IEEE International Conference on Robotics and Automation (ICRA)
San Francisco, April 24-28, 2000

# Pursuit-Evasion Using Beam Detection

Borislav H. Simov  Giora Slutzki  Steven M. LaValle

Dept. of Computer Science
Iowa State University
Ames, IA 50011 USA
{simov, slutzki, lavalle}@cs.iastate.edu

## Abstract

*We present an algorithm for searching a 2D environment for unpredictable moving targets using only beam-based detection. One or more pursuers move along the environment boundary, and carry a rotating beam that detects evaders. The beam could correspond in practice to a laser or a camera. The task is to compute motions for pursuers and their beams that ensure that all evaders will be detected. For a 2D polygonal environment, we solve a long-standing open problem by presenting a complete $O(n^3)$-time algorithm that is guaranteed to find a successful motion strategy for a single pursuer and its beam, if a solution exists. This algorithm is extended to the case of coordinating multiple pursuers, but the number of pursuers used in a solution is not necessarily optimal. An implementation is presented, and several computed examples are shown.*

## 1 Introduction

In recent years there has been a rising interest in robotics and computational geometry in designing motion strategies for pursuit-evasion scenarios. The basic task is to compute motion strategies for one or more robots (pursuers) to guarantee that unpredictable targets (evaders) will be detected using sensors. A key difficulty which makes the problem more challenging than basic exploration is that the evaders can sneak back to places already explored. Efficient algorithms that compute these strategies can be embedded in a variety of robotics systems to locate other robots and people. They can aid mobile surveillance systems that detect intruders using sonars, lasers, or cameras. Mobile robots can be used by special forces in high-risk military operations to systematically search a building in enemy territory before it is declared safe for entry. If robots have limited communication, the algorithms could be used to help robots locate each other for coordinated tasks such as map-building, localization, or target tracking. Beyond robotics, these algorithms could help in the training of firefighters engaged in a rescue effort, law enforcement officers in a hostage situation, or soldiers attempting to secure a potentially-hostile area. The strategies can also be used in virtual environments to help game developers design complicated pursuit strategies.

This paper focuses on a particular scenario that has simple sensing requirements. The evaders are unpredictable points that move continuously in a 2D environment. Each pursuer carries a thin detection beam that can be placed at any orientation. The beam could be implemented by a camera and vision system that uses feature detection to recognize a target. Alternatively, a single laser beam could be used. Unexpected changes in depth measurements would imply that an evader is blocking the beam, causing detection. We require that each pursuer moves along the boundary of the environment, or along the beam of another pursuer. If a single pursuer is used, this is not restrictive because it cannot protect its work from a position in the interior of the environment. For multiple pursuers, it is restrictive; however, this restriction can be beneficial in some cases. Localization is one of the most difficult problems in mobile robotics; however, wall-following is very reliable because one dimension has been eliminated from the localization problem. This might enable a pursuer robot to be constructed using less-expensive hardware. In fact, sensors could be mounted along tracks that are fastened to the walls of a building, as opposed to employing a general-purpose mobile robot. The pursuer could then be envisioned as an electric train that carries an inexpensive detection device.

Related problems have been considered in dynamic game theory, graph theory, computational geometry, and robotics. In game theory, pursuit-evasion scenarios, such as the Homicidal Chauffeur problem, express differential motion models for two opponents, and conditions of capture or optimal strategies are sought [5]. In graph theory, several 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 [1, 6, 10, 11, 12, 13]. The pursuer-evasion problem using a single detection beam in a polygonal environment was first introduced by Suzuki and Yamashita [15]. Although the problem has been open for a while, several variations have been considered. Lee et al [9] considered pursuit-evasion using beam detection in a room (i.e., a polygon with one door — a point which has to remain clear at all times) and presented a $O(n^2)$ solution. Icking and Klein [4] solved a search problem for two guards who move on the boundary of a polygon and are always mutually visible. Their result was later improved by Heffernan [3]. To the best of our knowledge, we present the first complete algorithm for the single-pursuer pursuit-evasion problem in a polygonal environment with a rotating beam, solving a problem originally posed in [15]. Pursuit-evasion in a 2D environment using 360° vision has also been considered [2, 7, 8, 15].

