Submitted to the International Journal of Computational Geometry and Applications

A Visibility-Based Pursuit-Evasion Problem

Leonidas J. Guibas Jean-Claude Latombe Steven M. LaValle David Lin Rajeev Motwani

Computer Science Department
Stanford University
Stanford, CA 94305
{guibas,latombe,lavalle,dlin,rajeev}@cs.stanford.edu

Abstract

This paper addresses the problem of planning the motion of one or more pursuers in a polygonal environment to eventually "see" an evader that is unpredictable, has unknown initial position, and is capable of moving arbitrarily fast. A visibility region is associated with each pursuer, and the goal is to guarantee that the evader will ultimately lie in at least one visibility region. The study of this problem is motivated in part by robotics applications, such as surveillance with a mobile robot equipped with a camera that must find a moving target in a cluttered workspace.

A few bounds are introduced, and a complete algorithm is presented for computing a successful motion strategy. For a simply-connected free space, a logarithmic bound is established on the minimum of pursuers needed. Loose bounds for multiply-connected free spaces are also given. A set of problems that are solvable by a single pursuer and require a linear number of recontaminations is shown. The complete algorithm searches a finite cell complex that is constructed on the basis of critical information changes. This concept can be applied in principle to multiple-pursuer problems, and the case of a single pursuer has been implemented. Several solution strategies are shown, most of which were computed in a few seconds on a standard workstation.

1 Introduction

The general problem addressed in this paper is an extension or combination of problems that have been considered in several contexts. Interesting results have been obtained for pursuit-evasion in a graph, in which the pursuers and evader can move from vertex to vertex until eventually a pursuer and evader lie in the same vertex [18, 21]. The *search number* of a graph refers to the minimum number of pursers needed to solve a pursuit-evasion problem, and has been closely related to other graph properties such as cutwidth [17, 19]. Pursuit-evasion scenarios in continuous spaces have arisen in a variety of applications such as air traffic control [2], military strategy [11], and trajectory tracking [10]. This has resulted in the formal study of general decision problems in which two decision makers have diametrically opposing interests. Classical pursuit-evasion games express differential motion models for two opponents, and conditions of capture or optimal strategies are sought [11]. For example, in the classical Homicidal Chauffeur game, conditions of inevitable collision can be expressed in terms of the nonholonomic turning-radius constraints of the pursuer and evader. Although interesting decision problems arise through the differential motion models, geometric free-space constraints are usually not considered in classical pursuit-evasion games. Once these constraints are introduced, the problem inherits the additional complications that arise in geometric motion planning.

A region of capture is often associated with a pursuit-evasion problem, and the "capture" for our problem is defined as having the evader lie within a line-of-sight view from a pursuer. A moving visibility polygon in a polygonal environment adds geometric information that must be utilized, and also leads to connections with the static art gallery problems [20, 24]. In the limiting case, art gallery results serve as a loose upper bound on the number of pursuers by allowing a covering of the free space by static guards, guaranteeing that any evader will be immediately visible. Far fewer guards are needed when they are allowed to move and search for an evader; however, the required motion strategies can become quite complex. A closely related art gallery variant is the watchman tour problem [6]. In this case a minimum-length closed path is computed such that any point in the polygon is visible from some point along the path. In our case, however, the pursuers have the additional burden of ensuring that an evader cannot "sneak" to a portion of the environment that has already been explored. Variations of this problem have also been considered in [25]. It was stated in [24] that it is difficult to even determine if a polygon is searchable by a single pursuer.

Several applications can be envisioned for problems and motion strategies of this type. For example, suppose a building security system involves a few mobile robots with cameras or range sensors that can detect an intruder. A patrolling route can be automatically computed that guarantees that any mobile intruder will eventually be found. To optimize expenses, it would also be important to know the minimum number of robots that would be needed.

Applications are not necessarily limited to adversarial targets. For example, the task might be to automatically locate another mobile robot, items in a warehouse or factory that might get moved during the search process, or possibly even people in a search/rescue effort. Such strategies could be used by automated systems or by human searchers.

Section 2 presents a precise mathematical formulation of the problem. Section 3 presents several bounds on the number of required pursuers and related problems. Section 4 presents general concepts for reducing the problem to a finite graph search and a complete algorithm for computing a solution strategy for a given free space. Section 5 shows several example solution strategies that were computed using our implemented algorithm, and discusses some of the practical implementation issues. Conclusions are presented in Section 6.

2 Problem Definition

The pursuers and evader are modeled as points that translate in a polygonal free space, F. Let $e(t) \in F$ denote the position of the *evader* at time $t \geq 0$. It is assumed that $e : [0, \infty) \rightarrow F$ is a continuous function, and the evader is capable of moving arbitrarily fast. The initial position $e(0)$ and the path e are assumed unknown to pursuers. Any region in F that might contain the evader will be referred to as *contaminated*, otherwise it will be referred to as *cleared*. If a region is contaminated, becomes cleared, and then becomes contaminated again, it will be referred to as *recontaminated*.

Let $\gamma^i(t)$ denote the position of the i^{th} *pursuer* at time $t \geq 0$. Let γ^i represent a continuous path of the i^{th} pursuer of the form $\gamma^i : [0, \infty) \rightarrow F$. Let γ denote a *(motion) strategy*, which refers to the specification of a continuous path for every pursuer: $\gamma = \{\gamma^1, \ldots, \gamma^N\}$.

For any point, $q \in F$, let $V(q)$ denote the set of all points in F that are visible from q (i.e., the linear segment joining q and any point in $V(q)$ lies in F). A strategy, γ, is a *solution strategy* if for every continuous function $e : [0, \infty) \rightarrow F$ there exists a time $t \in [0, \infty)$ and an $i \in \{1, \ldots, N\}$ such that $e(t) \in V(\gamma^i(t))$. This implies that the evader will eventually be seen by one or more pursuers, regardless of its path. Let $H(F)$ represent the minimum number of pursuers for which there exists a solution strategy for F.

Section 3 presents some bounds on $H(F)$ for classes of free spaces, and also shows that some polygons for which $H(F) = 1$ only admit solutions that require a linear number of recontaminations. Section 4 addresses the problem of computing a solution strategy, γ, for a given F.

3 Worst-Case Bounds

Several new bounds are presented in this section. For a simply-connected free space, F, with n edges, it is shown that $H(F) = \Theta(\lg n)$. For a free space, F, with h holes, it is shown that $H(F) = \Omega(\sqrt{h} + \lg n)$ and $H(F) = O(h + \lg n)$. For the class of problems in which $H(F) = 1$, it is shown that the same region can require recontamination as many as $\Omega(n)$ times. This result is surprising because pursuit-evasion in a graph is known not to require any recontaminations [15].

Consider the problem of determining the minimum number of pursuers, $H(F)$, required to find an evader in a given free space F. This number will generally depend on both the topological and geometric complexity of F. In [25] a class of simple polygons is identified for which a single pursuer suffices (referred to as "hedgehogs"). For any F that has at least one hole, it is clear that at least two pursuers will be necessary; if a single pursuer is used, the evader could always move so that the hole is between the evader and pursuer. In some cases subtle changes in the geometry significantly affect $H(F)$. Consider for example, the problems in Figure 1. Although the problems are similar, only the problem in the lower right requires two pursuers.

Consider $H(F)$ for the class of simply-connected free spaces. Let n represent the number of edges in the free space, which is represented by a simple polygon in this case. A logarithmic worst-case bound can be established:

Theorem 1 *For any simply-connected free space F at worst $H(F) = O(\lg n)$.*

Proof: There always exists a pair of vertices in a simple polygon that can be connected so that the polygon is partitioned into two regions, each with at least one third of the edges of the original polygon. Select such a pair of vertices, and partition the polygon by adding an edge, $E_{0,0}$, between the chosen vertices. Each of the two new simple polygons can be

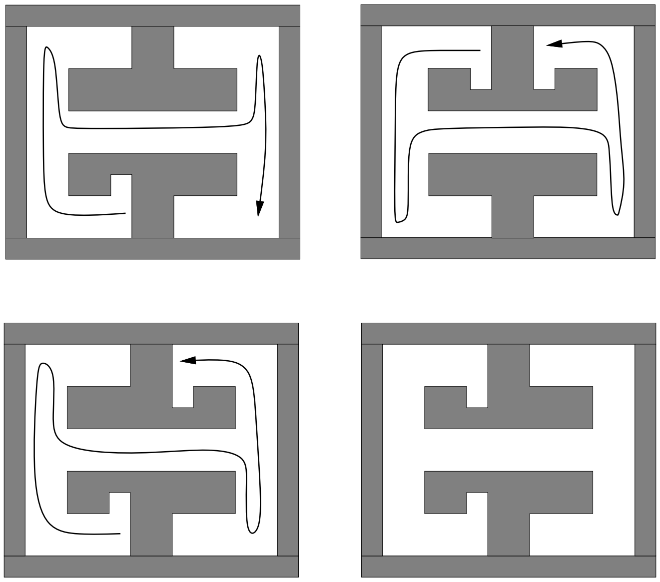


Figure 1: Four examples are shown that have similar geometry. The example in the lower right requires two pursuers, while the other three examples require only one.

