the samples for each map) in both configurations regarding the relevant parameters where only the pursuer use MCTS and evader use min-max.

Overall, considering all the combinations of parameters, pairs of starting points, and algorithms utilized to compute a strategy, the total number of simulation runs made is about 10,000.

Table 2 reports experimental results for the four environments. In all experiments we employed the speedup techniques we presented in the previous section. We actually run some experiments considering a grid decomposition without the speedup techniques. However, most of the simulated runs did not terminate within the timeout and so the results are not reported here, experimentally proving the goodness of the proposed speedup techniques. The values reported in each entry are the average and the standard deviation (in parentheses) over the *npoints* × (*npoints* - 1) combinations of initial positions for the corresponding environment (without considering the same initial position for both players), where *npoints* is the number of starting points defined for each map. The timeout has

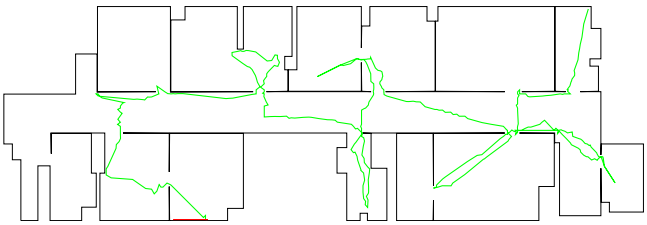

Figure 8: Simulation instance (pursuer and evader paths are in green and red, respectively).

been never reached. In every environment the proposed method is able tofi nd a solution. The relatively high standard deviation could be explained by the fact that the number of time steps to capture the evader is highly dependent on the initial positions of the two players. For example, in thefi rst environment, from positions 12 (pursuer) and 14 (evader), both players can see each other from the start of the game and the evader is basically trapped in that room, while from positions 14 (pursuer) and 9 (evader) the pursuer has to clear a large area before reaching the area where the evader is located. This is reflected also on the computational time forfinding solutions, which varies greatly with the setting: the more time steps required, the higher the computational time. Under the rash evader model it seems that there is a roughly linear relation between them.

Figure 8 shows an instance of the solution found with the starting position of the two players. As can be noted, the pursuer tries to clear one contaminated region after another considering the worst case position of the evader, in such a way that the evader can be trapped and cannot recontaminate the cleared region. Finally, when the evader is found, it is trapped in a corner, as the pursuer is able to guard the line that prevents the evader from escaping that area.

In *utk-claxton* environment, the number of time steps to capture the evader is relatively large because of the presence of very long corridors. If the pursuer starts in the middle, it hasfi rst to go to one side, and, if the evader is not in that area, it has to go to the other side. A similar situation happens in *fr079* and *ubremen-cartesium* because of their complex structure. If speed is increased, both the number of time steps and computational time decrease, because the game is more fast paced and the pursuer is able to close the distance to its adversary and trap it near a corner of the map faster.

When both players use MCTS to compute their strategy, they tend to make suboptimal moves, and the number of time steps is lower than in a min-max context. Table 3 shows the quantitative results relative to velocities $v = 0.5\,\mathrm{m/s}$ in *albert-b-laser*, *fr079*, and *ubremen-cartesium* and $v = 1\,\mathrm{m/s}$ in *utk-claxton*. The computational time increases because the timeout for a single turn of MCTS is often encountered: the algorithm runs several simulations that have a high number of randomly sampled moves before either reaching a terminal state or encountering the maximum allowed depth. This condition is emphasized when both of these parameters are increased. The timeout for a single turn of MCTS for thefirst set of experiments (left hand-side of Table 3 and Table 4, see later) isfi xed to 45 s and the maximum allowed simulation depth to the value of the heuristic function in the state when line-of-sight is established for thefi rst time. In the second set of experiments (right hand-side of the tables), both timeout and maximum simulation depth are doubled. When both players use MCTS (Table 3), both their strategies are suboptimal in general and it is possible that the evader is captured with a shorter path than in the case in which both players used min-max (and both strategies are optimal, Table 2).

The next comparison involves the utilization of MCTS only for the pursuer, whereas the evader uses min-max. Table 4 summarizes the experimental results (with the velocities that depend on the environment, as in Table 3). In this case, only the pursuer is playing sub-optimally, therefore, on average, it needs to make more moves before capturing its adversary. When lower timeout and maximum simulation depth are employed, this situation is more noticeable. Conversely, when both timeout and simulation depth are increased, the number of time steps approaches that of the min-max case. The computing times are higher than those of Table 3 because min-max should be recomputed at every turn for the evader, and it usually lasts more than a single iteration of MCTS.

Overall, the case where both players use min-max seems to provide superior performance with respect to MCTS and, in the realistic indoor environments we considered, it is possible tofi nd a deterministic solution against the rash evader model in a reasonable computing time. A possible explanation could be that, as the environment is highly structured, usually contaminated regions are disjoint, and passages between rooms and corridors are narrow, allowing the pursuer to protect the area cleared so far. Actually, in the environments we considered, recontamination does not happen. In more general environments, where recontamination could happen and the evader can "loop" between two areas that can be recontaminated, our algorithm cannotfi nd equilibrium deterministic solutions. In such a case, a randomization over some of the solutions in necessary, as shown in [10].