## 2 The Single Pursuer Problem

In this section, we assume that a single pursuer moves in a 2D polygonal world; the extension to multiple pursuers is presented in Section 4. Although we describe our method in terms of a polygonal environment, it is based on concepts that allow easy adaptation to curved environments [7].

The pursuer and evaders are modeled as points that move in a world, $P \in \mathbb{R}^2$, that is bounded by a simple polygon. An arbitrary number of evaders may exist in $P$. Let $e_i(t) \in int(P)$ denote the position of an *evader* at time $t \geq 0$, in which $int(P)$ denotes the interior of $P$. It is assumed that $e_i : [0, \infty) \to int(P)$ is a continuous function, implying that each evader is capable of executing arbitrarily fast motions. The initial positions $e_i(0)$ and the paths $e_i$ are assumed unknown to the pursuer.

Let $p(t) \in P$ denote the position of the *pursuer*. The pursuer is equipped with a *beam* that is modeled as a line segment in $P$, which emanates from $p(t)$, and terminates at its nearest intersection point on the boundary of $P$. Let $\theta(t) \in [0, 2\pi)$ denote the orientation of the beam. An evader is *detected* if it is "touched" by the beam at some time $t \geq 0$.

For a given world, $P$, the task is to determine both continuous functions, $p : [0, \infty) \to P$ and $\theta : [0, \infty) \to [0, 2\pi)$ for the pursuer such that all evaders will be detected in a finite amount of time. Note that the pursuer is required to move along the boundary of $P$

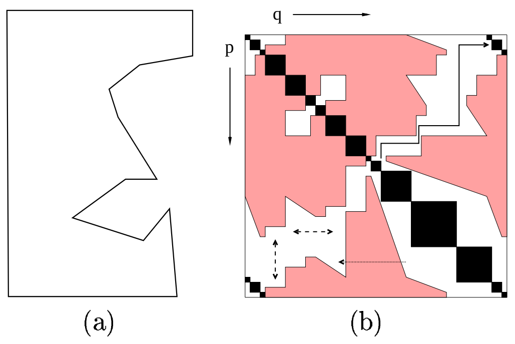

**Figure 1**. Simple polygon (a) and its corresponding state space (b)

at all times; otherwise, the evaders can sneak behind the beam. Thus without loss of generality, it can be assumed that $p : [0, \infty) \to \partial P$, in which $\partial P$ denotes the boundary of $P$. Note that $\partial P$ is a point set with an $S^1$ topology.

Using this restriction, note that the beam partitions $P$ into two regions. A point in $P$ is *cleared* if it is known not to contain any evaders; otherwise, it is *contaminated.* During the pursuit the region on one side of the beam is cleared, and the region on the other side is contaminated. Due to symmetry, we will assume without loss of generality that the cleared portion on $P$ is always to the left of the beam, when looking from the position of the pursuer.

Since the pursuer moves along $\partial P$, the configuration space can be simplified to a two-dimensional manifold, $S^1 \times S^1$, in which the first parameter indicates the position of the pursuer along $\partial P$, and the second parameter indicates the orientation of the beam. To better illustrate the concepts that follow, we use an alternative representation in which the second parameter corresponds to the intersection point of the beam in $\partial P$. Thus, we define a *state space*, $X = \partial P \times \partial P$, in which a state is represented as $(p, q)$ when the pursuer position is $p \in \partial P$, and the beam hits the position $q \in \partial P$. To preserve the original $S^1 \times S^1$ topology, we also allow states, $(p, q) \in X$ in which $q$ is not visible from $p$. The state space $X = \partial P \times \partial P$ can be partitioned into three sets:

- $X_d = \{(p, q) \in X \mid p$ and $q$ lie on the same edge$\}$
- $X_v = \{(p, q) \in X \mid p$ and $q$ are mutually visible$\}$
- $X_n = X \setminus (X_d \cup X_v)$

