2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

A Complete Algorithm for Visibility-Based Pursuit-Evasion with Multiple Pursuers

Nicholas M. Stiffler Jason M. O'Kane

Abstract— **We introduce a centralized algorithm for a visibility-based pursuit-evasion problem in a two-dimensional environment for the case of multiple pursuers. The input for our algorithm is an environment represented as a doubly-connected edge list and the initial positions of the pursuers. The output is a joint strategy for the pursuers that guarantees that the evader has been captured, or a statement that no such strategy exists. We create a Cylindrical Algebraic Decomposition(CAD) of the joint configuration space by using polynomials that capture where critical changes can occur to the region of the environment hidden from the pursuers. Then after computing the adjacency graph for the CAD we construct a Pursuit Evasion Graph(PEG) induced by the adjacency graph. A search through the PEG can produce one of the following outcomes; the search can reach a vertex where the pursuers' motions up to this point ensure that the evader has been captured, or the search terminates without finding a solution and produces a statement recognizing that no solution exists.**

I. Introduction

The *visibility-based pursuit-evasion problem* requires a pursuer to systematically search an environment to locate one or more evaders ensuring that all evaders will be found by the pursuer in a finite time.

Visibility-based pursuit-evasion has many important application areas. In a intruder detection scenario, the evaders are antagonistic and are actively trying to avoid pursuit. In the context of search and rescue the victims can be treated as evaders, and although they are not antagonistic, we still wish to guarantee that all "evaders" are captured, which in this case is equivalent to finding all of the victims. Area patrol is another application of visibility-based pursuit-evasion where a searcher wants to employ a route that can detect the intrusion of an evader.

The specific problem we consider is a variation on the visibility-based pursuit-evasion problem presented in [6] that utilizes a team of pursuers as seen in Figure 1. The pursuers move through a polygonal environment seeking to locate an unknown number of evaders, each of which may move arbitrarily fast. The pursuers have an omni-directional field-of-view that extends to the environment boundary. The goal is to compute a joint strategy for the pursuers, or identify when such a strategy does not exist.

There has been abundant research in the context of the single pursuer visibility-based pursuit-evasion problem that has yielded complete [6], randomized [9], and optimal [22] solutions. The downfall of these techniques centers around the complexity of the environments in which such algorithms

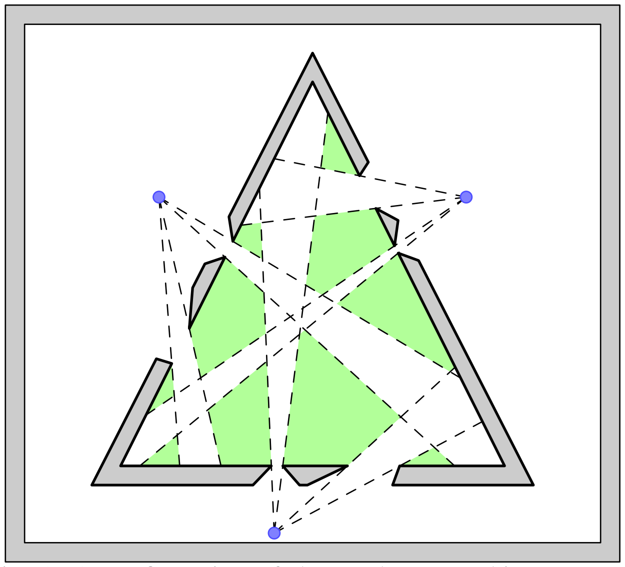


Fig. 1: A configuration of three robots searching an environment. The shaded regions represent areas hidden to the pursuers.

can yield solution strategies for the pursuers. All of the above require the environment to be a simply-connected polygon. This requirement is a limitation because the above utilize only a single pursuer.

When an environment with holes is encountered, a search strategy requires the use of multiple pursuers. In the context of visibility coverage in environments with holes, a provably distributed algorithm was presented that uses robots with an omnidirectional field-of-view but require line of sight communication between one another [17]. Note that while complete visibility coverage does solve the pursuit-evasion problem, it is not a necessary condition. The number of robots required for visibility coverage often exceeds the minimum number of pursuers needed to execute a solution strategy in visibility-based pursuit-evasion.

The main contribution of this work is a complete algorithm for multiple pursuer visibility-based pursuit-evasion that generates a solution strategy for the pursuers to execute through the joint configuration space. Our algorithm is a generalization of the previously-known complete algorithm for the case of a single pursuer [6].

The remainder of this paper is structured as follows: a review of related work (Section II), a formal problem statement (Section III), followed by details describing the area of the environment not visible to any of the pursuers:

- A formal definition for the area not visible to the

N. M. Stiffler and J. M. O'Kane are with the Department of Computer Science and Engineering, University of South Carolina, 301 Main St., Columbia, SC 29208, USA. {stifflen,jokane}@cse.sc.edu

pursues, called *shadows* (Section IV).

- A decomposition of the joint configuration space into *conservative regions* (Section V), that allows a reduction of the problem to a discrete graph search. This decomposition is based on an analysis of the *critical boundaries* (Section VI).

We then provide an algorithm that uses Cylindrical Algebraic Decomposition over these critical boundaries to produce a solution, or to conclude that none exists (Section VII). A synopsis of our algorithm and ideas for future work are found in Section VIII.

