# An Optimal Algorithm for the Two-Guard Problem

(Extended Abstract)

Paul J. Heffernan *
Dept. of Mathematical Sciences, Memphis State University, Memphis, TN 38152

## Abstract

In this paper we give optimal solutions for two versions of the *two-guard* problem. Given a simple polygon $P$ with vertices $s$ and $t$, the *straight walk* problem asks whether we can move two points monotonically on $P$ from $s$ to $t$, one clockwise and one counterclockwise, such that the points are always co-visible. In the *counter walk* problem, both points move clockwise, one from $s$ to $t$ and the other from $t$ to $s$. We provide $\Theta(n)$ constructive algorithms for both problems. We obtain our results by examining the structure of the restrictions placed on the motion of the two points, and by employing properties of shortest paths and shortest path trees.

## 1 Introduction

Work on polygonal visibility is essentially a study of structure: given a polygon, what properties does it exhibit, and how efficiently can these properties be determined? One interesting area of visibility is guard problems, where it is asked whether a set of point guards can see all points inside a polygon. In this paper we concentrate on the "two-guard walkable" problem, which was introduced by Icking and Klein [IK]. This problem takes as input a simple polygon $P$ and two specially designated vertices of $P$, which we call $s$ and $t$. The vertices $s$ and $t$ partition $P$ into two chains, $L$ and $R$. If we think of the interior of $P$ as a street in a dangerous section of town, complete with alleys and side streets, and $L$ and $R$ as the sidewalks on either side of the street, then the *straight walk* problem asks the following: can two guards walk down the street from $s$ to $t$, one on each sidewalk, without backtracking, while always staying in sight of each other? The *counter walk* problem is similar, except that one guard walks on his sidewalk from $s$ to $t$ while the other walks from $t$ to $s$.

Describing a straight walk or a counter walk on a polygon $P$ with $n$ vertices carries a trivial lower bound of $\Omega(n)$, as the two guards visit every vertex during their patrol. It is natural to ask whether we can test for existence of a walk and construct a walk in $O(n)$ time. Many fundamental questions that concern polygonal structure have been solved in linear time, including the computation of the visibility polygon from a point, triangulation of a polygon, and construction of the shortest path tree from a vertex. We will show that a linear-time solution exists for the two-guard problem, as well.

In this paper we give optimal $\Theta(n)$-time algorithms that determine the existence of straight and counter walks, and construct such walks if they exist (the counter walk algorithm is omitted from this extended abstract because of space constraint). We thereby improve upon the results of [IK], who provide $O(n \log n)$-time constructive algorithms for both cases. The methods of [IK] are locked into the $O(n \log n)$ time complexity

*Supported in part by the U.S. Army Research Office, Grant No. DAAL03-92-G-0378


**9th Annual Comptutational Geometry,5/93/CA,USA**
**© 1993 ACM 0-89791-583-6/93/0005/0348...$1.50**

from the start, since they perform ray shooting queries (in $O(\log n)$ time per shot) for each of the reflex vertices of $P$. We take a closer look at the need for ray shooting, and learn how the number of ray shooting queries can be reduced. Then, we employ the structure of shortest paths and shortest path trees to compute all essential ray shooting queries in only linear time. We find it necessary to develop a series of definitions and lemmas concerning shortest paths, while also using a number of shortest path results from the literature.

Guard problems (or "watchman problems") have been studied by several researchers. An overview can be found in [O'R]. The number of stationary guards needed to see all points in a polygon is studied in [Chv] and [Fi], while [LL] show that obtaining the minimum number of such guards is NP-hard. In [Sh] these results are generalized to "link visibility", where guards can see around one or more corners. Mobile guard problems, as studied in [CN], ask for paths which see all points. An $O(E + n \log n)$-time algorithm in [MW] gives minimum-length paths for a pair of guards who must remain co-visible while patrolling a polygonal domain with holes, where $n$ is the number of vertices on all holes and $E$ is the size of the visibility graph.

## 2 Notation

We define notation for this paper; much of our notation is borrowed from [IK]. A *polygonal chain* in the plane is a concatenation of line segments. The endpoints of the segments are called *vertices*, and the segments themselves are *edges*. If the segments intersect only at the endpoints of adjacent segments, then the chain is *simple*, and if a polygonal chain is closed we call it a *polygon*. In this paper, we deal with a simple polygon $P$, and its interior, $int(P)$. Two points $x, y \in P$ are *visible* (or *co-visible*) if $\overline{xy} \subset P \cup int(P)$. We assume that the input is in general position, which means that no three vertices are collinear, and no three lines defined by edges intersect in a common point.

If $x$ and $y$ are points of $P$, then $P_{CW}(x,y)$ ($P_{CCW}(x,y)$) is the subchain obtained by traversing $P$ clockwise (counterclockwise) from $x$ to $y$.

In the problems we consider, two vertices of $P$ are specially designated as the start vertex, $s$, and the goal vertex, $t$. We refer to the subchains $P_{CW}(s,t)$ and $P_{CCW}(s,t)$ as $L$ and $R$, respectively; both $L$ and $R$ are oriented from $s$ to $t$. Points on $L$ ($R$) are denoted by $p$, $p'$, $p_1$, etc. ($q$, $q'$, $q_1$, etc.). If $p$ is a vertex of a polygonal chain, $Succ(p)$ represents the vertex of the chain immediately succeeding $p$, and $Pred(p)$ the vertex preceding $p$.

For $p, p' \in L$, we say that $p$ *precedes* $p'$ (and $p'$ *succeeds* $p$) if we encounter $p$ before $p'$ when traversing $L$ from $s$ to $t$. We write $p < p'$. The chain $L_{<p}$ ($L_{>p}$) is the subchain of $L$ consisting of all points that precede (succeed) $p$. The chain $L_{p,p'}$ is the subchain of all points that succeed $p$ and precede $p'$. We make analogous definitions for $R$.