Figure 1 provides an example of a simple polygon and the state space corresponding to it. The black squares along the diagonal of Figure 1 represent $X_d$, the shaded area represents $X_n$, and the white area represents $X_v$. The motions of the pursuer and the rotations of the beam can be represented by corresponding moves in

$X$ as follows. A horizontal move in $X_v$ from left to right (right to left) denotes a rotation of the beam in clockwise (counterclockwise) direction and stationary pursuer (see the horizontal dashed line in Figure 1). Similarly, a vertical move down (up) in $X_v$ represents a motion of the pursuer in clockwise (counterclockwise) direction with the beam fixed at the same point (see vertical dashed line in Figure 1). Each of these moves is reversible. Apart from the four different moves within $X_v$ there is one more move which we call "recontamination"; it corresponds to a stationary pursuer who moves the beam to the left across a gap edge. In $X$ this is represented by a horizontal move to the left across a part of $X_n$ (see the dotted line in Figure 1). The move is not reversible, i.e., if the pursuer decides to move the beam back to the original position, this would contaminate the whole area to the left of the beam. For a more detailed example of recontamination, see Figure 5.

How is the search defined in terms of the diagram? It corresponds to finding a path in $X$ which is a sequence of the $4 + 1$ basic moves described above. The path starts immediately above and to the left of the diagonal and reaches the diagonal from the left or from below. (Remember that there is a vertical and horizontal wraparound along each of the axes.) The path cannot cross the diagonal since this would correspond to the pursuer pointing the beam outside the polygon. The solid black path in Figure 1 from the center to the upper right corner, represents a successful search by the pursuer.

## 3 Structure of the State Space

This section presents a complete algorithm that solves the single-pursuer problem by capturing the topology of the state space, $X$, and obtaining a combinatorial representation through the identification of critical events. This is similar to the philosophy used in [14] for classical path planning, and builds on previous efforts in visibility-based pursuit-evasion [7, 8].

**A cell decomposition of $\partial P$.** The first step is to partition $\partial P$ into open intervals and critical points. Let $V(p)$ denote the set of all points along $\partial P$ that are visible to the pursuer at position $p \in \partial P$. Each open interval identifies a maximal set of positions from which the topology of $V(p)$ remains invariant. The same connected components appear, although they may be shifted in a continuous manner made precise by piecewise homotopy arguments in [7]. Let $d : \partial P \times S^1 \to \Re$ denote the real-valued function that corresponds to the ideal distance measurements. The value $d(p, \theta)$ gives the distance from $p$ to another point on $\partial P$ by placing the beam at angle $\theta$. We call each data discontinuity a *gap* in $d(p)$ ($d(p)$ is considered as a function of $\theta$). Intuitively, piecewise homotopy can be considered as a generalization of classical homotopy to piecewise continuous functions. The gaps must move continuously in the piecewise homotopy. Note that from any $p$, the set of gaps partitions $\partial P$ into an alternating sequence of visible and invisible intervals. Each visible interval corresponds to one topologically-distinct direction in which the beam can be aimed.

The next task is to determine the points along $\partial P$ at which a critical change occurs in terms of this alternating sequence of visible intervals. These critical changes correspond directly to changes in gaps in $V(p)$. As the pursuer moves along $\partial P$, one of four changes can occur: 1) a gap disappears, 2) a gap appears, 3) two gaps merge into one, or 4) one gap splits into two. If $\partial P$ is not in general position, then several events could occur simultaneously; this technicality is excluded from the discussion (yet the algorithm still applies). The first two types of changes occur at *inflections*, and the last two occur at *bitangents*.

**Inflections.** Inflections either cause the set of topologically-distinct beam directions to change, or they cause an invisible interval to appear or disappear along the horizon (the limiting case in which the beam is nearly parallel to the edge(s) that contains the pursuer). Let $v_0, v_1, \ldots, v_{n-1}$, represent the set of polygon vertices in clockwise order. A vertex, $v_i$, is called a *reflex vertex* if $v_{i+1}$ lies to the left of the ray from $v_{i-1}$, through $v_i$ (this assumes modularity of indices, $v_{-1} = v_{n-1}$, and $v_n = v_0$).