II. Related Work

A. Pursuit-evasion

The pursuit-evasion problem was originally posed in the context of differential games [7], [8]. Pursuit-evasion was introduced as graph problem in which multiple pursuers and an evader move from vertex to vertex within the graph until one of the pursuers lies on the same vertex as the evader [18]. The visibility-based pursuit-evasion problem proposed by Suzuki and Yamashita [23] is an extension of the watchman route problem [2], in which the objective is to compute the shortest path that a guard should take to patrol an entire area populated with obstacles, given only a map of the area. A complete solution [6] and an optimal shortest path [22] solution have been found for the single pursuer visibility-based pursuit-evasion problem.

Others have studied scenarios where there are additional constraints on the pursuer, such as a limited field-of-view [5], [15], curved environments [14], an unknown environment [20], or a maximum bounded speed [24].

Multiple pursuer visibility-based pursuit-evasion has been studied when there are sensing and communication constraints on the pursuers. One technique organizes the pursuers into teams, whose joint sensing capability are a set of moving lines, each of which is spanned between obstacles. By using these teams of robots as sweep lines, the authors guarantee detection of the evaders [11]. Another technique uses a distributed algorithm built around maintaining complete coverage of the frontier between cleared and contaminated regions while expanding the cleared region [4].

There are other variants of the pursuit-evasion problem where the pursuers are teams of unmanned aerial vehicles [10]. Beyond this visibility-based formulation there are many different variations of the pursuit-evasion problem. In the lion and man game, a lion tries to capture a man who is trying to escape [16]. In game theory, the homicidal chauffeur is a pursuit evasion problem which pits a slowly moving but highly maneuverable runner against the driver of a vehicle, which is faster but less maneuverable, who is attempting to run him over [8], [19].

B. Cylindrical algebraic decomposition

A *cylindrical decomposition* of $\mathbb{R}^n$ is a partition of the space into cells that are constructible sets, such that the cells in the partition are cylindrically arranged. This means the projection of any two cells onto any lower dimensional space are either equal or disjoint. A *semi-algebraic decomposition* is a partition of $\mathbb{R}^n$ over a set of polynomials into a finite set of disjoint connected regions that are each *sign invariant*. This means that inside of each region, the sign for each polynomial remains constant (negative, zero, positive).

A *cylindrical algebraic decomposition* (CAD) [3] is a cylindrical semi-algebraic decomposition. Collins [3] is the original developer of CAD, and provided an algorithm that takes as input a collection of polynomials in $\mathbb{Q}[x_1 \dots x_n]$ and constructs a sign invariant CAD of $\mathbb{R}^n$.

CAD was originally designed to solve the quantifier elimination problem, but with the advent of a cell adjacency test [1], CAD could be effectively used in other domains, notably motion planning [12], [13], [21].

III. Problem Statement

A. Representing the environment, evaders, and pursuers

1) The environment: The environment is a polygonal free space, defined as a closed and bounded set $F \subset \mathbb{R}^2$, with a polygonal boundary ∂F. The environment is composed of m vertices.

2) The evader: The evader is modeled as a point that can translate within the environment. Let $e(t) \in F$ denote the position of the evader at time $t \geq 0$. The path e is a continuous function $e : [0, \infty) \rightarrow F$, in which the evader is capable of moving arbitrarily fast (i.e. a finite, unbounded speed) within F. Note that, by assuming that there is a single evader, we have not sacrificed any generality. If the pursuers can guarantee the capture of a single evader, then the same strategy can locate multiple evaders, or confirm that no evaders exist.

3) The pursuers: A collection of n identical pursuers cooperatively move to locate the evader. We assume that the pursuers know F, and that they are centrally coordinated. Therefore, from a given collection of starting positions, the pursuers' motions can be described by a continuous function $p : [0, \infty) \rightarrow F^n$, so that $p(t) \in F^n$ denotes the joint configuration of the pursuers at time $t \geq 0$. The function p, which our algorithm generates, is called a *joint motion strategy* for the pursuers. We use the notation $p^i(t) \in F$ to refer to the position of pursuer i at time t. Likewise, $x^i(t)$ and $y^i(t)$ denote the horizontal and vertical coordinates of $p^i(t)$. Without loss of generality, we assume that the pursuers move with maximum speed 1.

Each pursuer carries a sensor that can detect the evader. The sensor is omnidirectional and has unlimited range, but cannot see through obstacles. For any point $q \in F$, let $V(q)$ denote the visibility region at point q, which consists of the set of all points in F that are visible from point q. That is, $V(q)$ contains every point that can be connected to q by a line segment in F. Note that $V(q)$ is a closed set.

When considering the maximal path connected component of $V(q)$, the edges of its boundary are either along ∂F or belong to an occlusion ray.

Definition *An* occlusion ray, $\overrightarrow{qr}$, *is a ray starting at a pursuer position q tangent to a visible environment reflex vertex r.*

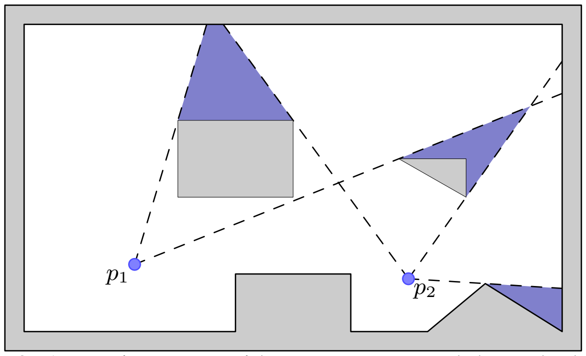


Fig. 2: An environment with two pursuers and three shadows

Informally, an occlusion ray originating at point q is a ray that acts as a boundary separating a visible and non-visible portion of F.

B. Capture conditions

The pursuers' goal is to guarantee the capture of the evader for any continuous evader trajectory.