A *(general) walk* is a pair of continuous functions

$$f : [0,1] \to L, \quad g : [0,1] \to R,$$

where $f(0) = g(0) = s$, $f(1) = g(1) = t$, and $f(t)$ and $g(t)$ are co-visible for all $t$. A walk is *straight* if $f$ and $g$ are monotonic functions; that is, $f(t_1) \leq f(t_2)$ and $g(t_1) \leq g(t_2)$ whenever $t_1 < t_2$ (see Figure 1). A *counter walk* is a pair of monotonic, continuous functions with $f(0) = g(1) = s$ and $f(1) = g(0) = t$.

For points $x$ and $y$, $d(x,y)$ is the direction of the directed segment from $x$ to $y$. The (undirected) segment between $x$ and $y$ is $\overline{xy}$. The ray $\vec{r}(x,y)$ is the ray with terminus $x$ in direction $d(x,y)$. The line containing $x$ and $y$ is denoted $\ell(x,y)$, and the directed line from $x$ through $y$ is $\vec{\ell}(x,y)$. An important definition is that of *(ray) shots*: the *backward (ray) shot* (or *hit point*) from a reflex vertex $p \in L$, denoted $Backw(p)$, is the first point of $P$ encountered by a "bullet" shot from $p$ in direction $d(Succ(p),p)$, and the *forward (ray) shot* $Forw(p)$ is the point encountered by the bullet shot from $p$ in direction $d(Pred(p),p)$ (see Figure 2). We let $d(Backw(p))$ represent the direction of the backward shot from $p$, i.e. $d(Succ(p),p)$. We define the backward shot

$Backw(q)$ and forward shot $Forw(q)$ for a reflex vertex $q \in R$ in similar fashion. (Note that a shot cannot hit a vertex, by the assumption of general position.)

A vertex $p$ of a polygonal chain is a *left turn* (*right turn*) if $Succ(p)$ lies on the left (right) side of the directed line $\vec{\ell}(Pred(p), p)$.

Two rays with common endpoint $x$ partition the plane into two regions, each of which is the union of a set of rays with endpoint $x$. We call the region containing all rays encountered as we sweep from $\vec{r}(x, y_1)$ to $\vec{r}(x, y_2)$ counterclockwise a *cone*, and denote it $cone(\overline{xy_1}, \overline{xy_2})$ (or $cone(y_1, x, y_2)$). We can also think of a cone as an interval of directions.

## 3 Shortest Paths

The algorithms of this paper are based on the properties of shortest paths in a polygon. We now give definitions concerning shortest paths, and establish some visibility lemmas based on their properties.

The *shortest path* between two vertices $w$ and $v$ of a simple polygon $P$, denoted $SP(w, v)$, is the (Euclidean) minimum-distance curve with endpoints $w$ and $v$ lying entirely in $P \cup int(P)$. Shortest paths are unique. This means that two shortest paths cannot cross twice, since this would imply distinct shortest paths between a pair of points. The path $SP(w, v)$ is always a polygonal chain, whose vertices are also vertices of $P$. This can be seen by a local analysis: if one of the above two conditions is violated, some small amount of local improvement is possible. By a similar argument, we have the following.

**Lemma 1** *If $w$ and $v$ are vertices of $P$, and $SP(w, v)$ is the shortest path directed from $w$ to $v$, then any vertex of $SP(w, v)$ that lies on $P_{CW}(w, v)$ is a left turn, while a vertex of $SP(w, v)$ on $P_{CCW}(w, v)$ is a right turn.*

We write $FE(w, v)$ to denote the *first edge* of $SP(w, v)$; that is, the edge of $SP(w, v)$ incident to $w$. The direction of this edge away from $w$ is denoted $dFE(w, v)$. The *parent* of $w$ is the vertex of $SP(w, v)$ adjacent to $w$; in other words, it is the other endpoint of $FE(w, v)$. The *shortest path tree* from a vertex $v$ of $P$, denoted $SPT(v)$, is the union of all shortest paths $SP(v, w)$, for $w$ a vertex of $P$. If we think of the vertices of $P$ as nodes, and the segments of $SPT(v)$ as arcs, then we can think of $SPT(v)$ as a graph. The graph $SPT(v)$ is acyclic and connected, and therefore is a spanning tree.

A path of paramount importance in our discussion is $SP(s, t)$, the shortest path from $s$ to $t$. We will refer to this path as $C$, for it is the center path from $s$ to $t$, lying between the left path $L$ and the right path $R$. Points of $C$ will be denoted $r$, $r'$, etc. The chain $C$ is oriented from $s$ to $t$, and we say that $r$ precedes $r'$ (or $r < r'$) if $r$ is closer to $s$ on $C$ than is $r'$. We define $C_{<r}$, $C_{>r}$, and $C_{r,r'}$ analogously to $L_{<p}$, $L_{>p}$, and $L_{p,p'}$. Every vertex of $C$ is a vertex of $P$. Furthermore, if a vertex of $C$ is a left turn, then the vertex lies on $L$, while a right turn lies on $R$ (by Lemma 1).

For $p \in L$, we define $C_s(p)$ to be the point of $SP(p, s) \cap C$ nearest to $p$ on $SP(p, s)$. In other words, it is the first point of $C$ that we encounter upon traveling from $p$ to $s$ by the shortest route. We define $C_t(p)$, $C_s(q)$, and $C_t(q)$ similarly. Unless $p \in C$ (in which case $C_s(p) = p$), the point $C_s(p)$ must be a vertex of $C$.

If we think of $C_s$ as a function on the points of $L$, then $C_s$ is monotonic:

**Lemma 2** *If $p_1, p_2 \in L$ and $p_1 < p_2$, then $C_s(p_1) \leq C_s(p_2)$.*

Earlier we defined a *cone* around a point. Of special interest are the cones $cone(FE(q, t), FE(q, s))$ for a vertex $q \in R$ and $cone(FE(p, s), FE(p, t))$ for a vertex $p \in L$; since we will refer to these cones often, we will denote them simply as $cone(q)$ and $cone(p)$, respectively (see Figure 3).