An *inflection point* is a reflex vertex $v_i$, which is adjacent to a non-reflex one, $v_{i\pm1}$. The point, $p_i \in \partial P$, that is touched by a ray extended from $v_i$, in the direction of $\overrightarrow{v_{i\pm1}v_i}$, is called the *inflection image* of $v_i$. The pairs of points $(2, 10)$ and $(0, 9)$ (connected with a dotted line in Figure 2) show pairs of an inflection point and its image. The inflection point is a critical event because as the pursuer moves from Point 3 to Point 1, part of $\partial P$ between Point 3 and Point 2 is no longer visible. The inflection image is a critical event because all of $\partial P$ from Point 3 to Point 2 becomes visible as the pursuer moves from Point 11 to Point 9 (an invisible interval vanishes). Both kinds of inflections cause critical events in terms of the pursuit, and will be used in our combinatorial representation of the pursuit.

**Bitangents.** Bitangents either cause two visible intervals to merge into one, or cause one visible interval to split into two. A pair of reflex points, $v_i$ and $v_j$ are each called *bitangent points* if: 1) both edges incident to $v_i$ lie on the same side of the line through $v_i$ and $v_j$, 2) both incident to $v_j$ lie on the same side of the line (although these may lie on a different side than the edges incident to $v_i$), and 3) $v_i$ and $v_j$ are mutu-

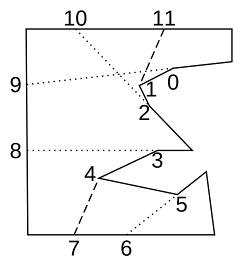

**Figure 2**. Cell decomposition of the polygon in Fig 1.

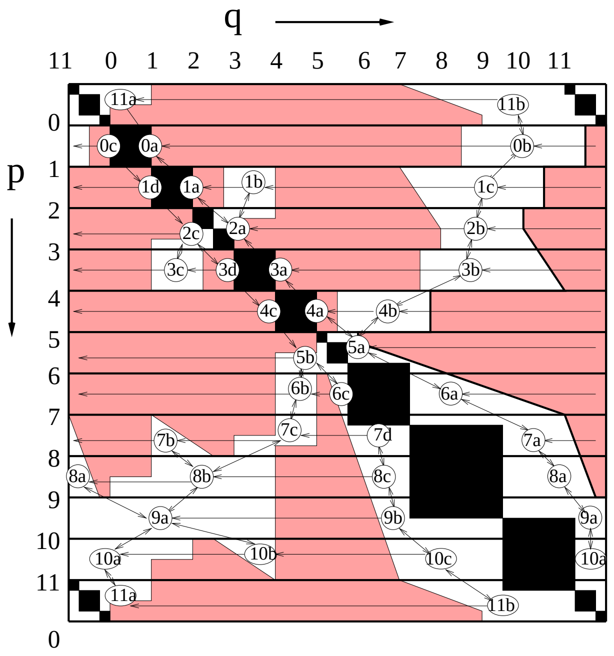

**Figure 3**. State space for the polygon from Fig 2.

ally visible. Points 1 and 4 of Figure 2 are bitangent points.

Note that the line through $v_i$ and $v_j$ always intersects $\partial P$ at two places other than $v_i$ and $v_j$, before leaving $P$. These two intersections are called *bitangent images*. Points 7 and 11 in Figure 2 are bitangent images. When a bitangent image is crossed, the number of topologically-distinct directions in which the beam can be aimed either decreases by one or increases by one. For example, if the pursuer moves along $\partial P$ from Point 6 to Point 8, the number increases because the pursuer is able to see into the "pocket" between Points 1 and 4. In Figure 2 the each pair of a bitangent point and its image is connected with a dashed line.