partitioned in the same manner by selecting appropriate vertices, resulting in edges $E_{1,0}$ and $E_{1,1}$. This procedure can be recursively applied until the original polygon is triangulated with edges of the form $E_{i,j}$, in which i represents the level of recursion. The index j represents the position of the edge from left to right in level i of a binary tree in which each node represents a partitioning edge (see Figure 2). Let N represent the maximum number of levels in the tree, and note that $N = O(\lg n)$ due to the recursive partitioning. For each $i \in \{1, \ldots, N\}$, place a pursuer anywhere on edge $E_{i,0}$. If a pursuer is placed on a partition edge, then the evader is forced to remain on the same "side" of the partition. A motion strategy can now be specified for the N pursuers that guarantees that the evader will be seen. For the purpose of discussion only, assume that the binary tree is height balanced. The N^{th} pursuer can be moved through $E_{N-1,0}$, and placed on $E_{N,1}$. The portion of the polygon to the left of $E_{N-1,0}$ has now been eliminated from consideration. In the next step, the $(N-1)^{th}$ pursuer can be moved through $E_{N-2,0}$, and placed on $E_{N-1,1}$, while the N^{th} pursuer is moved through $E_{N-1,0}$ and $E_{N-2,0}$, and is placed on $E_{N,2}$. During this step, another portion of the free space is eliminated. This type of motion can be iterated, eventually sweeping all of the pursuers across the tree from left to right. In the final step, the i^{th} pursuer terminates on $E_{i,2^i}$, and the entire free space has been successfully searched using $N = O(\lg n)$ pursuers. $\square$

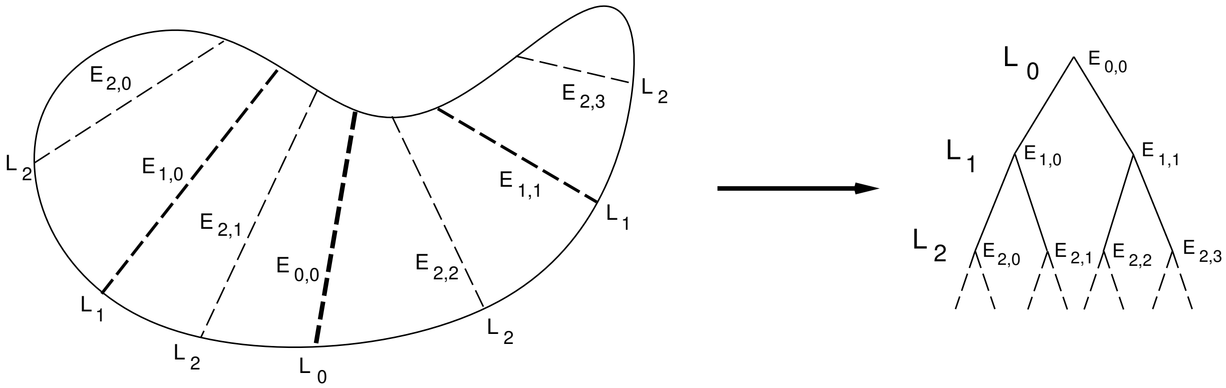


Figure 2: Recursive partitioning to determine a solution strategy that requires $O(\lg n)$ pursuers.

The remaining question for simply-connected free spaces is whether there actually exist problems that require a logarithmic number of pursuers. Some results from graph searching will first be described and utilized to construct difficult worst-case problem instances. Let *Parsons' problem* refer to the graph-searching problem presented in [18, 21]. The task is to specify the number of pursuers required to find an evader that can execute continuous motions along the edges of a graph. Instead of using visibility, capture is achieved when one of the pursuers "touches" the evader. Let G represent a graph, and $S(N)$ represent the number of needed pursuers, referred to as the search number of G.

The following lemma implies that a geometric realization of any planar graph instance can be constructed:

Lemma 1 *For every planar graph, G, there exists a polygonal free space F such that Parsons' problem on G is equivalent to the visibility-based pursuit evasion problem on F.*

Proof: Since G is planar, a geometric representation exists in the plane in which points in $\Re^2$ correspond to vertices in G, and linear segments between the points correspond to edges in G. Consider the corridor structure shown in Figure 3. Every linear segment in the geometric representation of G can be replaced by a corridor of sufficient length as shown in Figure 3. Furthermore, there exists an $\epsilon > 0$ such that no pair of corridors intersect, except near the points that correspond to vertices of G. Portions of the corridor edges can be removed at corridor junctions to prevent overlap. Let F refer to the resulting polygon,

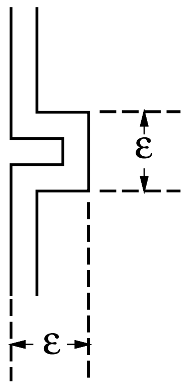


Figure 3: A corridor of this shape disconnects second-order visibility between the two entrances, and can be used to construct geometric equivalents of Parsons' problem for planar graphs.

which represents a network of bent corridors.

The next task is to show that searching F is equivalent to searching G. Recall that any successful Parsons' search strategy can be specified by the traversal of a sequence of edges for each pursuer. If the sequence of bent corridors is explored that correspond to the edges of a solution strategy for G, then F will be successfully searched. This is true since using visibility to "see" an evader in a corridor is at least as powerful as attempting to "touch" an evader in a continuous graph edge.

Next consider whether any solution strategy for F can be used to equivalently search G. The corridor piece from Figure 3 is intentionally bent in four places, which causes all advantages of visibility to be lost. Suppose this corridor is connected at both ends to other corridors, and that pursuers are placed at each end (at positions x^1 and x^2). For this corridor, $V(V(x^1))$ and $V(V(x^2))$ are disjoint, in which $V(V(x^i))$ represents the set of all points from which $V(x^i)$ is visible. Although both pursuers can see into the corridor, one of the pursuers must travel into the middle of the corridor at some point to search for the evader. It must travel far enough so that the entrance to the corridor is no longer visible (leaving the entrance "unguarded"). The central portion of a corridor will correspond directly to the edges in G, since either can only be explored by leaving the junctions or vertices unguarded.

Consider any given motion strategy γ that is a solution for F. Suppose there are N pursuers. For each $i \in \{1, \ldots, N\}$, it will be shown that γ_i can be used to determine a sequence of edges for the i^{th} pursuer in G. Without loss of generality, it can be assumed that γ_i only traverses the centers of the corridors (i.e. equal distance is maintained between the corridor walls). Thus, γ_i can be characterized by indicating how far into a corridor the i^{th} pursuer travels, at which point it performs a reversal, which corridor it selects at a junction, etc. Suppose γ_i travels from junction to junction, with reversals only being made at junctions. In this case, every corridor in F that is cleared will cause the corresponding edge in G to be cleared. If γ is a solution strategy for this case, then a corresponding solution strategy for G is implied. Suppose that γ_i actually causes reversals to occur in a corridor (i.e., not at a junction). If the pursuer changes direction but still traverses the full length of the corridor (it must change direction at least twice), then the corresponding edge in G will still be cleared. If the pursuer returns to the originating junction, then there are two possible cases. If the pursuer travels far enough to clear the central portion of the corridor, then the originating junction is left unguarded. The corresponding edge in G can be cleared by moving the pursuer from the originating vertex (which corresponds to the originating junction in F), across the edge in G to the edge's other vertex, and back to the originating vertex. If the pursuer does does not travel far enough into the corridor to clear the central portion, then this portion of γ^i does not make progress, and can be discarded (i.e., no corresponding strategy portion needs to be considered for G). Thus any solution strategy, γ, for F can be used to determine a corresponding solution strategy for G. □

A theorem from [21] will be useful for proving Theorem 2, which provides a logarithmic lower bound on the number of pursuers needed to successfully search a simply-connected free space:

Lemma 2 (Parsons) *Let G be a tree. Then $S(G) = N + 1$ if and only if there exists a vertex in G whose removal separates G into three components, G_1, G_2, and G_3, such that $S(G_i) \geq N$ for $i \in \{1, 2, 3\}$.*

Theorem 2 *There exist simply-connected free spaces F with n edges such that $H(F) = \Omega(\lg n)$.*