**Lemma 3** *A bullet shot from $p \in L$ inside $P$ in direction $\alpha$ hits $R$ if $\alpha \in cone(p)$, and the bullet hits $L$ if $\alpha$ is not in the cone. A similar property holds for $q \in R$.*

**Corollary 1** *Given $p \in L$, $q \in R$, if $SP(p, q)$ contains points $p' \in L_{<p}$ and $p'' \in L_{>p}$ ($p' \neq s$,*

$p'' \neq t$), then $p$ is not visible from any point of $R$.*

**Lemma 4** *Suppose $p \in L$, $q \in R$ are not both on $C$. The points $p$ and $q$ are co-visible if and only if*

$$C_s(p) \leq C_t(q),\ C_s(q) \leq C_t(p)$$
$$q \in cone(p),\ p \in cone(q)$$

A *bridge* between points $p \in L$ and $q \in R$ is an edge of $SP(p, q)$ with one endpoint on $L$ and the other on $R$.

We now discuss some of the computational considerations involving shortest paths. The shortest path between two points of a triangulated polygon can be computed in linear time [LP], and the shortest path tree from a vertex $v$ of a triangulated polygon can be computed in linear time [GHLST]. A polygon can be triangulated in linear time by the algorithm of Chazelle [Cha]. While the Chazelle algorithm is necessary to obtain optimal worst-case time-bounds for the straight walk algorithm, it can be avoided in the counter walk case. For a polygon $P$ to be counter-walkable from $s$ to $t$, these points must be co-visible. The chord $\overline{st}$ partitions $P$ into two subpolygons, each of which must be weakly-visible from $\overline{st}$. A polygon can be tested for weak-visibility in linear time [AT], and if found to be weakly-visible, it can be triangulated in linear time by one of several uncomplicated algorithms [ET, He].

It is possible to modify the shortest path tree $SPT(s)$ so as to obtain a triangulation of $P$, through the addition of Steiner points. A Steiner point is created by adding a vertex to the interior of an edge, thereby splitting the edge into two. For each reflex vertex $v$, we extend the last edge of $SP(s, v)$ (if possible) and insert a Steiner point at the end of the extension. With the addition of these Steiner points as vertices of $P$, $SPT(s)$ is a triangulation of $P$; that is, $SPT(s)$ partitions $P$ and its interior into triangular regions.

An important characterization of $P$ with its Steiner points added is that for any pair of adjacent vertices, $v$ and $v'$, their parents in $SPT(s)$ are either equal or adjacent. More precisely, all interior points of an edge have the same parent. We call this point the *parent* of the edge. Also, all interior points of an edge have the same value $C_s$, so we can define $C_s$ on the edges of $P$.

For our algorithms, we construct the Steiner points with respect to $SPT(s)$, and with respect to $SPT(t)$, and add them to the set of vertices of $P$. (This creates a violation of the non-degeneracy assumption, since it allows a triple of vertices to be collinear, but every such triple must come from an original edge of $P$.) The Steiner points can be constructed and added to $P$ in linear time through an adaptation of the algorithm of [GHLST] (note that only $O(n)$ Steiner points are constructed). Once $P$ has been modified in this manner, all interior points of an edge have the same parent with respect to $SPT(s)$ and with respect to $SPT(t)$, as well as the same values of $C_s$ and $C_t$. We can compute the parents for all vertices and edges through one traversal of $SPT(s)$ and one of $SPT(t)$. If we observe that each vertex $w \in C$ has $C_s(w) = w$, and a vertex $w \notin C$ has $C_s(w) = C_s(w')$ where $w'$ is the parent of $w$ with respect to $SPT(s)$, we see that a single depth-first traversal of $SPT(s)$ can generate all values $C_s$ for $L$ and $R$. Similarly, a traversal of $SPT(t)$ can compute $C_t$ for $L$ and $R$. Our algorithms perform these preprocessing steps in order to easily employ Lemma 4. This lemma states that visibility between two points of $L$ and $R$ not both on $C$ can be determined in constant time, if each of the points can query its cone and its values of $C_s$ and $C_t$ in constant time. Since knowledge of the parents of a point in $SPT(s)$ and $SPT(t)$ suffices to compute the point's cone, and the preprocessing we have mentioned can be performed in $O(n)$ time, we have the following lemma.

**Lemma 5** *After an $O(n)$-time preprocessing step on $P$, we can determine in $O(1)$ time whether two points $p \in L$ and $q \in R$, not both on $C$, are visible.*

Our algorithm for the straight walk case requires that for a point $x \in P$, we be able to traverse $SP(x, t)$ starting from $x$. We allow ourselves to do this easily if we store $SPT(t)$ as a directed graph, where each edge is directed from a vertex $v$ to its parent in $SP(v,t)$. Additionally, for each edge of $P$, we store the parent of the edge in $SPT(t)$ by means of a pointer from the edge to the appropriate node in the directed graph $SPT(t)$.

In order for $P$ to have a walk (straight, counter, or general), $L$ and $R$ must be *weakly visible*; that is, each point of $L$ ($R$) must be co-visible with at least one point of $R$ ($L$). [IK] give a simple test for weak visibility, but it requires knowledge of the shots $Backw(v)$ and $Forw(v)$ for all reflex vertices $v \in P$. We employ an alternate test that requires only $O(n)$ time. While we omit the description here because of space restrictions, the test is based on the fact that $L$ is visible from $R$ if and only if it is visible from $C$; this problem reduces to testing whether a polygon is visible from a reflex chain, which can be determined in linear time [LC]. For the remainder of the paper, therefore, we assume that the chains $L$ and $R$ of the input polygon $P$ are weakly visible.