Definition *A joint motion strategy is a* solution strategy *if, for any evader trajectory $e : [0, \infty) \to F$, there exists some time t and some pursuer i such that $e(t) \in V\big(p^i(t)\big)$.*

C. Algorithm inputs and outputs

In the remainder of this paper, we describe an algorithm for the following problem:

- **Input**: An environment F, represented as a doubly-connected edge list, and a list of n starting pursuer positions $p_1(0), \ldots, p_n(0) \in F$.
- **Output**: A solution strategy for those pursuers in F, or a statement that no such strategy exists.

IV. Shadows

The key difficulty in locating our evader is that the pursuers cannot, in general, see the entire environment at once. This section contains some definitions for describing and reasoning about the portion of the environment that is not visible to the pursuers at any particular time.

Definition *The portion of the environment not visible to the pursuers at time t is called the shadow region $S(t)$, and defined as*

$$S(t) = F - \bigcup_{i=1,\ldots,n} V\big(p_i(t)\big).$$

Note that the shadow region may contain zero or more nonempty path-connected components as seen in Figure 2.

Definition *A* shadow *is a maximal path connected component of the shadow region.*

Notice that $S(t)$ is the union of the shadows at time t. The important idea is that the evader, if it has not been captured, is always contained in exactly one shadow, in which it can move freely.

As the pursuers move, the shadows can change in any of four ways, called *shadow events*.

- *Appear*: A new shadow can appear, when a previously visible part of the environment becomes hidden.
- *Disappear*: An existing shadow can disappear, when one or more pursuers move to locations from which that region is visible.
- *Split*: A shadow can split into multiple shadows, when the pursuers move so that a given shadow is no longer path-connected.
- *Merge*: Multiple existing shadows can merge into a single shadow, when previously disconnected shadows become path-connected.

These events were originally enumerated in the context of the single-pursuer version of this problem [6] and examined more generally by Yu and LaValle [25].

For our pursuit-evasion problem, the crucial piece of information about each shadow is whether or not the evader might be hiding within it.

Definition *A shadow s is called* clear *at time t if, based on the pursuers' motions up to time t, it is not possible for the evader to be within s without having been captured. A shadow is called* contaminated *if it is not clear. That is, a contaminated shadow is one in which the evader may be hiding.*

Notice that, since the evader can move arbitrarily quickly, the pursuers cannot draw any more detailed conclusion about each shadow than its clear/contaminated status; if any part of a shadow might contain the evader, then the entire shadow is contaminated.

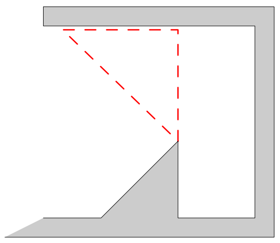
Our algorithm tracks the clear/contaminated status of each shadow. Each time a shadow event occurs, the labels can be updated based on worst case reasoning.

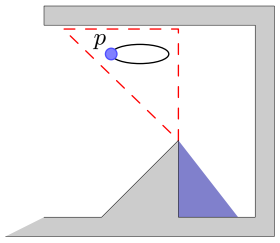
- *Appear*: New shadows are formed from regions that had just been visible, so they are assigned a clear label.
- *Disappear*: When a shadow disappears, its label is discarded.
- *Split*: When a shadow splits, the new shadows inherit the same label as the original.
- *Merge*: When shadows merge, the new shadow is assigned the worst label of any of the original shadows' labels. That is, a shadow formed by a merge event is labeled clear if and only if all of the original shadows were also clear.

Notice in particular that, if all of shadows are clear, then we can be certain the evader has been seen at some point. The result of this reasoning is that we can connect the shadow labels to our goal of finding a solution strategy. A pursuer strategy is a solution strategy if and only if, after its execution, all of the shadows are clear.

V. Conservative Regions

As mentioned above, during the execution of a strategy the pursuers need to identify contaminated shadows. This information depends on the initial positions of the pursuers and their history of past positions. As the pursuers move, the shapes of the shadows change continuously. However, our algorithm only needs to track times at which the shadows change combinatorially. That is, we are only concerned with pursuer movements that generate shadow events, as seen in





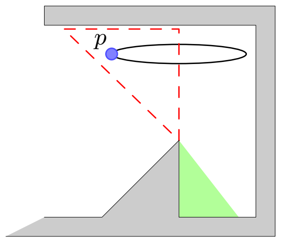


Fig. 3: An illustration of the concept of conservative regions.

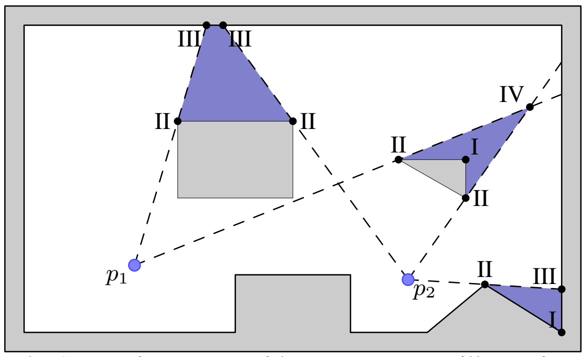


Fig. 4: An environment with two pursuers illustrating the different types of shadow vertices.

Event Types	Critical boundary occurs when...	Details
I-III, II-III, II-IV	pursuer colinear with two ∂F vertices	Sec. VI-A
III-III, III-IV	occlusion rays intersect on ∂F	Sec. VI-B
IV-IV	three occlusion rays share an intersection	Sec. VI-C
I-I, I-II, II-II, I-IV	never	Sec. VI-D

TABLE I: The ten possible shadow vertex merges can be grouped into four general cases.