Proof: Using Lemma 2, a tree, G, can be constructed recursively that has a constant branching factor of three, height $N - 1$, and requires N pursuers (an example is given in

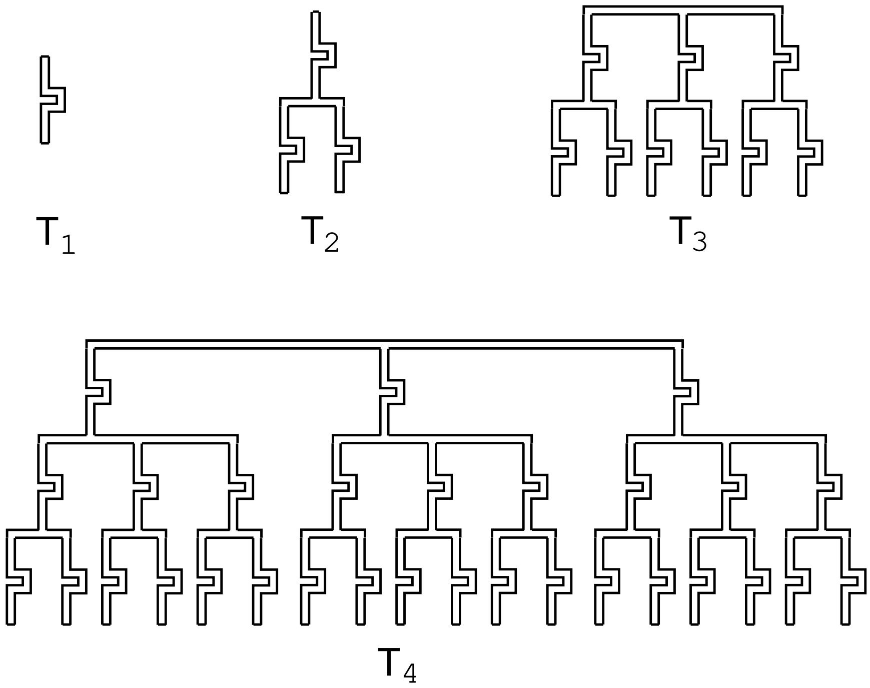


Figure 4: Systematic construction of simply-connected free spaces that require $\Omega(\lg n)$ pursuers.

[21]). By Lemma 1, an equivalent geometric instance can be constructed for each N. Figure 4 depicts these geometric instances, for which $H(F) = \Omega(\lg n)$ □

Theorem 1 and Theorem 2 together imply a tight logarithmic bound, $H(F) = \Theta(\lg n)$.

Next consider the class of problems for which F has h holes. A square-root lower bound and a linear upper bound are obtained, in terms of h. This obviously leaves the problem open of determining a tight bound for this class.

Theorem 3 *For any free space F with h holes at worst $H(F) = O(h + \lg n)$.*

Proof: A linear number of line segments can be used to connect between holes and the exterior edges of F so that any continuous path that is not homotopic to a stationary path must cross one of the line segments. Each segment effectively removes one component from the allowable path classes in F. A stationary pursuer can be placed on each line segment, preventing an evader from moving in a path that encloses a hole. The evader is trapped in a region that can be delineated with a simple polygon. Each such region can be independently searched using only a logarithmic number of pursuers by Theorem 2. Thus the total number

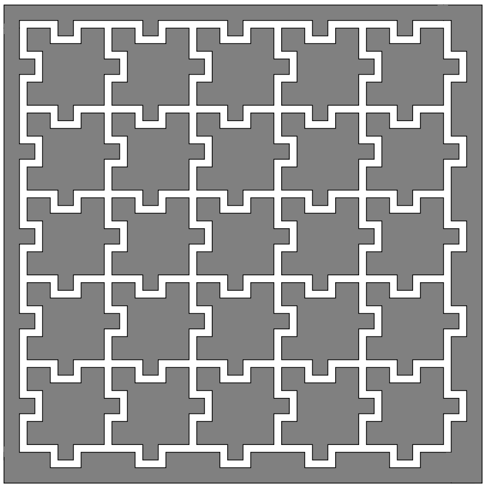


Figure 5: An instance from a sequence of problems that requires a number of pursuers that is at least proportional to the square root of the number of holes.

of pursuers is no worse than $O(h + \lg n)$. $\Box$

Theorem 4 *There exist free spaces F with h holes such that $H(F) = \Omega(\sqrt{h} + \lg n)$.*

Proof: For any positive integer k, a planar graph of cutwidth k can be constructed using $O(k^2)$ vertices and edges. Recall that the cutwidth, $CW(G)$, is the minimum cutwidth taken over all possible linear layouts of G. A linear layout of G is a one-to-one function mapping the vertices of G to integers, and the cutwidth for a particular layout is the maximum over all i of the number of edges connecting vertices assigned to integers less than i to vertices assigned to integers as large as i. Define a sequence of planar graphs, $G_1, G_2, \ldots$. Let the vertices of G_k correspond to the set of all points with integer coordinates, (i, j), such that $0 \leq i, j \leq k$. Let the edges of G_k connect any two vertices for which one coordinate differs by one unit (i.e., a standard four-neighborhood). The cutwidth of G_k is k.

It is established in [17] that for all graphs G, the search number $S(G)$ is related to the cutwidth as $S(G) \leq CW(G) \leq \lfloor deg(G)/2 \rfloor \cdot S(G)$, in which $deg(G)$ is the maximum vertex degree of G. Because $deg(G_k) = 4$, $S(G) \leq k \leq 2S(G)$. Using Lemma 1, geometric instances of G_k such as the one shown in Figure 5 can be constructed. Both G_k and each geometric instance require $\Omega(k)$ pursuers. There is a quadratic number of holes in each geometric instance; hence, $H(F) = \Omega(\sqrt{h})$. This corridor structure can be combined with the structure from Theorem 2 to yield an example that requires $\Omega(\sqrt{h} + \lg n)$ pursuers. $\Box$

The final theorem of this section pertains to the class of free spaces that can be searched by a single pursuer. It states that there exist examples that require recontaminating some portion of the free space a linear number of times. This result is surprising because for Parsons' problem it was shown in [13] that no recontamination is necessary (a shorter proof of this appears in [4]). In [25] a free space was given that requires two recontaminations, which at least established that recontamination is generally necessary for visibility-based pursuit evasion. It appeared to many that this might be the worst pathological case; however, Theorem 5 establishes that a linear number of recontaminations can be needed, which makes the visibility-based pursuit-evasion problem significantly worse than Parsons' graph problem. The theorem represents a lower bound, and it still remains open to determine whether the number of recontaminations can be bounded from above by a polynomial, which would imply that the problem of deciding whether $H(F) = 1$ lies in NP.

Theorem 5 *There exists a sequence of simply-connected free spaces with $H(F) = 1$ such that $\Omega(n)$ recontaminations are required for n edges.*

Proof: It will be shown that the example in Figure 6 requires $k - 2$ recontaminations by visiting the point $a \in F$ a total of $k - 1$ times to repeatedly clear the "peak." Without loss of generality, consider the set of strategies that can be specified by identifying the sequence of points, $a, b_1, \ldots, b_k, c_1, \ldots, c_k$, that are visited. Assume that the shortest-distance path is taken between any pair of points. Consider visiting b_i for some $1 < i < k$, followed by a visit to another "leg", say b_j (or c_j). If any legs between b_i and b_j are contaminated, then b_i will get contaminated, which undoes previous work. If all legs are initially contaminated, then they must be visited in one of two orders: $(b_1, c_1, b_2, c_2, \ldots, b_k, c_k)$ or $(b_k, c_k, b_{k-1}, c_{k-1}, \ldots, b_1, c_1)$. Because of symmetry, consider visiting the legs from left to right without loss of generality. Assume that the peak is initially contaminated. The points b_1 and c_1 can be visited to clear the leftmost set of legs; however, these will get contaminated when b_2 is visited. By traveling from c_1 to a to b_2, the leftmost set of legs remain cleared because the peak is cleared. When c_2 is visited, the leftmost three legs remain cleared; however, the peak becomes recontaminated. Thus, a will have to be visited again before clearing b_3. By induction on i for $1 < i \leq k$,