## 4 Straight walks

In this section we give a linear-time algorithm that determines whether $P$ is straight walkable, and, if it is, returns a straight walk. To answer questions on walkability, [IK] develop a considerable collection of definitions and theorems. As these results are necessary for our work, we begin by summarizing them.

Pictorially, we can consider two types of forbidden configurations, known as deadlocks and wedges, which are shown in Figure 4. It is clear that an instance of either configuration prevents a polygon from being straight walkable; [IK] show that the absense of any instances of these configurations ensures that a polygon is straight walkable. To generate an algorithmic approach, [IK] define functions *lo* and *hi* on the vertices of $P$. Specifically, the following definition is given for $L$, with a symmetric definition applying to $R$. (The operations *min* and *max* are defined with respect to the ordering on the chain $L$, so that *min* of a set of points is a point of the set not preceded by any other.)

**Definition 1** *[IK] for a vertex $p \in L$, we define:*

$$\begin{aligned}
hiP(p) &= \min\{q \mid q \text{ vertex of } R \text{ and } L \ni Backw(q) > p\} \\
hiS(p) &= \min\{Forw(p') \in R \mid p' \text{ vertex of } L_{>p}\} \\
hi(p) &= \min\{hiP(p), hiS(p), g\} \\
loP(p) &= \max\{q \mid q \text{ vertex of } R \text{ and } L \ni Forw(q) < p\} \\
loS(p) &= \max\{Backw(p') \in R \mid p' \text{ vertex of } L_{<p}\} \\
lo(p) &= \max\{loP(p), loS(p), s\}
\end{aligned}$$

We can think of *hi* and *lo* as functions from the vertices of $L$ to the points of $R$, and also from the vertices of $R$ to the points of $L$. It is important to note that these are monotonic functions. [IK] are able to show that a polygon $P$ is straight walkable if and only if $[lo(v), hi(v)]$ is a non-empty interval for every vertex $v$. Furthermore, if $P$ is straight walkable, then the set of possible *walk partners* for a vertex $v$ is precisely the points of $[lo(v), hi(v)]$, where two points $p \in L$, $q \in R$ are walk partners in a given straight walk if the two guards are at points $p$ and $q$ at some moment of the walk.

(We should note here a special case not considered by [IK]: their definition of *hi* and *lo* does not allow for $s$ and/or $t$ being a reflex vertex. It can be shown that if $s$ is a reflex vertex, the chains $L$ and $R$ are straight walkable if and only if $L \setminus \{s\}$ and $R$ *or* $L$ and $R \setminus \{s\}$ are straight walkable—similar observations can be made if $t$ or if $s$ and $t$ are reflex. As a result, slight modifications to our algorithm will handle the case of $s$ and/or $t$ being reflex.)

[IK] discuss the following type of search structure. Construct two doubly-linked lists, one for $L$ and one for $R$. The list for $L$ consists of all vertices of $L$, and all ray shots $Forw(q) \in L$ or $Backw(q) \in L$ where $q \in R$, with the points appearing in the list according to their order on $L$. The list for $R$ is similar. In addition, for every pair of points $p \in L$, $q \in R$ on the lists such that one point is the ray shot of the other, construct a pair of pointers between the points. As noted in [IK], it is possible to construct the points $lo(p)$, $p \in L$ in sorted order in linear time, by means of a single forward traversal of the search structure lists. Similar traversals yield $hi(p)$, $lo(q)$, and $hi(q)$. Clearly this information suffices to determine straight walkability, since $P$ is straight walkable if and only if $[lo(v), hi(v)]$ is non-empty for all vertices $v$. [IK] also present a simple, linear-time algorithm that constructs a straight walk for a walkable polygon, given $lo(p)$, $hi(p)$, $lo(q)$, and $hi(q)$, each in sorted order.

We see that to solve the straight walkability problem, it suffices to compute functions $hi$ and $lo$ on $L$ and $R$ in sorted order, and that this is done in [IK] by constructing a search structure that incorporates all of the ray shots. We will describe a search structure similar to that of [IK], but that omits some ray shots. The omitted ray shots (which we will call *dominated*) will be seen to be unneccessary when computing $hi$ and $lo$, implying that our smaller search structure is adequate for the computation of these functions.

The bottleneck step of the algorithm of [IK] is the construction of the search structure. This step requires $O(n \log n)$ time, since each of the $O(n)$ ray shots takes $O(\log n)$ time, and the $O(n)$ hit points must then be sorted. All other steps of their algorithm require $O(n)$ time. We will show how to construct our search structure in linear time. Thus, by substituting our search structure for that of [IK], we will obtain an overall linear-time algorithm.

## 4.1 Definition of dominated

We give now our definition of dominated, and show how it suggests the construction of a smaller search structure that omits dominated shots.

Consider vertices $p_1, p_2 \in L$ with $p_1 < p_2$, such that $Backw(p_1)$, $Backw(p_2) \in R$ and $Backw(p_1) > Backw(p_2)$ (see Figure 5). In other words, the shots cross. Is it possible that knowledge of $Backw(p_2)$ is necessary in computing $hi$ and $lo$? Figure 5 allows us to see pictorially why a dominated shot $Backw(p_2)$ from $L$ to $R$ can be ignored. An internal point on the segment $\overline{p_1 Succ(p_1)}$ is visible from no point of $R$ before $Backw(p_1)$, so the shot $Backw(p_1) \in R$ is saying, in effect, "The guard on $R$ must reach $Backw(p_1)$ by the time the guard on $L$ reaches $p_1$." It is easily seen that satisfying the condition imposed by $p_1$ implies that the one imposed by $p_2$ is satisfied (while the converse is not true).