**The pursuit graph.** Suppose that we now "slice" $X$ into horizontal rows along the critical boundaries of $\partial P$. Within each slice, starting to the right from the diagonal, there will be alternating white and shaded regions. All the points in a white region within the slice correspond to a single topologically-distinct direction to aim the beam from a given position of the pursuer. So instead of considering the transitions within the continuous space of points in $X_v$, we are going to substitute all the white points from the region with a single vertex in a graph. Also, vertices in the graph will be connected if they correspond to neighboring white regions in $X$. The edges between neighboring vertices from different rows are two-directional and correspond to the four reversible moves of the pursuer. In order to take into account the possibility of a recontamination move, we also add unidirectional edges from every vertex to its left neighbor (if one exists) within the same row, but not across the diagonal.

This naturally reduces the state space $X$ to a directed graph, $G = (V, E)$, that has $O(n^3)$ vertices and edges, where $n$ is the number of vertices in the bounding polygon of $P$. (Each of the $O(n^2)$ rows contribute $O(n)$ vertices to $G$.) For example, the state space as described in Figure 1 is overlaid with its corresponding graph $G$ and the result is shown in Figure 3.

Define the initial vertices to be the ones which are immediately to the right of the diagonal. (For example, in Figure 3 these are vertices $0a, 1a, 2a, \ldots, 11a$.) Similarly, define the goal vertices as the ones which are immediately to the left of the diagonal. (For example, in Figure 3 these are vertices $0c, 1d, 2c, \ldots, 11b$.) A successful search of the polygon corresponds to a path from an initial vertex to a goal vertex. Furthermore, if we add a *start* node $S$ which has only outgoing edges to all the initial vertices, and also a *finish* node $F$ which has only incoming edges from all the goal vertices, then finding a solution path will be equivalent to finding a path in $G$ from node $S$ to $F$.

The graph representation can be easily constructed in time $\Theta(n^3)$, and the graph can be searched in time $\Theta(n^3)$ for a solution path using a standard method such as breadth-first search. The graph search algorithm finds a solution if and only if the polygon is solvable. The proof is based on the fact that by construction, every node in the graph corresponds exactly to a topologically equivalent set of pairs $(p, q)$ in $X$. Similarly, every transition in the graph corresponds exactly to a movement of the pursuer or rotation of the beam. Thus solving the graph search problem in $\Theta(n^3)$ is equivalent to solving the pursuit-evasion problem in $\Theta(n^3)$.

## 4 Simple Multiple-Pursuer Extension

In this section we assume that more than one pursuer is necessary to detect all of the evaders. We continue to restrict the pursuer to move along $\partial P$, but note that in general, one might wish to consider solutions in which pursuers move in the interior of $P$ (unless a particular robot system prohibits this). Although the single-pursuer algorithm is restricted to a simply-connected environment, the multiple pursuer algorithm also applies to multiply-connected environments (i.e., polygons with holes). The multiple-pursuer algorithm is complete in the sense that it will find a solution when a solution exists; however, it is not guaranteed to use the optimal number of pursuers.

The single-pursuer algorithm is used as a module in the multiple-pursuer algorithm. This general idea was also used in [8] for visibility-based pursuit evasion. The multiple-pursuer algorithm progressively uses the single pursuer algorithm to clear as much of $P$ as possible, in terms of the number of inflections that lie to the left of the beam. The remaining, contaminated portion of $P$ is then cleared using another pursuer that executes the single-pursuer algorithm on the remaining subpolygon. If it succeeds then a two-pursuer solution has been determined; otherwise, the algorithm attempts to use a third pursuer, after the second pursuer clears as much a possible. The process continues until the entire environment is cleared. A number of possible variations are obvious by designing different heuristics for stopping a pursuer while other pursuers continue. There is also the issue of recovering a pursuer that has finished clearing one portion, to be used for clearing another portion. In this paper, we present one simple coordination algorithm and show computed results.