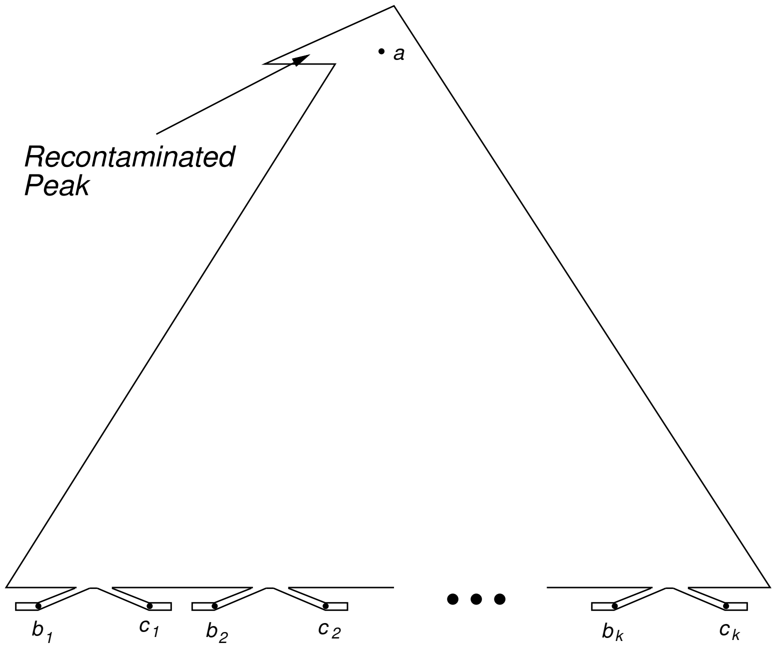


Figure 6: A linear number of recontaminations is required. Although this polygon can be searched by a single pursuer, the peak must be visited $k-1$ times.

the peak will have to be cleared by visiting a each time between visits to c_i and b_{i+1}. This implies that a will be visited $k-1$ times, resulting in $k-2$ recontaminations. □

4 Computing a Solution Strategy

While Section 3 addressed worst-case bounds over certain problem classes, this section covers concepts and algorithms for computing a solution strategy for a given free space. Section 4.1 defines a general information space (or space of knowledge states) for this problem, and provides a general method for partitioning the information space into equivalence classes, which can reduce the general problem to finite cell searching. NP-hardness is also established in Section 4.1. Section 4.2 presents a complete algorithm for the case in which $H(F) = 1$ that computes a solution strategy by decomposing F into convex cells based on edge visibility. This algorithm is quite efficient in practice, and was used to compute the examples shown in Section 5. The case in which $H(F) > 1$ is discussed in Section 4.3.

4.1 General issues

In general, one would prefer a sound, complete algorithm. A *sound* algorithm which computes a solution strategy should accept F as input and return a γ that will guarantee that the evader will eventually be seen. A *complete* algorithm must compute a solution strategy for a given number of pursuers, if such a strategy exists. Another algorithm could be used to compute the minimum number of pursuers, or the complete algorithm could be iterated while incrementally increasing the number of pursuers after each failure. It is natural to compare the notion of completeness for this problem to completeness for the basic motion planning problem (i.e., the algorithm will find a collision-free path if such a path exists [5]). One important difference, however, is that the *minimum* number of pursuers is crucial, but does not have a correspondence for the basic path planning problem. A variety of simple, heuristic algorithms can be developed that require more pursuers than necessary (for example, triangulate the workspace, and place a static pursuer in each triangle). The problem becomes most difficult when the minimum number of pursuers is requested.

The general problem is intractable if $P \neq NP$:

Theorem 6 *Computing $H(F)$ is NP-hard.*

Proof: It is shown in [19] that Parsons' problem for a planar graph with maximum vertex degree 3 is NP-complete (i.e., computing the search number, $S(G)$). By Lemma 1, equivalent geometric instances can be constructed, which implies that computing $H(F)$ is NP-hard. □

This significantly reduces hopes that an efficient algorithm can be determined for the general problem; nevertheless, a complete algorithm that depends exponentially on the number of pursuers is presented in this paper (it is efficient in practice for $H(F) = 1$). One interesting direction for future research is to consider algorithms that find solutions using a number of pursuers that is within a bound of optimal.

An information space Because the position of the evader is unknown, one does not have direct access to the state at a given time. This motivates the consideration of an information space that identifies all unique situations that can occur during the execution of a motion strategy. Let a *state space*, X, be spanned by the coordinates $x = (x^1, \ldots, x^N, x^e)$, in which

x^i for $1 \leq i \leq N$ represents the position of the i^{th} pursuer, and x^e represents the position of the evader. Since the positions of the pursuers are always known, let X^p denote the subspace of X that is spanned by the pursuer positions, $x^p = (x^1, \ldots, x^N)$.

It will be useful to analyze a strategy in terms of manipulating the set of possible positions of the evader. Let $S \subseteq F$ represent the set of all contaminated points in F. Let $\eta = (x^p, S)$ for which $x^p \in X^p$ and $S \subseteq F$ represent an *information state*. Let the *information space*, $\mathcal{I}$, represent the set of possible information states. The information space is a standard representational tool for problems that have imperfect state information, and has been useful in stochastic optimal control and dynamic game theory (e.g., [2, 12]), and in motion planning [3, 16]. The information states also correspond to the knowledge states introduced for manipulation planning under uncertainty in [7].

For a fixed strategy, γ, a path in the information space will be obtained by $\eta(t) = (\gamma^1, \ldots, \gamma^N, S(t))$ in which $S(t)$ can be determined from an initial $S(0)$ and the trajectories $\{\gamma^i(t') | t' \in [0, t]\}$ for each $i \in \{1, \ldots, N\}$. Let $\Psi(\eta, \gamma, t_0, t_1)$ represent the information state that will be obtained by starting from information state η, and applying the strategy γ from t_0 to t_1. The function Ψ can be thought of as a "black box" that produces the resulting information state when a portion of a given strategy is executed.

Identifying critical information changes We next describe a general mechanism for defining critical information changes. This is inspired in part by a standard approach used in motion planning, which is to preserve completeness by using a decomposition of the configuration space that is constructed by analyzing critical events (e.g., [1, 14, 22]). For example, in [23] a cell decomposition is determined by analyzing the contact manifolds in a composite configuration space that is generated by the positions of several disks in the plane.

The next definition describes an information invariance property, which allows the information space, $\mathcal{I}$, to be partitioned into equivalence classes. A connected set $D \subseteq X^p$ is *conservative* if $\forall \eta \in \mathcal{I}$ such that $x^p \in D$, and $\forall \gamma : [t_0, t_1] \rightarrow D$ such that γ is continuous and $\gamma(t_0) = \gamma(t_1) = x^p$, then the same information state, $\eta = \Psi(\eta, \gamma, t_0, t_1)$, is obtained.

This implies that the information state cannot be altered by moving along closed paths in D. Just as in the case of motions in a conservative field, the following holds:

Theorem 7 (Path invariance) *If D is conservative then for any two continuous trajectories, γ_1, γ_2, mapping into D such that $\gamma_1(t_0) = \gamma_2(t_0)$ and $\gamma_1(t_1) = \gamma_2(t_1)$ then $\Psi(\eta, \gamma_1, t_0, t_1) = \Psi(\eta, \gamma_2, t_0, t_1)$, for any η.*

Proof: Select any third continuous trajectory, $\gamma_3 : [t_0, t_1] \to D$, such that $\gamma_3(t_0) = \gamma_1(t_1)$ and $\gamma_3(t_1) = \gamma_1(t_0)$ (i.e., heading in the opposite direction). Form a new trajectory, γ_{132}, by concatenating the trajectories γ_1, γ_3, and γ_2. The resulting information state will be $\Psi(\eta, \gamma_1, t_0, t_1)$ because γ_3 followed by γ_2 forms a closed-loop path, and thus yields the same information state by conservativity of D. Note that γ_1 followed by γ_3 is also a closed-loop path, which implies that γ_2 must bring the information state from η to $\Psi(\eta, \gamma_1, t_0, t_1)$. Hence, $\Psi(\eta, \gamma_1, t_0, t_1) = \Psi(\eta, \gamma_2, t_0, t_1)$. $\square$

Thus, the information state from moving between $x_1^p \in D$ and $x_2^p \in D$ is invariant with respect to the chosen path. Let $\mathcal{D}$ represent a collection of conservative cells that forms a partition of X^p. The cellular partition, $\mathcal{D}$, of X can be used to define an equivalence relation, $\sim$, on the information space, $\mathcal{I}$. For any two information states $\eta_1 = (x_1^p, S_1)$ and $\eta_2 = (x_2^p, S_2)$, define $\eta_1 \sim \eta_2$ if there exists some $D \in \mathcal{D}$ such that $x_1^p, x_2^p \in D$, and there exists some continuous γ, t_0, and t_1 such that $\gamma(t_0) = x_1^p$, $\gamma(t_1) = x_2^p$, γ maps into D, and $\Psi(\eta_1, \gamma, t_0, t_1) = \eta_2$. In other words, two information states are equivalent if they project into the same cell, and one is reachable from the other by executing a continuous motion that remains in this cell. Let $\mathcal{I}/\!\sim$ represent a quotient space in which elements are the equivalence classes with respect to $\sim$. For the complete algorithm, the elements of $\mathcal{I}/\!\sim$ will be considered as vertices in a graph. The edges in the graph yield pairs of information equivalence classes that can be reached by moving from one conservative cell to another. Such a graph can be searched to yield a solution strategy, without regard to the particular choice of path within each conservative cell.