Formally, we say that a shot $Backw(p_2) \in R$ from a vertex $p_2 \in L$ is *dominated* if there exists a vertex $p_1 \in L_{<p_2}$ such that $Backw(p_1) \in R$ and $Backw(p_1) > Backw(p_2)$. In an analogous manner we define dominated for shots of the types $Forw(p)$, $Backw(q)$, and $Forw(q)$. Any shot from $L$ that hits $R$ or from $R$ that hits $L$ that is not dominated is called *non-dominated.*

For each family of non-dominated shots, we have a *non-crossing property.* For example, if $Backw(p_1)$ and $Backw(p_2)$, $p_1, p_2 \in L$, are non-dominated shots, then the segments $\overline{p_1 Backw(p_1)}$ and $\overline{p_2 Backw(p_2)}$ do not cross. This means that if the origins of the non-dominated backward shots from $L$ are, in sorted order, $p_1, \ldots, p_k$, then the hit points $Backw(p_1), \ldots, Backw(p_k)$ are sorted on $R$.

The search structure that we wish to build is identical to that of [IK], except that we include only non-dominated shots as opposed to all shots. The above observation on dominated shots implies that we can compute $hi$ and $lo$ over $L$ and $R$ from our search structure in linear time, by means of a constant number of passes over the structure. The search structure can be constructed easily once we have computed a sorted list of each of the four types of non-dominated shots. Consequently, the remainder of the section will consist of a description of our linear-time method for constructing non-dominated shots in sorted order.

## 4.2 The search structure

We now give a procedure that constructs a sorted list of all non-dominated backward ray shots from $L$. The procedures for the other types of ray shots are similar. We begin with a preliminary stage, in which we mark all reflex vertices $p \in L$ such that $Backw(p) \in R$. By Lemma 3, this can be done in linear time, since each vertex can compute its cone in constant time. All non-dominated backward shots must emanate from these vertices. We will refer to these vertices as *shooting vertices.*

The basic scheme is to traverse $L$ and $R$ simultaneously from $s$ to $t$. When we encounter a shooting vertex $p \in L$, we will determine if $p$ is non-dominated, and if it is we will compute the hit point $Backw(p)$. If we have computed all non-dominated shots up to a shooting vertex $p$, then, by the non-crossing property and the definition of non-dominated, $p$ is non-dominated if and only if its shot does not cross the previous non-dominated shot. We let $p_1, \ldots, p_k$ denote the non-dominated shooting vertices of $L$, listed in the order that they appear on $L$, and we let $q_1, \ldots, q_k$ denote the corresponding hit points (i.e. $q_i = Backw(p_i)$).

The algorithm is inductive. The algorithm uses two sub-procedures, **Search** and **Bridge**, which will be described in detail below. We call procedure **Search** if we know that the shot from $c$ is non-dominated; the input is a point $q \in R$ that precedes the hit point $Backw(c)$, and the output is $Backw(q)$. When we do not know whether the shot from $c$ is dominated, procedure **Bridge** answers our question. We state our inductive hypothesis.

> Inductive Hypothesis: At the end of step $i-1$, we have traversed $L$ up to point $a$ and $R$ up to point $b$. We have computed the non-dominated shots $p_1, \ldots, p_{i-1}$ that precede $a$, and their hit points $q_1, \ldots, q_{i-1}$, where $q_{i-1} \leq b$. The points $a$ and $b$ are co-visible, and we call $\overline{ab}$ the *cutting chord*. The next shooting vertex is non-dominated if and only if its shot does not cross $\overline{ab}$.

The basis step (step 0) consists of setting $a, b \leftarrow s$.

We begin step $i$ by traversing $L$ from $a$ to the next shooting vertex, $c$. We must determine if $c$ is dominated or not. If $c$ is dominated we traverse until the next shooting vertex, but if $c$ is non-dominated we must compute the hit point $Backw(c)$. Let $\vec{r} = \vec{r}(Succ(c), c)$, and $\vec{\ell} = \vec{\ell}(Succ(c), c)$. We separate our analysis into three cases:

1. $\vec{r} \cap \overline{ab} = \emptyset$,
2. $\vec{r} \cap \overline{ab} \neq \emptyset$, and $b$ lies on the right of $\vec{\ell}$,
3. $\vec{r} \cap \overline{ab} \neq \emptyset$, and $b$ lies on the left of $\vec{\ell}$.

Case (1): The chord $\overline{ab}$ partitions $P$ into two subpolygons; let $P^s_{ab}$ be the subpolygon containing $s$, and $P^t_{ab}$ the one with $t$. Since $\overline{ab}$ is the cutting chord, $c$ is dominated if and only if its hit point precedes $b$. This is not possible in case (1), since $c \in P^t_{ab}$ and $R_{<b} \subset P^s_{ab}$; therefore $c$ is non-dominated in case (1), and we call procedure **Search($b$)**.

Case (2): In order for the shot from $c$ to be dominated, it must cross $\overline{ab}$. Since the shot starts in $P^t_{ab}$, as it crosses $\overline{ab}$ it must have $a$ on its right and $b$ on its left. In case (2), therefore, the shot is non-dominated, and we call **Search($b$)**.

Case (3): Here, the shot from $c$ is fired towards the chord $\overline{ab}$, but we cannot be sure if it hits $R_{<b}$ or $R_{>b}$ (we know it hits $R$ because $c$ is a shooting vertex). If $C_s(c) > C_t(b)$, then the shot is non-dominated (see Figure 6). This can be seen as follows. Since the hit point $Backw(c) \in R$ is visible with $c$, and $c$ and $Backw(c)$ are not both on $C$ (by the non-degeneracy assumption), we know by Lemma 4 that $C_s(c) \leq C_t(Backw(c))$; since $C_s(c) > C_t(b)$, we have $C_t(b) < C_t(Backw(c))$, which implies that $b < Backw(c)$. Since the shot is non-dominated we call **Search($b$)**.

If $C_s(c) \leq C_t(b)$, then the shot may be either dominated or non-dominated (see Figure 7). It is therefore appropriate to call the procedure **Bridge($c$,$b$)** to determine if the hit point precedes or succeeds $b$.