Figure 3. Our algorithm exploits this idea by partitioning the joint configuration space into regions where no shadow events occur.

Definition *A region $R \subseteq F^n$ is a* conservative region *if any path that remains within R generates no shadow events.*

Given a partition of F^n into conservative regions, the original problem of generating a continuous solution strategy can be reduced to a simpler discrete problem of selecting a sequence of adjacent conservative regions.

VI. Critical Boundaries

In this section, we provide a foundation for dividing F^n into conservative regions—within which shadow events cannot occur—by describing a complete set of *critical boundaries* at which such events *can* occur. Specifically, we examine the four different types of vertices that can compose the boundary of a shadow and establish critical boundaries where those vertices can change. The key idea is that each shadow can be characterized by its set of vertices, and that no shadow events can occur if the vertex set of every shadow region remains unchanged.

The vertices of every shadow can be classified into four types, as shown in Figure 4, which we call Types I, II, III, and IV.

- Type I vertices are environment vertices for which the adjacent edges in the shadow boundary lie along ∂F. Informally, these are vertices of the environment that no pursuer can see.
- Type II vertices are environment vertices, at which one of the two adjacent edges in the shadow boundary lies along ∂F and the other lies along an occlusion ray. Informally, these are vertices that are visible to some pursuer, but that block that pursuer's view of some other part of F.
- Type III vertices are the endpoints of occlusion rays. Each lies on the interior of an edge of ∂F.
- Type IV vertices occur at intersections between occlusion rays.

We use the definition of conservative region from Section V to argue that just by thinking about when two shadow vertices can merge—and the inverse *split* events where a shadow vertex can split into two shadow vertices—we have identified all the ways in which a shadow can change. By definition a region is conservative if it generates no shadow events, which means that the cardinality for the vertex set of the shadow stays the same. It follows that a shadow can only gain or lose shadow vertices when a pursuer crosses the boundary between conservative regions. By describing an exhaustive list of how two shadow vertices can merge at these critical boundary we have identified all the ways in which a shadow can lose vertices. A inverse method of gaining vertices is the result of split events. Note that when a shadow has less than three shadow vertices the shadow disappears, likewise a shadow appears when there are at least three shadow vertices.

The next step is to characterize the sets of joint configurations at which such vertex merges can occur. Considering all pairs of vertex types, there are ten distinct possible types of merges. We'll consider each of these ten cases. Fortunately, the ten cases can be grouped into four general categories that can be analyzed in similar ways. Table I summarizes the merge types.

A. Merges resulting from pursuers colinear with a pair of environment vertices

First, we argue that merge types I-III, II-III, and II-IV occur only when some pursuer is colinear with some pair of environment vertices.

1) I-III merges: Consider the case in which a Type I and Type III vertex merge. This situation requires a vertex of ∂F to be coincident with the endpoint of an occlusion ray ∂F. Figure 5 shows how this can occur. On one side of

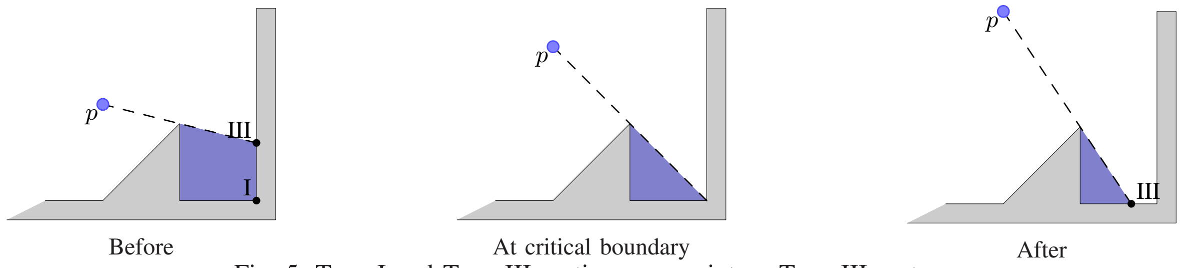


Fig. 5: Type I and Type III vertices merge into a Type III vertex.

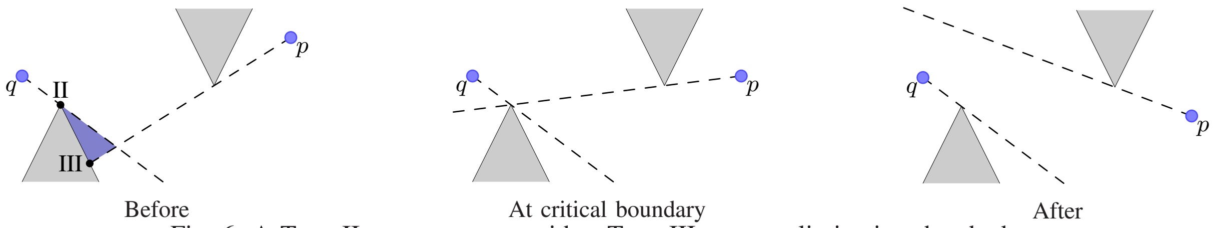


Fig. 6: A Type II vertex merges with a Type III vertex, eliminating the shadow.