## 5 Implementation

Both the single-pursuer and multiple-pursuer algorithms were implemented using GNU C++ and the Library of Efficient Data Types and Algorithms (LEDA), and experiments were performed on a Pentium III 500Mhz PC running Linux. The algorithms were determined to be efficient enough for practical use in real environments. The running time to compute a solution for a polygon with 253 vertices, 130 inflection points, and 394 bitangents is 5 seconds, with very little optimization of the code performed. Figures 4 and 5 show examples that were solved using the complete, single-pursuer algorithm. Figure 6 shows an example that was solved using multiple pursuers.

Figure 4 corresponds to the polygon already discussed in Figures 1, 2, and 3. It has a quite simple solution. The pursuer starts from the upper left corner. In the beginning and in the end the pursuer moves and simultaneously rotates the beam. In between, without loss of generality we can assume that the movements of the pursuer and the rotations of the beam do not happen at the same time, rather they alternate. Note that the position of the pursuer on the boundary is designated with a small white circle.

Figure 5 is a polygon which requires recontamination in order to be searched successfully. The recontamination happens between frames (d) and (e): some of the area which was already cleared is contaminated again after the pursuer rotates the beam to the left.

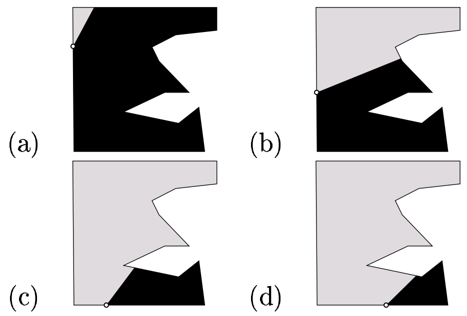

**Figure 4**. Solution for the polygon in Fig 1.

Finally, Figure 6 is an example of search by multiple pursuers. Frame (c) shows the maximum area (white) cleared by the first pursuer. Frame (d) corresponds to the first pursuer becoming stationary and the second pursuer taking over the search. In frame (f) the second pursuer has cleared another part of the polygon, so a third pursuer can finish the search. As we already noted, sometimes pursuers can be “recycled”, i.e., when the second pursuer is finished, the first one is no longer needed in the previous position. Thus the first pursuer can finish the search without actual need for a third one.

## 6 Discussion

We presented algorithms that compute motion strategies for pursuers that must detect all unpredictable evaders in a polygonal environment using rotating beams. A complete algorithm was presented for the one-pursuer case, which solves an open problem posed in [15]. This was extended to a greedy algorithm for the case of multiple pursuers, but it does not necessarily use an optimal number of pursuers. The simulation results presented in this paper are very encouraging. A logical next step is to assess the value of our model through experimentation. This could be performed, for example, on any indoor mobile robot, using a color camera or laser scanner for detection.

We believe that some of the ideas presented in this paper can be extended to other pursuit-evasion and visibility problems. A variety of improvements and alternatives can be considered for our multiple-pursuer algorithm, especially in the choice of stopping point for each pursuer. Also, a recursive algorithm can be considered that attempts to solve and evaluate multiple subproblems using the single-pursuer algorithm.