4.2 A complete algorithm for a single pursuer

Since the general problem is NP-hard, it is worth focusing on the complete algorithm for the case of a single pursuer. The basic idea is to partition the free space into convex cells that maintain completeness, and perform a search on the resulting quotient information space. This algorithm has been implemented and tested on a variety of examples, which are shown in Section 5. Issues pertaining to multiple pursuers are deferred until Section 4.3.

Maintaining information states Suppose the pursuer is at a point $q \in F$. Consider the circular sequence of edges in the resulting visibility polygon. The edges generally alternate between bordering an obstacle and bordering free space. See Figure 7. Let each edge that borders free space be referred to as a *gap edge*. Consider associating a binary label with each gap edge. If the portion of the free space that borders the gap edge is contaminated, then it is assigned a "1" label; otherwise, it is assigned a "0" label indicating that it is clear. Let $B(q)$ denote a binary sequence that corresponds to labelings that were assigned from $q \in F$. Note that the set of all contaminated points is bounded by a polygon that must contain either edges of F or gap edges from the visibility polygon of the pursuer. Thus, the specification of q and $B(q)$ uniquely characterizes the information state.

There is an additional restriction on $B(q)$ that must be enforced. If two gap edges lie on the border of the same component of $F \setminus V(q)$, then they must have the same label at all times. If this is not true, then the same component would be identified as being both recontaminated and cleared. By performing connectivity analysis, one can consequently reduce the number of bits in $B(q)$; the algorithm described in this section will function correctly in either case.

A cell decomposition that preserves completeness Consider representing the information state using q and $B(q)$, and let pursuer move in a continuous, closed-loop path that does not cause gap edges to appear or disappear at any time. Each gap edge will continuously change during the motion of the pursuer; however, the corresponding gap edge label will not change. The information state cannot change unless gap edges appear or disappear.

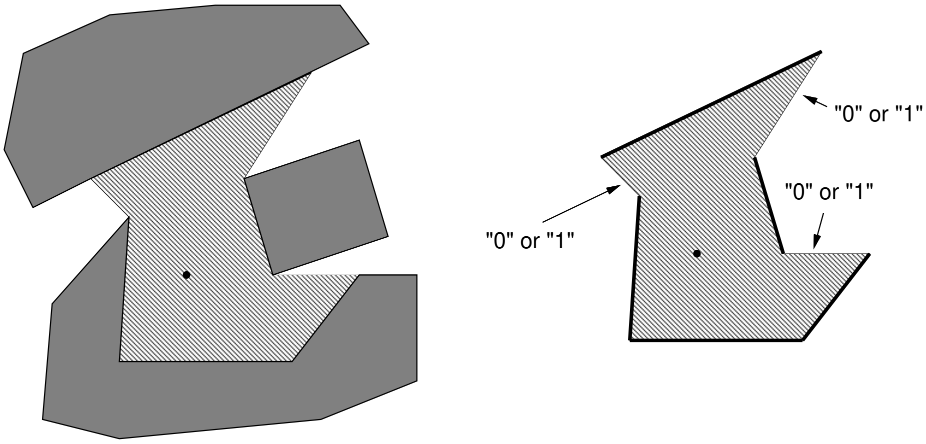


Figure 7: Edge labels can be used to encode the information state.

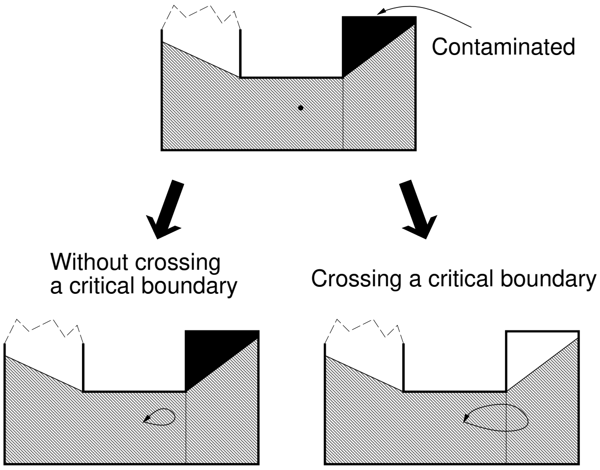


Figure 8: A critical event in the information space can only occur when edge visibility changes.

For example, consider the problem shown in Figure 8 which shows a single pursuer that is approaching the end of a corridor. If the closed-loop motion on the left is executed, the end of the corridor remains contaminated. This implies that although the information state chages during the motion, the original information state is obtained upon returning. During the closed-loop motion on the right, the gap edge disappears and reappears. In this case, the resulting information state is different. The gap label is changed from “1” to “0”.

Hence, a cell decomposition that maintains the same corresponding gap edges will only contain conservative cells. The idea is to partition the free space into convex cells by identifying critical places at which edge visibility changes. A decomposition of this type has been

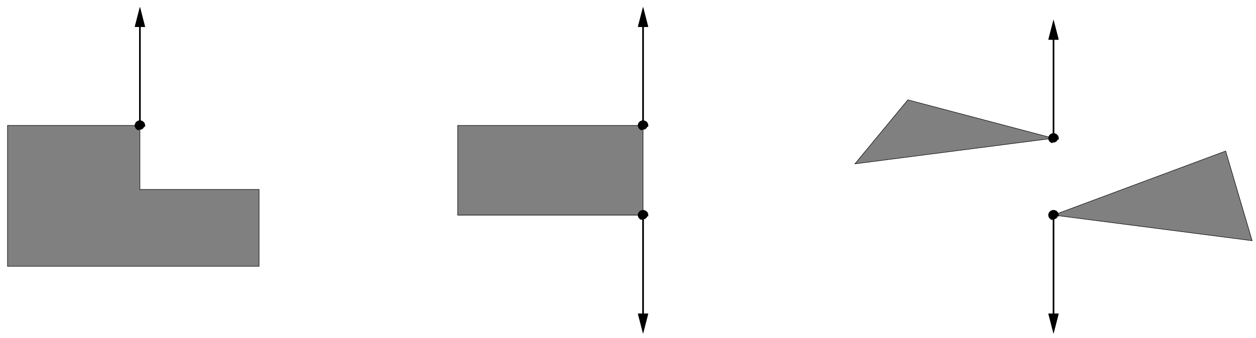


Figure 9: Ray shooting is performed for three general cases to form the edge-visibility cells.

used for robot localization in [9, 26], and generates $O(n^3)$ cells in the worst case for a simple polygon (which is always true if $H(F) = 1$). The free space can be sufficiently partitioned in our case by extending rays in the three general cases shown in Figure 9. Obstacle edges are extended in either direction, or both directions if possible. Pairs of vertices are extended outward only if both directions are free along the line drawn through the pair of points. This precludes the case in which one direction is cannot be extended; although edge visibility actually changes for this case, it does not represent a critical change in information. Figure 16.a shows a computed example of this cell decomposition.

Searching the information space The next issue is searching the information space for a solution, which corresponds to specifying a sequence of adjacent cells. The solution strategy must take the form of a path that maps into F. This can be constructed by concatenating linear path segments, in which each segment connects the centroids of a consecutive pair of cells in the sequence.

The cells and their natural adjacency relationships define a finite, planar graph, G_c, referred to as the *cell graph.* Vertices in G_c are generally visited multiple times in a solution sequence because of the changing information states. For each vertex in G_c, a point, $q \in F$, in the corresponding cell can be identified, and the labels $B(q)$ can be distinct at each visit. Initially, the pursuer will be in some position at which all gap labels are "1". The goal is to find any sequence of cells in G_c that leave the pursuer at some position at which all gap labels are "0".

A directed *information state graph*, G_I, can be derived from G_c, for which each vertex is visited at most once during the execution of a solution strategy. For each vertex in G_c, a set of vertices are included in G_I for each possible $B(q)$. For example, suppose a vertex in G_c represents some cell D, and there are 2 gap edges for $B(q)$ and any $q \in D$. Four vertices will be included in G_I that all correspond to the pursuer at cell D; however, each vertex represents a unique possibility for $B(q)$: "00", "01", "10", or "11". Let a vertex in G_I be identified by specifying the pair $(q, B(q))$.