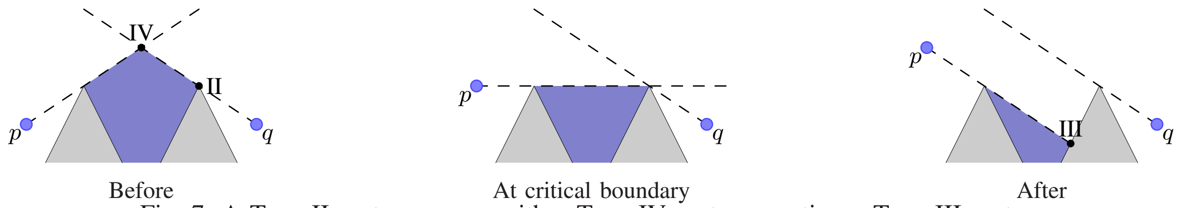


Fig. 7: A Type II vertex merges with a Type IV vertex, creating a Type III vertex.

this boundary, the shadow has a Type I vertex adjacent to a Type III vertex; on the other side, those vertices are replaced with a single Type III vertex.

Specifically, for a Type I vertex at $u = (x_{\rm u}, y_{\rm u})$ and a Type III vertex owned by pursuer $p = (x_{\rm p}, y_{\rm p})$ and induced by occlusion vertex $v = (x_{\rm v}, y_{\rm v})$, this kind of event occurs when

$$\begin{vmatrix} x_{\rm p} & y_{\rm p} & 1 \\ x_{\rm u} & y_{\rm u} & 1 \\ x_{\rm v} & y_{\rm v} & 1 \end{vmatrix} = 0. \quad (1)$$

Treating $x_{\rm u}$, $y_{\rm u}$, $x_{\rm v}$, and $y_{\rm v}$ as constants, this equation expands to a polynomial of degree 1 in the variables $x_{\rm p}$ and $y_{\rm p}$. To form the complete set of critical boundaries of this type, we must iterate over all n choices of pursuers, and all $\binom{m}{2}$ choices for $u = (x_{\rm u}, y_{\rm u})$ and $v = (x_{\rm v}, y_{\rm v})$.

2) II-III merges: For a Type II vertex to merge with a Type III vertex, we must have an occlusion ray of one pursuer colinear with an occluding vertex of another pursuer, as illustrated in Figure 6. This requires a pursuer p to be colinear with the two occluding vertices $u = (x_{\rm u}, y_{\rm u})$ and $v = (x_{\rm v}, y_{\rm v})$. Thus, the critical boundary polynomial is identical to Equation 1; the only difference is that, in this case, both $u = (x_{\rm u}, y_{\rm u})$ and $v = (x_{\rm v}, y_{\rm v})$ must be reflex (i.e. non-convex) vertices.

3) II-IV merges: Likewise, for a Type II vertex to merge with a Type IV vertex, two occlusion rays from two different pursuers must intersect at the occluding vertex of one of those rays. See Figure 7. As in the previous two cases, this can occur only when a pursuer p is colinear with two vertices $u = (x_{\rm u}, y_{\rm u})$ and $v = (x_{\rm v}, y_{\rm v})$ of ∂F, and Equation 1 defines the critical boundary.

4) Number of polynomials: For a fixed pursuer, the total number of critical event polynomials for these three merge types is at most $\binom{m}{2}$, yielding a maximum of $\binom{n}{1}\binom{m}{2}$ polynomials across all n pursuers.

B. Merges resulting from two occlusion rays intersecting on ∂F

Next we consider merge types III-III and III-IV, and argue that these events occur when occlusion rays from two distinct pursuers meet precisely on the environment boundary.

1) III-III merges: For a Type III vertex to merge with another Type III vertex, these two vertices must occupy the same location along an edge of ∂F. Let p and q denote the pursuers that own these two vertices, and let u and v denote the respective occlusion vertices that generate the two Type III vertices. Finally, let w and z denote the endpoints of the environment edge on which the two Type III vertices lie. Figure 8 illustrates this situation.

These two vertices merge when the lines $\overleftrightarrow{pu}$, $\overleftrightarrow{qv}$, and $\overleftrightarrow{wz}$ all share an intersection point. This triple intersection occurs when

$$\begin{vmatrix} y_{\rm u} - y_{\rm p} & x_{\rm p} - x_{\rm u} & x_{\rm u}y_{\rm p} - x_{\rm p}y_{\rm u} \\ y_{\rm v} - y_{\rm q} & x_{\rm q} - x_{\rm v} & x_{\rm v}y_{\rm q} - x_{\rm q}y_{\rm v} \\ y_{\rm z} - y_{\rm w} & x_{\rm w} - x_{\rm z} & x_{\rm z}y_{\rm w} - x_{\rm w}y_{\rm z} \end{vmatrix} = 0. \quad (2)$$

The equation expands to a polynomial of degree 2 in four variables—namely $x_{\rm p}$, $y_{\rm p}$, $x_{\rm q}$, and $y_{\rm q}$—and 8 constants.

2) III-IV merges: For a Type III vertex to merge with a Type IV vertex, again we need two occlusion rays to meet on ∂F. This situation is the same as the III-III case above, except that we are approaching from the opposite side; see Figure 9. As with the III-III case, this requires three lines (two occlusion rays and one environment edge) to meet a single point. As a result, Equation 2 describes the III-IV critical boundary as well.

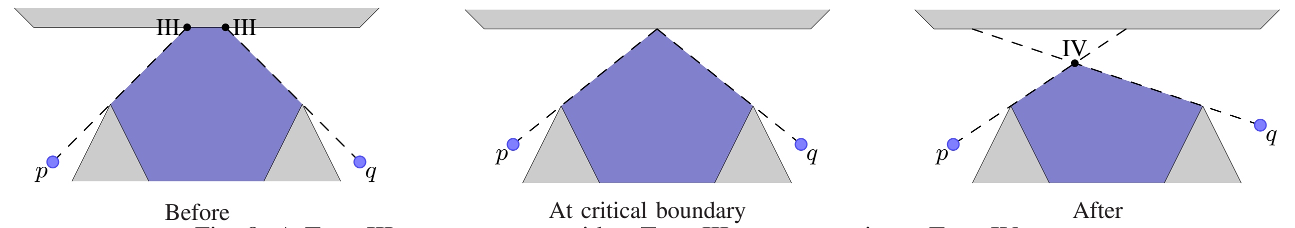


Fig. 8: A Type III vertex merges with a Type III vertex creating a Type IV vertex.