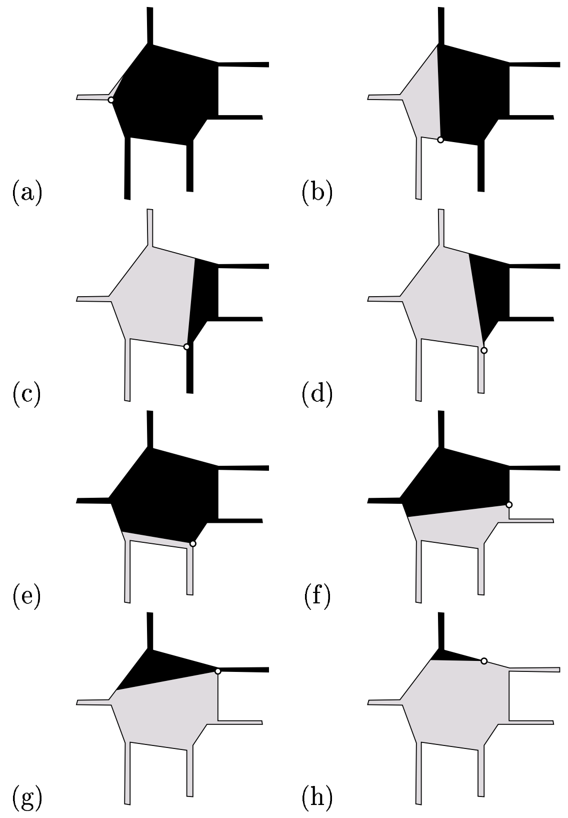

**Figure 5**. A single-pursuer example which requires recontamination (between frames (d) and (e)).

## Acknowledgments

We are grateful for the funding provided in part by NSF CAREER Award IRI-9875304 (LaValle).

## References

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

[2] D. Crass, I. Suzuki, and M. Yamashita. Searching for a mobile intruder in a corridor – the open edge variant of the polygon search problem. *Int. J. Comput. Geom. & Appl.*, 5(4):397–412, 1995.

[3] P.C. Heffernan. An optimal algorithm for the two-guard problem. In *Proc. ACM Symp. Comp. Geom.*, pp. 348–358, 1993.

[4] C. Icking and R. Klein. The two guards problem. In *Proc. ACM Symp. Comp. Geom.*, pp. 166–175, 1991.

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

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

[7] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: An extension to curved environments. In *Proc. IEEE ICRA '1999*, pp. 1677-1682, 1999.

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

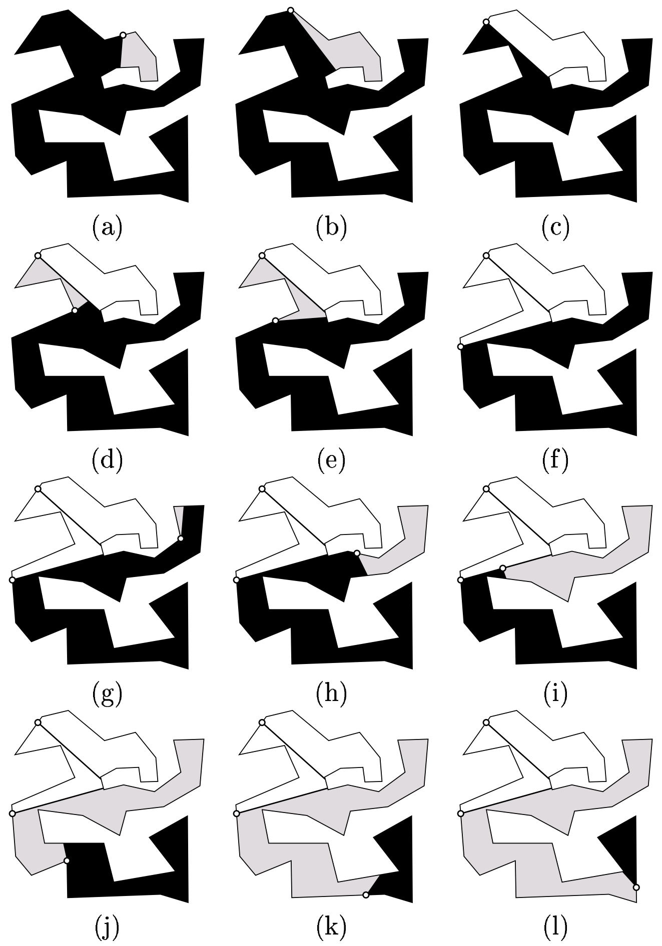

**Figure 6**. Multiple-pursuer example, which requires two pursuers and tree runs of the single-pursuer algorithm.

[9] J.H. Lee, S.M. Park, and K.Y. Chwa. Searching a polygonal room with a door by a 1-searcher. Manuscript, 1999.

[10] F. Makedon and I. H. Sudborough. Minimizing width in linear layouts. In *Proc. 10th ICALP, LNCS 154*, pp. 478–490, 1983.

[11] 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.

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

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

[14] 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.

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