We now describe the sub-procedures.

Procedure **Search($q$)**

This procedure takes as input a point $q \in R_{\geq b}$ such that the hit point of $c$ succeeds $q$. It returns the hit point $Backw(c)$. Specifically, the algorithm traverses $R$ from $q$ until reaching a vertex $d$ such that

> $d$ lies on the left side of $\vec{\ell}$ and $Pred(d)$ lies on the right.

It is clear that no point preceding $Pred(d)$ can be the hit point, but $d^* = \vec{\ell} \cap \overline{Pred(d), d}$ might be. We test $c$ and $d^*$ for visibility in constant time (Lemma 5). If the points are visible, then we set $p_i, a \leftarrow c$ and $q_i, b \leftarrow d^*$, and increment $i$. If they are not visible, then we know that the hit point succeeds $d$, so we call **Search($d$)**.

End **Search.**

Procedure **Bridge($c$,$b$)**

This procedure accepts as input $c$ and $b$, where $b$ lies on the left side of $\vec{\ell}$. It can be shown that $SP(c, b)$ is convex and has only one bridge whenever **Bridge** is called (these conditions that all vertices of $SP(c, b)$ as we traverse from $c$ to $b$ are left turns). The procedure either produces the bridge of $SP(c, b)$, or answers that $SP(c, b)$ intersects $\vec{\ell}$; in the former case the shot is dominated while in the latter it is non-dominated. The procedure is described in an appendix.

End **Bridge.**

We summarize the algorithm and the argument for its correctness. We begin a step with a non-dominated chord $\overline{ab}$, and a shooting vertex $c$ which may be the next non-dominated shot. Several cases imply that $c$ is non-dominated, and we respond by calling procedure **Search**. This procedure outputs the next non-dominated shot. In the case where we are not sure whether the shot is dominated or not, we call **Bridge**, which calls **Search** if the shot is dominated. In all cases, the inductive hypothesis is re-established.

The entire algorithm runs in linear time. Each shooting vertex requires that we call one or two procedures. The work performed finding $c$ and executing the two procedures consists of traversing $L$, $R$, and portions of $SPT(t)$. However, the traversals of $L$ and $R$ progess monotonically from $s$ to $t$, and thus require only $O(n)$ time. It can be shown that all work spent on $SPT(t)$ is $O(n)$. The result is that our algorithm constructs all non-dominated backwards shots from $L$ in linear time. As described above, this is sufficient to establish our linear-time constructive algorithm for determining straight walkability.

# 5 Conclusion

This paper has taken the two-guard problem as introduced by [IK] and produced optimal-time algorithms for it. Both the straight walk and counter walk problems were originally solved in $O(n \log n)$ by [IK], and we have given $\Theta(n)$ algorithms for both.

It is interesting to note how the different time complexities arise in the two pairs of algorithms. The key to solving a walk problem is determining ray shots from reflex vertices, since these shots exactly form the restrictions on the guards' movement. [IK] respond to this need by immediately computing ray shots for all reflex vertices, and sorting the hit points, thereby locking themselves into an $O(n \log n)$ time complexity from the start. We observe that a certain class of ray shots are not essential, and that the remaining ray shots exhibit a special structure; in the straight walk case this structure is the non-crossing property, and in the counter walk case it is the crossing property. It is perhaps not surprising that guards who are required to move monotonically on $L$ and $R$ have their motion governed by chords, the essential ones which also move monotonically.

We state two open questions. The first is whether the linear time-bound can be obtained without as much use of previous results from the literature, especially triangulation and shortest path tree algorithms. The algorithms of this paper depend upon the shortest path tree, but the ideas on non-dominated and non-c-dominated shots developed here might lead in other directions as well.

A second question concerns the general walk problem, for which [IK] give an $O(n \log n + k)$ algorithm, where $k$ is the size of the instruction set of the minimum distance walk. Since $k$ could be as small as $O(n)$, their algorithm is not optimal in the output-sensitive sense, and we ask whether there exists an $O(k)$ algorithm.

# References

[AT] D. Avis and G.T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge," *IEEE Transactions on Computers*, **30** (1981), pp. 910-914.

[Cha] B. Chazelle, "Triangulating a simple polygon in linear time," *Discrete and Computational Geometry*, **6** (1991), pp. 485-524.

[CN] W.-P. Chin and S. Ntafos, "Optimum watchman routes," *Proc. 2nd ACM Symp. on Computational Geometry*, 1986, pp. 24-33.

[Chv] V. Chvatal, "A combinatorial theorem in plane geometry," *J. of Combinatorial Theory B*, **13** (1975), pp. 39-41.

[ET] H. ElGindy and G.T. Toussaint, "On geodesic properties of polygons relevant to linear time triangulation," *The Visual Computer*, **5** (1989), pp. 68-74.

[Fi] S. Fisk, "A short proof of Chvatal's watchman theorem," *J. of Combinatorial Theory B*, **24** (1978), p. 374.

[GHLST] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons," *Algorithmica*, **2** (1987), pp. 209-233.

[He] P.J. Heffernan, "Linear-time algorithms for weakly-monotone polygons," *Proc. 2nd Canadian Conference on Computational Geometry*, 1990, pp. 236-239.

[IK] C. Icking and R. Klein, "The two guards problem," *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 166-175.

[LL] D.T. Lee and A.K. Lin, "Computational complexity of art gallery problems," *IEEE Transactions on Information Theory*, **32** (1986), pp. 276-282.

[LP] D.T. Lee and F.P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks*, **14** (1984), pp. 393-410.

[LC] S.-H. Lee and K.-Y. Chwa, "Some chain visibility problems in a simple polygon," *Algorithmica*, **5** (1990), pp. 485-507.