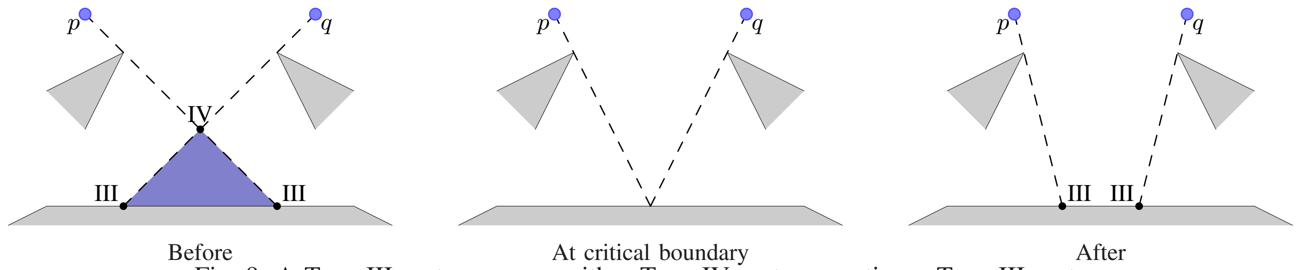


Fig. 9: A Type III vertex merges with a Type IV vertex, creating a Type III vertex.

3) Number of polynomials: These types of critical boundaries are defined by a pair of mutually visible environment vertices, along with an additional environment boundary edge. Therefore, for a fixed pair of pursuers, it can be instantiated at most $\binom{m}{3}$ different ways. It also depends on the positions of two different pursuers, of which there are $\binom{n}{2}$ unique combinations. Therefore, in total—across both III-III and III-IV—this type of critical boundary yields a maximum of $\binom{n}{2}\binom{m}{3}$ polynomials.

C. Merges resulting from three occlusion rays meeting a single point

The final plausible merge type we consider is IV-IV. For two Type IV vertices to meet, we must have at least three occlusion rays that share a single intersection point. Figure 10 shows this scenario.

Since these two vertices are adjacent, the shadow edge connecting them must be part of an occlusion ray. Let p denote the pursuer that owns the occlusion ray. Notice that the two Type IV vertices must arise from intersections with occlusion rays owned by two more pursuers, which we denote q and r. We know that those two additional pursuers are distinct—that is, $q \neq r$—because if the vertices do merge, the occlusion rays will intersect at a location other than the pursuer's location itself, which cannot occur unless the pursuer locations are distinct. The occlusion vertices for p,q, and r are denoted by u, v, and w respectively.

Thus, a IV-IV merge can occur when three distinct pursuers have occlusion rays that meet at a single point. This is, in principle, similar to the situation from Section VI-B, except that the pursuers movements can move all three relevant lines:

$$\begin{vmatrix} y_\mathrm{u} - y_\mathrm{p} & x_\mathrm{p} - x_\mathrm{u} & x_\mathrm{u} y_\mathrm{p} - x_\mathrm{p} y_\mathrm{u} \\ y_\mathrm{v} - y_\mathrm{q} & x_\mathrm{q} - x_\mathrm{v} & x_\mathrm{v} y_\mathrm{q} - x_\mathrm{q} y_\mathrm{v} \\ y_\mathrm{w} - y_\mathrm{r} & x_\mathrm{r} - x_\mathrm{w} & x_\mathrm{w} y_\mathrm{r} - x_\mathrm{r} y_\mathrm{w} \end{vmatrix} = 0.$$

In this equation, the x and y coordinates for each of the three relevant pursuers form 6 total variables, and the coordinates of their three occlusion vertices form 6 constants. The expanded polynomial has degree 3.

This scenario requires three unique environment vertices to induce occlusion rays from the pursuers, there are at most $\binom{m}{3}$ places where this can occur. This type of merge also requires three pursuers and there are $\binom{n}{3}$ unique combinations of pursuers. In total this critical boundary yields a maximum of $\binom{n}{3}\binom{m}{3}$ polynomials.

D. Merges that never occur

Finally, we argue that the remaining four merge types can never occur.

- Merges that involve only environment vertices—that is, merges of types I-I, I-II, and II-II—cannot occur because environment vertices do not move, and therefore never merge with one another.
- Merges of type I-IV cannot occur because Type I and Type IV vertices are never adjacent. Notice that, in a shadow polygon, a Type I vertex is incident to two edges along ∂F, whereas a Type IV vertex is incident to two edges in the interior of F. Therefore, there always exists at least one other vertex between any Type I and Type IV pair.

Because these merges cannot occur, they do not generate any critical boundary polynomials.

VII. Algorithm

Armed with this complete description of the critical boundaries in F^n, we can finally describe our algorithm for multiple-pursuer visibility-based pursuit-evasion in detail. The basic process is to use the critical boundaries to form a partition of F^n into conservative regions, to compute an adjacency graph of the full-dimensional cells in that partition, and then to search for a sequence of adjacent conservative regions that causes all of the shadows to be cleared.