To complete the construction of G_I, the set of edges must be defined. This requires determining the appropriate gap labels as the pursuer changes cells. Suppose the pursuer moves from $q_i \in D_i$ to $q_j \in D_j$. For the simple case shown in the lower right of Figure 8, assume that the gap edge on the left initially has a label of "0" and the gap edge on the right has a label of "1". Let the first bit denote the leftmost gap edge label. The first transition is from "01" to "0", and the second transition is from "0" to "00". The directed edges in G_I are (q_i,"01") leads to (q_j,"0"), (q_j,"0") leads to (q_i,"00").

In the case of multiple gap edges, correspondences must be determined to correctly compute the gap labels. Consider the example shown in Figure 10 which illustrates the general cases that can occur. A gap edge from $V(q_i)$ corresponds to a gap edge from $V(q_j)$ if they share a vertex, and neither touch the extension of their common cell boundary. This case is shown in the upper left of Figure 10. In this case the binary label with be preserved when traveling directly from q_1 to q_2. The case is more interesting when gap edges touch the extension of the cell boundary, as in the lower portion of Figure 10. In general, all edges that touch the extension below the cell correspond to each other, and all edges that touch the extension above the cell separately correspond to each other. Transitions of this type essentially cause gap edges to be split or merged. There are two gap edges in the lower portion of Figure 10 while the pursuer is at q_1; however, there is only one gap edge when the pursuer is at q_2. In the transition from q_1 to q_2, if the gap edges at q_1 are labeled "0" and "0", then the corresponding gap from q_2 will be labeled "0". If either gap edge at q_1 is labeled "1", then the gap edge label from q_2 will be "1" (contamination spreads easily). In general, if any n gap edges are merged, the corresponding gap edges will receive a "1" label if any of the original gap edges contain a "1" label.

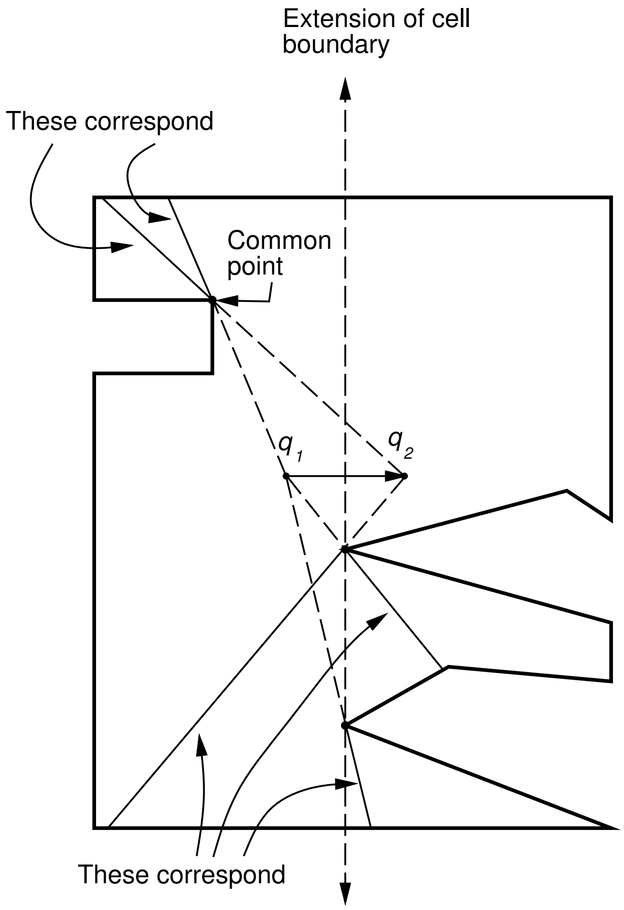


Figure 10: The correspondences between gap edges from different neighboring cells can be directly determined. The information states are updated when moving between cells by using this correspondence.

Once the gap edge correspondences have been determined, the information state graph can be searched using Dijkstra's algorithm with an edge cost that corresponds to the distance traveled in the free space by the pursuer. Unfortunately, the precise complexity of the complete algorithm cannot be determined because it is still open whether the problem even lies in P or NP. In the worst-case, examples can be constructed that yield an exponential number of information states, but it is not clear whether these information states necessarily have to be represented and searched to determine a solution (or to verify a solution).

4.3 Multiple pursuers

In general, the conservative concept that was presented in Section 4.1 can be applied to yield a cell decomposition of X^p, which is the $2N$-dimensional space that encodes the positions of the pursuers. Due to the hardness of the general problem, however, it still remains challenging to develop and implement a practical algorithm even for the case of two pursuers. A cell decomposition must be constructed for the case of N pursers such that the edges in the union of the N visibility polygons do not change (edges to not vanish or appear). Conservative cells were formed by maintaining constant edge visibility for the single-pursuer case, and one would hope that a Cartesian product of planar cells could be formed to directly define conservative cells for multiple pursuers; however, as Figure 11 indicates, the conservative cells can be considerably more complicated for the two pursuer case. Visibility edges from two different visibility polygons can intersect in such a way that it is possible to execute a closed-loop path that changes the information state while keeping both pursuers within their edge visibility cells. For the example in Figure 11, suppose the pursuer on the left moves slightly toward the right and returns, while the pursuer on the right remains stationary. This will cause the portion indicated in the figure to be cleared, even though both pursuers are confined to their cells. The critical information change actually occurs when the vertices from the two visibility polygons meet at an obstacle edge. This constraint defines a three-dimensional algebraic manifold in $\Re^4$ that partitions cells in X^p. The algebraic constraints that correspond to these types of cases significantly increase the implementation difficulty and add numerous cells which decreases practical efficiency.

Because the number of vertices in the cell graph is exponential in the number of pursuers, several issues relating to search efficiency remain to be addressed. Many cells can be combined without affecting completeness if it is shown that their common boundary does not represent a critical information change. The number of cells can be substantially reduced in many cases. Also, clever search strategies could keep much of the information space from being explored. One cause for this potential reduction is the fact that multiple solutions exist, including multiple states that each can represent the final step in a solution strategy. Even with these reductions, however, it seems unlikely that a complete, practical algorithm

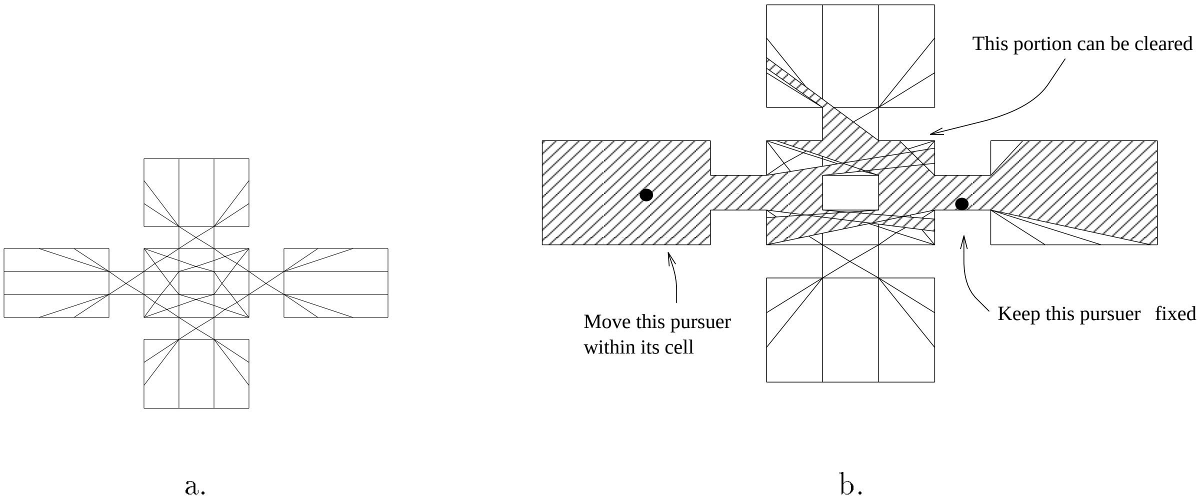


Figure 11: This example shows that the information state can be changed by a closed loop path that maintains constant edge visibility for the case of two pursuers: a) the cell decomposition based on edge visibility; b) overlapping visibility polygons permit the information state change.

can be developed for the case of $H(F) \geq 3$.