[MW] J.S.B. Mitchell and E.L. Wynters, "Optimal motion of covisible points among obstacles in the plane," *Proc. 2nd Canadian Conference on Computational Geometry*, 1990, pp. 116-119.

[O'R] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987.

[Sh] T. Shermer, *Visibility Properties of Polygons*, Ph.D. dissertation, McGill University, June 1989.

# Appendix

This appendix gives the details of Procedure **Bridge**, and establishes its correctness and runtime.

Procedure **Bridge**($c$,$b$)

This procedure accepts as input $c$ and $b$, where $b$ lies on the left side of $\vec{\ell}$. It is necessary that $SP(c,b)$ be convex and have only one bridge. A bridge, as we recall, is an edge of a shortest path from a point of $L$ to a point of $R$ with one endpoint on $L$ and one on $R$. We will show later that these conditions are satisfied whenever **Bridge** is called (if these conditions hold, then all vertices of $SP(c,b)$ as we traverse from $c$ to $b$ are left turns). The procedure either produces the bridge of $SP(c,b)$, or answers that $SP(c,b)$ intersects $\vec{\ell}$; in the former case the shot is dominated while in the latter it is non-dominated.

We first check whether $c$ and $b$ are visible, since if they are then the shot is dominated. If not, we procede as follows.

Let $\overline{c'b'}$ represent the bridge of $SP(c,b)$, where $c' \in L$ and $b' \in R$. The chord $\overline{c'b'}$ partitions $P$ into two subpolygons, one containing $t$ and the other $s$. Because $SP(c,b)$ contains $c'$ and consists of only left turns, the path $SP(c,v)$ contains $c'$ for any vertex $v$ of the subpolygon containing $t$. Specifically, $SP(c,t)$ contains $c'$. Similarly, $SP(b,t)$ contains $b'$. If $Pred'$ and $Succ'$ are defined on the chains $SP(c,t)$ and $SP(b,t)$, then we see that $b' \in cone(d(Pred'(c'),c'),d(c',Succ'(c')))$ and $c' \in cone(d(b',Succ'(b')),d(Pred'(b'),b'))$ (Figure A1).

To find $c'$ and $b'$, we will simultaneously traverse $SP(c,t)$ and $SP(b,t)$ (recall that we can easily do this because we have stored $SPT(t)$ as a directed graph). If, at any time, the current edge of $SP(b,t)$ crosses $\vec{r}$, the procedure halts, and we know that the shot from $c$ is non-dominated. We let $b^*$ and $c^*$ denote the current vertices of $SP(b,t)$ and $SP(c,t)$, respectively, initially set to $Succ'(b)$ and $Succ'(c)$. We alternate between the steps of advancing $b^*$ to $Succ'(b^*)$ and $c^*$ to $Succ'(c^*)$. Initially at least one of the current points "shoots ahead" of its counterpart on the other chain, in the sense that

$c^*$ lies on the right of $\vec{\ell}(Pred'(b^*), b^*)$ and/or $b^*$ lies on the left of $\vec{\ell}(Pred'(c^*), c^*)$. If ever a current point "shoots behind" its counterpart, we have traversed too far with that point. For example, if by advancing $b^*$ we shoot behind $c^*$, we back up by setting $b^* \leftarrow Pred'(b^*)$. Now we have $c^*$ in the cone of $b^*$, since $c^* \in cone(d(b^*, Succ'(b^*)), d(Pred'(b^*), b^*))$. We continue to traverse forward with $c^*$. If necessary, we traverse backward with $b^*$ so that $c^*$ remains in the cone of $b^*$. We stop when $c^*$ first shoots behind $b^*$, and set $c' \leftarrow Pred'(c^*)$ and $b' \leftarrow b^*$. The procedure is symmetric if $c^*$ shoots behind $b^*$ first.

Another possible event is that either $b^*$ may encounter a right turn or $c^*$ a left turn. Since $SP(c, b)$ is convex, such events can occur only after $b'$ and $c'$, respectively. Therefore we stop traversing forward with such a pointer. For example, if $b^*$ is a left turn, we set $b^*$ to $Pred'(b^*)$. We traverse forward with $c^*$ while leaving $b^*$ fixed. If $c^*$ moves to the left of $\vec{\ell}(Pred'(b^*), b^*)$, we traverse backward with $b^*$, so that $c^*$ stays in the cone of $b^*$. We stop when $c^*$ shoots behind $b^*$, and construct the bridge as described above.

If we find that $SP(b, t)$ intersects $\vec{\ell}$, where $b''$ is the first vertex of $SP(b, t)$ on the right side of $\vec{\ell}$, then we know that the hit point succeeds $b''$; we call **Search**$(b'')$ in order to find the non-dominated hit point of $c$. Otherwise, all of $SP(b, t)$ lies left of $\vec{\ell}$, which we learn when we find that $b'$ is left of $\vec{\ell}$; this implies that the hit point precedes $b$ and that the shot is dominated. In this case, any shooting vertex on $L_{c,c'}$ has a hit point that precedes $b$, as seen in Figure A1, so we can skip over these points. Since any shot from $L_{>c'}$ cannot hit $R_{b,b'}$, we advance $b$ to $b'$. This gives a new cutting chord $\overline{c'b'}$. Thus, in the case where the shot from $c$ is dominated, we have $a \leftarrow c'$, $b \leftarrow b'$; we traverse forward on $L$ from $c'$ until finding the first shooting vertex, which becomes the new point $c$, and we test whether this new $c$ is dominated.
End **Bridge**.

We now show that procedure **Bridge** is called only when the necessary conditions hold.