A. Partitioning F^n via Cylindrical Algebraic Decomposition

The first step of our algorithm is to compute each of the critical boundary polynomials described in Section VI. This results in a collection $\mathcal{P}$ of $\mathrm{O}\left(n^3 m^3\right)$ polynomials in the $2n$ variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. Each of these

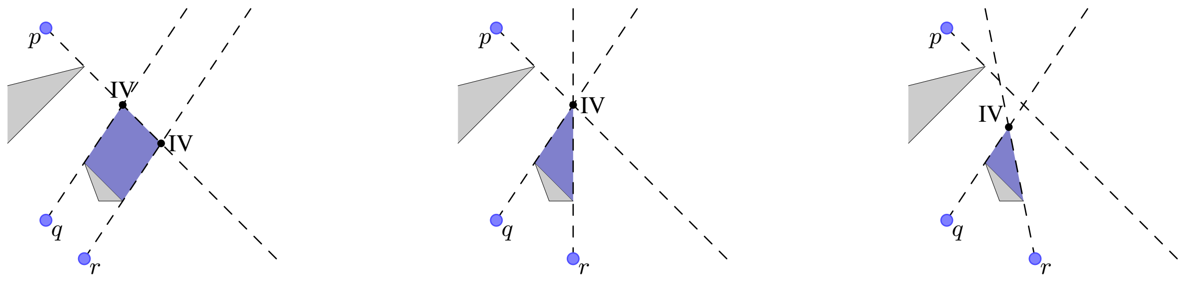


Fig. 10: A Type IV vertex merging with a Type IV vertex requires multiple robots and creates a single Type IV vertex.

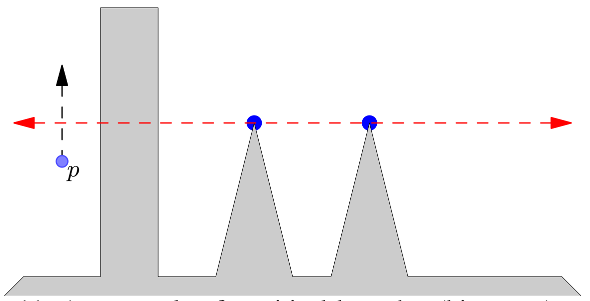


Fig. 11: An example of a critical boundary(bitangent) polynomial passing through obstacles. Because the pursuer motion shown crosses this boundary, it moves to a new CAD cell, even though no shadow event occurs.

polynomials can be constructed in constant time, so this step takes time $O\left(n^3m^3\right)$.

We then use these polynomials as input to the standard cylindrical algebraic decomposition (CAD) algorithm [3], which generates a partition of $\mathbb{R}^{2n}$ into cells with dimensions ranging from 0 to $2n$. The CAD algorithm guarantees that, within each cell of the decomposition, the sign of each polynomial in $\mathcal{P}$ remains constant. In particular, because $\mathcal{P}$ includes every critical boundary curve, this implies that every cell of dimension $2n$ is either a conservative region of the joint free space, or an obstacle portion of the joint configuration space.

Moreover, each cell of dimension $2n-1$ separates a pair of adjacent cells of dimension $2n$. Each $(2n-1)$-cell *may* correspond to a shadow event, but may also exist because of the CAD algorithm's need to form cells that are cylindrical, or may occur due to extensions of the critical boundaries—which, in the CAD algorithm, are treated as polynomials that do not stop at the environment boundary—beyond the portion of the free space in which they are relevant. See Figure 11.

B. Computing the adjacency graph of the conservative regions

Next, our algorithm forms an *adjacency graph* describing how the pursuers can move through those conservative regions.

- Each vertex of the adjacency graph corresponds to a $2n$-dimensional cell of the CAD within the joint free space.
- Edges of the adjacency graph correspond to $2n-1$-dimensional CAD cells, and connect vertices corresponding to conservative regions that share a portion of their boundaries.

There are two different approaches to the construction and search of the adjacency graph. The first [1], [12] has a multiply-exponential dependence on $2n$, whereas the second [21] takes double exponential time in $2n$. The exact construction and search of the adjacency graph is beyond the scope of this paper, and the authors refer the reader to the original text.

In addition, we label each edge of the adjacency graph with the shadow events, if any, that occur when the pursuers move between the corresponding conservative regions. By examining the shadows before and after we can retroactively assign labels to $2n-1$ cells that represent critical boundaries.

C. Path generation

Finally, we can use the adjacency graph to search for a solution strategy for the pursuers. The intuition is to search through the *pursuit-evasion graph (PEG)* induced by the adjacency graph.

1) Specifically, given a vertex v of the adjacency graph, let $k(v)$ denote the number of shadows that exist when the pursuers are within the conservative region corresponding to v. The PEG contains $2^{k(v)}$ vertices for each adjacency graph vertex v. Each such vertex is labeled with a unique binary string of length $k(v)$, representing one possible combination of clear and contaminated shadow labels. The total number of PEG vertices is $\sum_v 2^{k(v)}$.
2) A pair of PEG vertices (u, v) is connected by a directed edge $u \to v$ if
   a) the adjacency graph vertices underlying u and v are connected in that graph, and
   b) the changes in shadow labels between u and v are correct, according to the rules introduced in Section IV.

The intuition is that each vertex of the PEG fully describes one discrete information state that the pursuers might reach—including both their positions and the clear/contaminated status of each shadow—and that the edges represent "actions" that the pursuers can take to change those shadow labels.

Therefore, the final step of the algorithm is a forward search through the PEG. The search starts from the pursuers' initial position with all of the shadows labeled as contaminated, and terminates at a PEG vertex with all of the shadows are labeled clear.