6 CONCLUDING REMARKS

In this paper we have presented a method forfi nding the optimal pursuit path in a simply-connected polygon when the pursuer and the evader have line-of-sight visibility. There is no closed-form solution known for this game. Our approach models the pursuit-evasion problem as a search problem andfi nds the optimal solution using a min-max-search-based approach. We presented an efficient representation of the state of the game using a visibility-based decomposition of the environment. We showed that such a decomposition is complete for a rash evader model and we provided some speedup techniques that preserve the optimality of the solution. Simulations, comparing our method with a Monte-Carlo tree search method, showed that the proposed approach can effectivelyfind deterministic solutions in realistic indoor environments for the rash evader model.

Future work will investigate how to further reduce the size of the game tree by, for example, collapsing some cells of the decomposition or using a triangulation that preserves the visibility information but reduces the depth of the resulting tree. Another possibility is to use a hierarchical decomposition. In a broader perspective, building on the insights given by the solution found by our approach, it could be interesting to derive a formal characterization of the kind of environments where a deterministic strategy for a single pursuer guarantees capture no matter what the evader does. Finally, the case of a general evader model should be studied.

REFERENCES

[1] 2014. CGAL - Computational Geometry Algorithms Library. (2014). http://www.cgal.org/

[2] D. Bhadauria, K. Klein, V. Isler, and S. Suri. 2012. Capturing an evader in polygonal environments with obstacles: The full visibility case. *Int J Robot Res* 31, 10 (2012), 1176–1189.

[3] R. Borie, C. Tovey, and S. Koenig. 2009. Algorithms and Complexity Results for Pursuit-evasion Problems. In *Proc. IJCAI*. 59–66.

[4] T. Chung, G. Hollinger, and V. Isler. 2011. Search and pursuit-evasion in mobile robotics - A survey. *Auton Robot* 31, 4 (2011), 299–316.

[5] R. Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proc. CG*. 72–83.

[6] B. Gerkey, S. Thrun, and G. Gordon. 2006. Visibility-based Pursuit-evasion with Limited Field of View. *Int J Robot Res* 25, 4 (2006), 299–315.

[7] L. Guibas, J.-C. Latombe, S. LaValle, D. Lin, and R. Motwani. 1999. A Visibility-Based Pursuit-Evasion Problem. *Int J Comput Geom Ap* 9, 4/5 (1999), 471–494.

[8] G. Hollinger, S. Singh, and A. Kehagias. 2010. Improving the Efficiency of Clearing with Multi-agent Teams. *Int J Robot Res* 29, 8 (2010), 1088–1105.

[9] A. Howard and N. Roy. 2003. The Robotics Data Set Repository (Radish). http://radish.sourceforge.net/. (2003).

[10] V. Isler, S. Kannan, and S. Khanna. 2005. Randomized pursuit-evasion in a polygonal environment. *IEEE T Robot* 21, 5 (2005), 875–884.

[11] V. Isler, S. Kannan, and S. Khanna. 2006. Randomized pursuit-evasion with local visibility. *SIAM J Discrete Math* 20, 1 (2006), 26–41.

[12] K. Klein and S. Suri. 2013. Capture Bounds for Visibility-based Pursuit Evasion. In *Proc. SOCG*. 329–338.

[13] L. Kocsis and C. Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Proc. ECML*. 282–293.

[14] A. Kolling and S. Carpin. 2007. The GRAPH-CLEAR problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *Proc. IROS*. 1003–1008.

[15] A. Kröller, T. Baumgartner, S. Fekete, and C. Schmidt. 2012. Exact solutions and bounds for general art gallery problems. *ACM J Exp Algorithmic* 17, 1 (2012).

[16] S. LaValle. 2006. *Planning Algorithms*. Cambridge University Press.

[17] S. LaValle, D. Lin, L. Guibas, J.-C. Latombe, and R. Motwani. 1997. Finding an unpredictable target in a workspace with obstacles. In *Proc. ICRA*, Vol. 1. 737–742.

[18] V. Lisý, B. Bosanský, and M. Pechoucek. 2012. Anytime algorithms for multi-agent visibility-based pursuit-evasion games. In *Proc. AAMAS*. 1301–1302.

[19] A. Lumsdaine, L.-Q. Lee, and J. Siek. 2002. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley.

[20] R. Murrieta-Cid, R. Monroy, S. Hutchinson, and J.-P. Laumond. 2008. A Complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *Proc. ICRA*. 2657–2664.

[21] N. Noori and V. Isler. 2014. Lion and man with visibility in monotone polygons. *Int J Robot Res* 33, 1 (2014), 155–181.

[22] T. Parsons. 1978. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*, Y. Alavi and Don R. Lick (Eds.). Lecture Notes in Mathematics, Vol. 642. Springer, 426–441.

[23] J. Pearl. 1982. The Solution for the Branching Factor of the Alpha-beta Pruning Algorithm and Its Optimality. *Commun ACM* 25, 8 (1982), 559–564.

[24] S. Russell and P. Norvig. 2010. *Artificial Intelligence: A Modern Approach*. Pearson.

[25] J. Sgall. 2001. Solution of David Gale's Lion and Man Problem. *Theor Comput Sci* 259, 1-2 (2001), 663–670.

[26] N. Stiffler and J. O'Kane. 2012. Shortest paths for visibility-based pursuit-evasion. In *Proc. ICRA*. 3997–4002.

[27] N. Stiffler and J. O'Kane. 2017. Complete and optimal visibility-based pursuit-evasion. *Int J Robot Res* 36, 8 (2017), 923–946.

[28] B. Tovar and S. LaValle. 2008. Visibility-based Pursuit - Evasion with Bounded Speed. *Int J Robot Res* 27, 11-12 (2008), 1350–1360.