**Lemma 6** *If* **Bridge**$(c,b)$ *is called, the shortest path $SP(c, b)$ is convex. Furthermore, it has only one bridge.*

We briefly discuss the time-complexity of procedure **Bridge**. The traversals of $L$ and $R$ are simply a part of the monotonic traversals of those chains made by the full algorithm. However, **Bridge** also traverses portions of $SPT(t)$; we claim that the total traversal time of $SPT(t)$ is $O(n)$. First, let us consider the backtracking of $SPT(t)$ that can occur in the procedure. This happens when one of the points shoots behind its partner or encounters a non-convex turn. However, because we alternate forward steps of $b^*$ with those of $c^*$, all of the work can be charged to the chain which is not backtracked, at the expense of an extra constant factor. We claim that the charged portion of $SPT(t)$ is not traversed again. If we assume, without loss of generality, that $SP(b, b')$ is the path charged, then it suffices to show that no portion of $SP(b, b')$ is on the shortest path of any point $q \in R_{>b'}$. If such a point $q$ did have a point $q' \in R_{b,b'}$ on $SP(q, t)$, then we would have $q' \in R_{<q}$ and $t \in R_{>q}$ both on $SP(q, t)$—a slight variant of Corollary 1 implies that $q$ is not visible from $L$, since $q'$ precedes $t$ on $SP(q, t)$. Therefore only $O(n)$ time is spent traversing portions of $SPT(t)$, and the linear run-time of the algorithm is established.

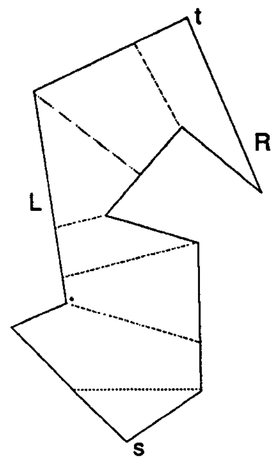

Figure 1: A straight walk

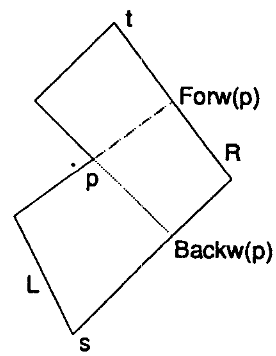

Figure 2: Forward and backward ray shots

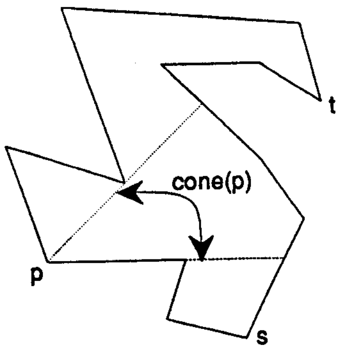


Figure 3: cone(p)

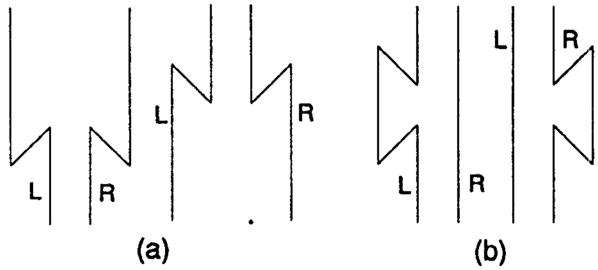


Figure 4: (a) Deadlocks, and (b) wedges

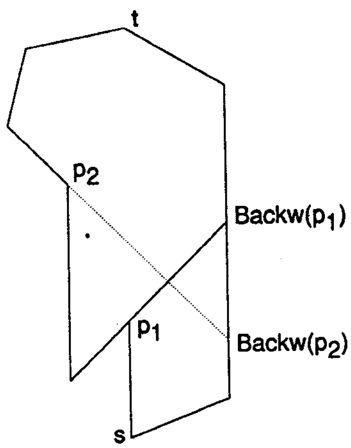


Figure 5: Definition of dominated

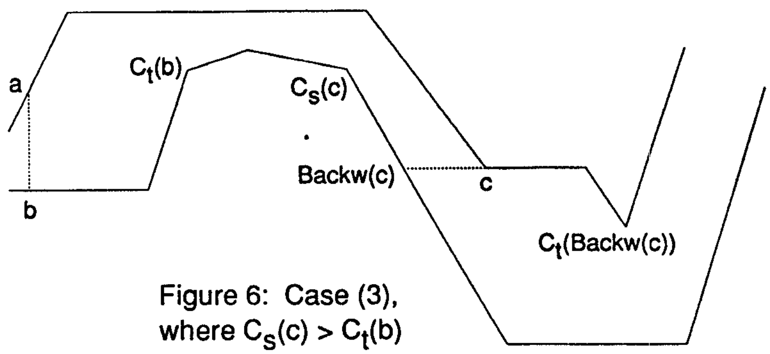


Figure 6: Case (3),
where $C_s(c) > C_t(b)$

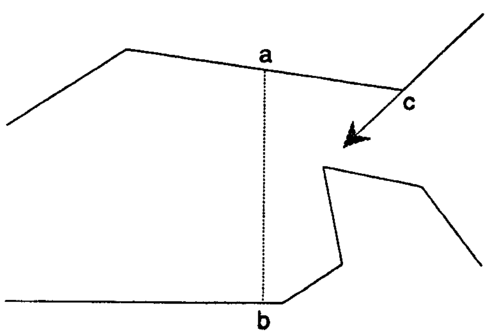


Figure 7: Case (3),
where $C_s(c) <= C_t(b)$

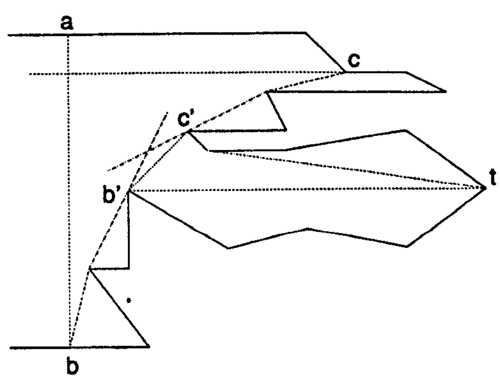


Figure A1: Procedure Bridge