5 Computed Examples

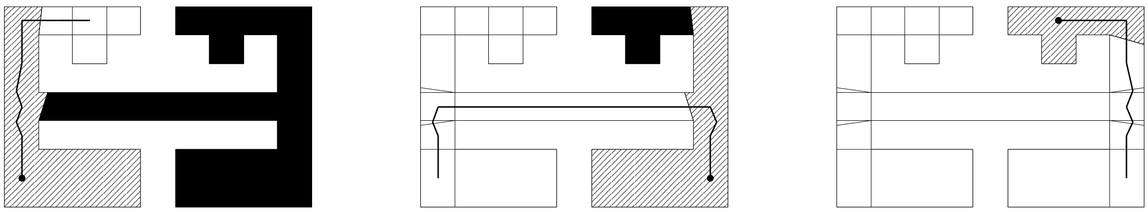
The complete algorithm is implemented in C++ and executed on an SGI Indigo2 workstation with a 200 Mhz MIPS R4400 processor. The computation times and other parameters for several examples are listed in Figure 12. The implementation uses the quad-edge structure from [8] to maintain the topological ordering of the conservative cells. The searching strategy is essentially Dijkstra's shortest path algorithm, where the distance is measured from the adjacent cell centroids. The solution is computed by traversing from cell centroids to cell centroids, causing the computed path for the pursuer to be jagged in most cases. In some applications, it might be appropriate to employ smoothing algorithms to the path to respect additional problem constraints.

Figures 13-17 show several computed examples. Due to a large number of conservative cells, Figures 15-17 are illustrated with the cell decompositions in separate diagrams from

Problem	Edges	Nodes in G_c (Cells)	Nodes in G_I (Information)	Precomp. Time (sec)	Searching Time (sec)	Total Time
Fig. 13	28	25	200	0.04	0.02	0.06
Fig. 14	68	130	1727	0.44	0.12	0.56
Fig. 15	46	237	8787	0.53	1.59	2.12
Fig. 16	65	246	18830	0.87	9.86	10.73
Fig. 17	70	888	103049	3.00	168.63	171.63

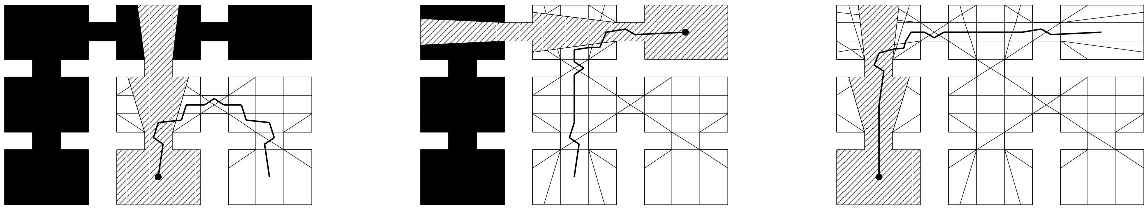
Figure 12: Various statistics are shown for the computed examples.

the solution diagrams. Figure 15 shows the hookpin example described in [25]. Note that the leftmost pin is recontaminated twice, and the pins are visited in the same order as mentioned in [25]. Figure 16 is an instance of the sequence described in Section 3 that requires a linear number of recontaminations. The region near the top of the figure is recontaminated 3 times. The final example generated a large number of conservative cells, which significantly increased computation time.



a. b. c.

Figure 13: A computed solution trajectory is shown in three frames. The black area represents the contaminated region, and the white area represents the cleared region. The thick curve shows a portion of computed trajectory, which is continued in each frame. The shaded region indicates the visibility region at the final time step of the indicated portion of the trajectory. The thin lines in the cleared region indicate the cell boundaries. In the final snapshot, there is no place remaining where the evader could be hiding.



a. b. c.

Figure 14: Another computed example.

6 Conclusions

A visibility-based planning problem has been identified in this paper that involves searching for an unpredictable evader in a polygonal environment. This task can represent a basic operation in a variety of robotic applications, such as surveillance with mobile robots. Other potential applications include search-rescue operations and military strategy.

Several bounds were obtained. A tight logarithmic bound on the number of needed pursuers was shown for the case of a simply-connected free space. A linear upper bound and square-root lower bound expressed in terms of the number of holes was also determined. It was also shown that there exist problems requiring a linear number of recontaminations. A few open problems remain, such as determining tight bounds on the number of pursuers for general polygons, and determining whether a polynomial-time algorithm exists to decide whether $H(F) = 1$.

Information space concepts were used to provide a natural characterization of the unique problem states. The visibility-based pursuit-evasion problem was established as NP-hard. The general concept of partitioning the information space on the basis of critical information changes was introduced to develop a complete algorithm. For the case in which $H(F) = 1$, the complete algorithm was implemented, and several examples were shown that were computed in a few seconds or less on a standard workstation. Considerable implementation issues remain for the case in which $H(F) = 2$, and in general, approximation algorithms might provide the only hope of obtaining practical solutions to many problems.

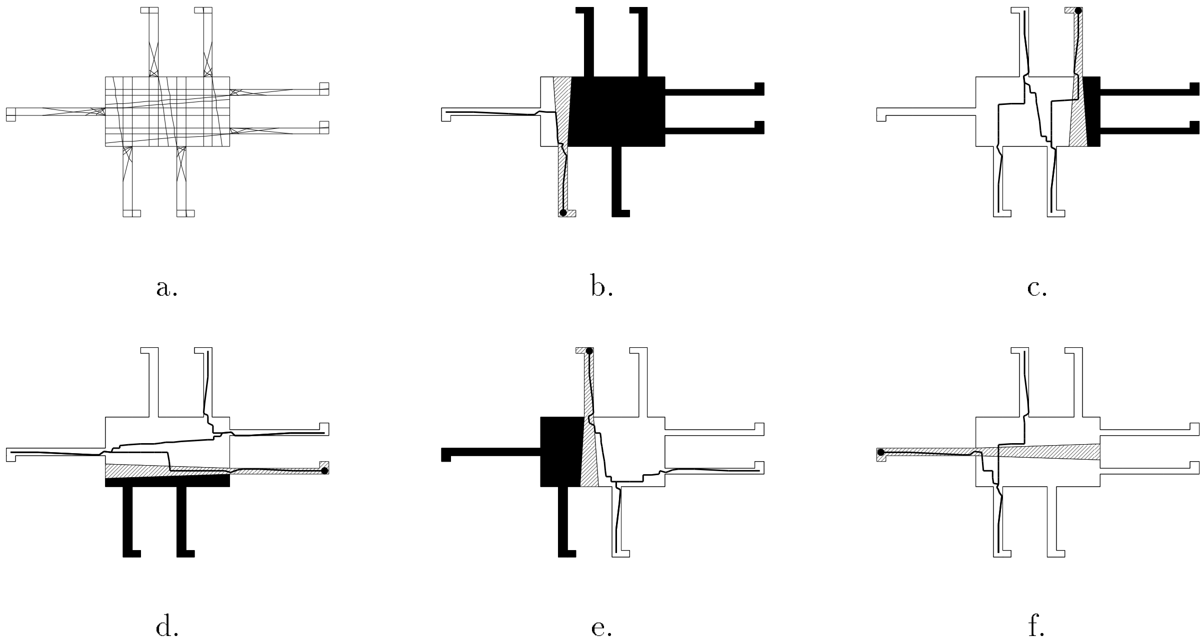


Figure 15: This difficult example requires two recontaminations of the leftmost corridor.

Several variations and extensions of the problem are worth exploring. In addition to a visibility region, each pursuer could have a region of capture, and the task could be to capture the evader using one or more pursuers. Using the current evader model, only connectivity issues become critical for determining a solution strategy; however, the problem can be made more challenging by strengthening the model to include a bounded velocity, or possibly stochastic prediction. The topological issues could become significantly more complex for 3-D free spaces. The conservative cell and edge-visibility concepts could be applied for the 3-D case, but considerable challenges would be faced to produce an efficient algorithm. Another problem variation is to consider a limited viewing angle, or a set of viewing rays as considered in [25]. A limited viewing angle can realistically occur in applications, and the problem can be extended to planning strategies that sweep viewing angles in addition to moving the pursuers. Finally, a cost functional could be additionally defined, leading to problems such as finding the evader in minimum time.