The forward search is done using a Breadth-first search(BFS) algorithm. The search takes time $O(V+E)$

where V is the number of vertices in the PEG and E is the number of edges. Since the PEG is induced by the adjacency graph, any sequence of visited PEG nodes can be mapped back to the original CAD, and the process of generating a continuous path is similar to extracting a path from the original CAD as done in the standard Schwartz and Sharir algorithm [21]. If the search fails to find a path, we know that a solution does not exist because BFS performs an exhaustive search. Since by definition a PEG vertex describes one discrete information state that the pursuers might reach, the union of all PEG vertices completely describes all possible information states for the pursuers. By conducting an exhaustive search of PEG without finding a solution we conclude that there is no possible sequence of actions that the pursuers can take through the joint configuration space that guarantees the capture of the evader.

D. Algorithm analysis

We begin the analysis of our algorithm by examining the individual steps of the algorithm. The dimension of the joint configuration space is $2n$. The number of polynomials in $\mathcal{P}$—which is used as input into the CAD algorithm—is the sum of the critical boundaries and is bounded by $\mathrm{O}\left(n^3m^3\right)$. The maximum degree among the polynomials in $\mathcal{P}$ is 3 (which occurs for the IV-IV merge event.)

The total running time [13] for the construction and adjacency test on our CAD is bounded by $(3 \cdot n^3m^3)^{\mathrm{O}(1)^n}$ where $\mathrm{O}(\cdot)$ means that there exists $c \in [0, \infty]$ such that the running time is bounded by $(3 \cdot n^3m^3)^{c^n}$ [13]. The number of cells [3], [21] produced by our CAD is bound by $\mathrm{O}\left(6^{6n+1} \cdot (n^3m^3)^{4n}\right)$.

VIII. Conclusion

In this paper the authors' presented an algorithm for computing a pursuer solution strategy for a group of pursuers searching a polygonal environment for an evader. The algorithm creates a Cylindrical Algebraic Decomposition(CAD) of the pursuers joint configuration space by using polynomials that capture where critical changes can occur to the region of the environment hidden from the pursuers. Then after computing the adjacency graph for the CAD we construct a data structure called a Pursuit Evasion Graph(PEG), induced by the adjacency graph. The PEG is then exhaustively searched and returns either a path through the pursuers joint configuration space that is a pursuer solution strategy, or a statement that no such strategy exists.

One avenue of future work is providing a hardness result for the visibility-based pursuit-evasion problem. Another open problem is the construction of an incremental solution to the visibility based pursuit-evasion problem. The idea is to explore the PEG incrementally, rather than constructing the full graph explicitly.

Acknowledgement

We acknowledge support for this work from NSF (IIS-0953503).

References

[1] D.S. Arnon. A cellular decomposition algorithm for semi-algebraic sets. In *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 301–315. Springer, 1979.

[2] W. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.

[3] G. E. Collins. Hauptvortag: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In H. Barkhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer, 1975.

[4] J. W. Durham, A. Franchi, and F. Bullo. Distributed pursuit-evasion without mapping or global localization via local frontiers. *Autonomous Robots*, 32(1):81–95, 2012.

[5] B. P. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research*, 25(4):299–315, 2006.

[6] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *International Journal on Computational Geometry and Applications*, 9(5):471–494, 1999.

[7] Y. C. Ho, A. Bryson, and S. Baron. Differential games and optimal pursuit-evasion strategies. *IEEE Transactions on Automatic Control*, 10(4):385–389, October 1965.

[8] R. Isaacs. *Differential Games*. Wiley, New York, 1965.

[9] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion in a polygonal environment. *IEEE Transactions on Robotics*, 5(21):864–875, 2005.

[10] A. Kleiner and A. Kolling. Guaranteed search with large teams of unmanned aerial vehicles. In *Proc. IEEE International Conference on Robotics and Automation*, 2013.

[11] A. Kolling and C. Stefano. Multi-robot pursuit-evasion without maps. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3045–3051, 2010.

[12] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic, 1990.

[13] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[14] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–201, April 2001.

[15] S. M. LaValle, B. Simov, and G. Slutzki. An algorithm for searching a polygonal region with a flashlight. *International Journal on Computational Geometry and Applications*, 12(1-2):87–113, 2002.

[16] N. Noori and V. Isler. Lion and man with visibility in monotone polygons. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, volume 86 of *Springer Tracts in Advanced Robotics*, pages 263–278. Springer, 2013.

[17] K. J. Obermeyer, A. Ganguli, and F. Bullo. Multi-agent deployment for visibility coverage in polygonal environments with holes. *International Journal of Robust and Nonlinear Control*, 21(12):1467–1492, 2011.

[18] T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Application of Graphs*, pages 426–441. Springer-Verlag, Berlin, 1976.

[19] U. Ruiz and R. Murrieta-Cid. Time-optimal motion strategies for capturing an omnidirectional evader using a differential drive robot. *IEEE Transactions on Robotics*, 21(3), June 2013.

[20] S. M. LaValle S. Sachs and S. Rajko. Visibility-based pursuit-evasion in an unknown planar environment. *International Journal of Robotics Research*, 23(1):3–26, 2004.

[21] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: II. General techniques for computing topological properties of algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983.

[22] N. M. Stiffler and J. M. O'Kane. Shortest paths for visibility-based pursuit-evasion. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3997–4002. IEEE, 2012.

[23] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, October 1992.

[24] B. Tovar and S. M. LaValle. Visibility-based pursuit-evasion with bounded speed. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2006.

[25] J. Yu and S. M. LaValle. Shadow information spaces: Combinatorial filters for tracking targets. *IEEE Transactions on Robotics*, 28(2):440–456, 2012.