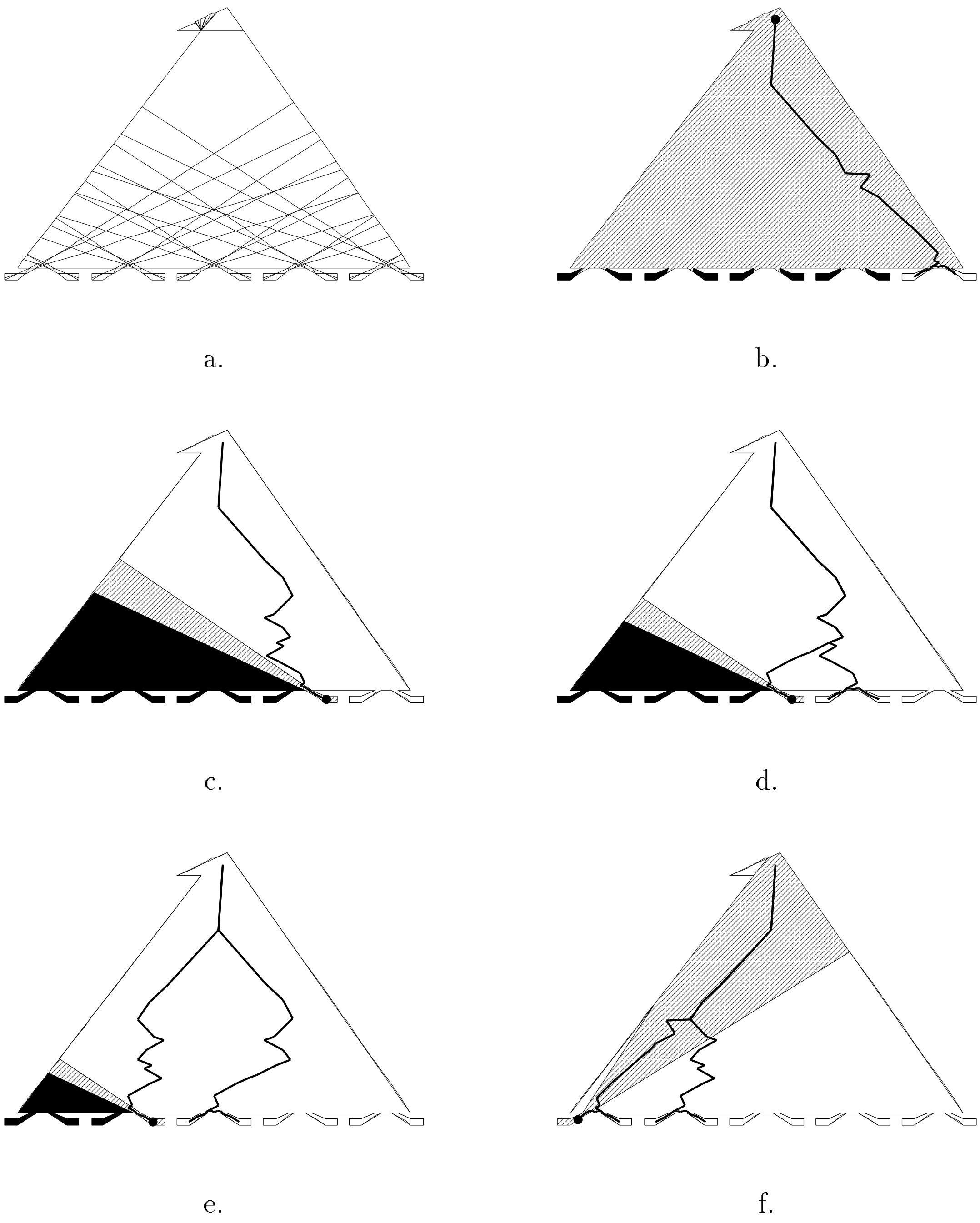
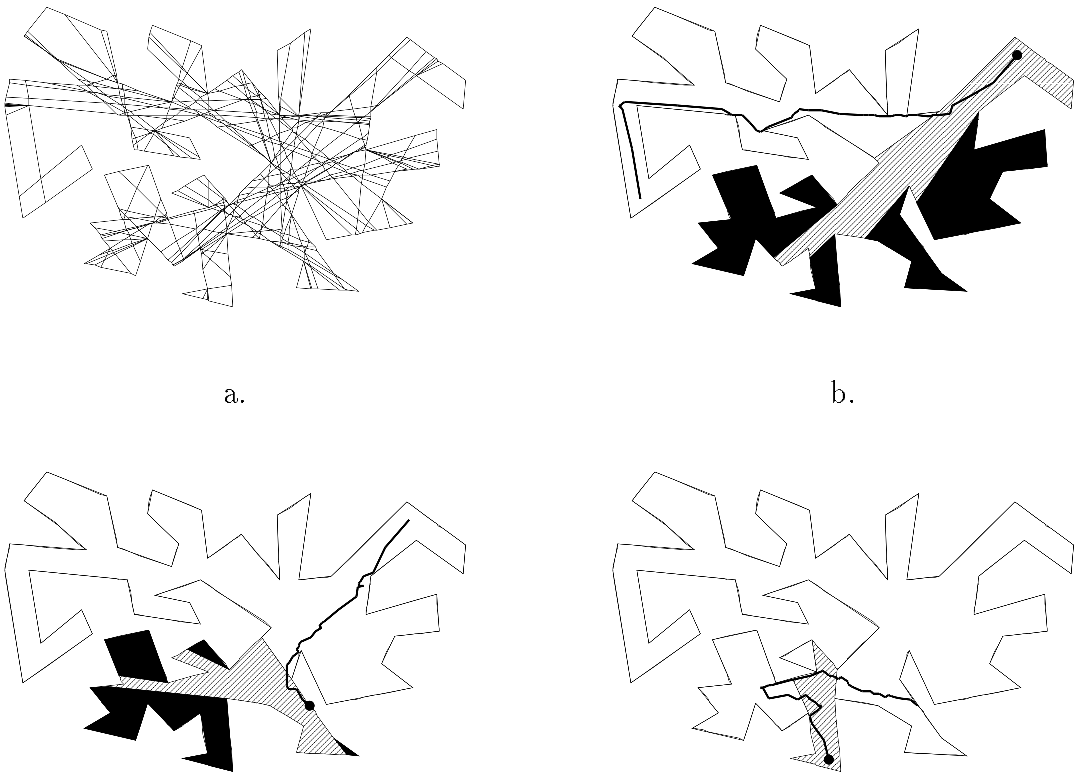


Figure 16: This example requires three recontaminations, and represents one in the sequence that requires a linear number of recontaminations.



a. b.

Figure 17: This bad example yields many edge-visibility cells.

Acknowledgments

The research of Leonidas J. Guibas is supported by NSF grant CCR-9623851 and US Army MURI grant 5-23542-A. Rajeev Motwani's research is supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation. The remaining researchers are supported by ARO MURI grant DAAH04-96-1-007 and ONR grant N00014-94-1-0721. The authors thank Julien Basch, Frédéric Cazals, Bruce Donald, Héctor González-Baños, Gary Kalmanovich, Jon Kleinberg, and Li Zhang, for their helpful suggestions.

References

[1] D. S. Arnon. Geometric reasoning with logic and algebra. *Artif. Intell.*, 37(1-3):37–60, 1988.

[2] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory.* Academic Press, London, 1982.

[3] J. Barraquand and P. Ferbach. Motion planning with uncertainty: The information space approach. In *IEEE Int. Conf. Robot. & Autom.*, pages 1341–1348, 1995.

[4] D. Bienstock and P. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12:239–245, 1991.

[5] J. F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[6] W.-P. Chin and S. Ntafos. Optimum watchman routes. *Information Processing Letters*, 28:39–44, 1988.

[7] M. Erdmann. Randomization for robot tasks: Using dynamic programming in the space of knowledge states. *Algorithmica*, 10:248–291, 1993.

[8] L. Guibas and J. Stolfe. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *AMC Trans. Graphics*, 4(2):74–123, 1985.

[9] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. 1st Workshop on Algorithmic Foundations of Robotics*, pages 269–282. A.K. Peters, Wellesley, MA, 1995.

[10] O. Hájek. *Pursuit Games.* Academic Press, New York, 1975.

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

[12] P. R. Kumar and P. Varaiya. *Stochastic Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[13] A. S. Lapaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, April 1993.

[14] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[15] J.-P. Laumond. Singularities and topological aspects in nonholonomic motion planning. In Z. Li and J. F. Canny, editors, *Nonholonomic Motion Planning*, pages 149–200. Kluwer Academic Publishers, Boston, MA, 1993.

[16] S. M. LaValle. *A Game-Theoretic Framework for Robot Motion Planning.* PhD thesis, University of Illinois, Urbana, IL, July 1995.

[17] F. Makedon and I. H. Sudborough. Minimizing width in linear layouts. In *Proc. 10th ICALP, Lecture Notes in Computer Science 154*, pages 478–490. Springer-Verlag, 1983.

[18] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, January 1988.

[19] B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted graphs. *Theoretical Computer Science*, 58:209–229, 1988.

[20] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, NY, 1987.

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

[22] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniqies for computing topological properties of algebraic manifolds. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[23] J. T. Schwartz and M. Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies. *Int. J. Robot. Res.*, 2(3):97–140, 1983.

[24] T. Shermer. Recent results in art galleries. *Proc. IEEE*, 80(9):1384–1399, September 1992.

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

[26] R. Talluri and J. K. Aggarwal. Mobile robot self-location using model-image feature correspondence. *IEEE Trans. Robot. & Autom.*, 12(1):63–77, February 1996.