International Journal of Computational Geometry & Applications

TWO-GUARD WALKABILITY OF SIMPLE POLYGONS

L. H. TSENG
Department of Applied Mathematics
Feng Chia University, Taichung 407, TAIWAN
e-mail: lhtseng@math.fcu.edu.tw

P. HEFFERNAN
The Prudential Insurance Company of America
213 Washington St., Newark, NJ 07102-2992
e-mail: heff@email.njin.net

and

D. T. LEE
Department of Electrical and Computer Engineering
Northwestern University, Evanston, IL 60208
e-mail: dtlee@ece.nwu.edu

Received 7 July 1992
Revised 29 July 1996
Communicated by J. S. B. Mitchell

ABSTRACT

A pair of points s and g on the boundary of a simple polygon P admits a walk if two guards can simultaneously walk along the two boundary chains of P from s to g such that they are always visible to each other. The walk is a counter-walk if one guard moves from s to g while the other moves from g to s in the same direction along the boundary and they are always visible to each other. The (counter-)walk is straight if no backtracking is necessary during the (counter-)walk. In this paper, we show that, given a polygon with n vertices, to test if there exists (s, g) that admits a (straight) (counter-)walk can be solved in time $O(n \log n)$ and in linear space. Also we compute all (s, g)'s that admit a (straight) walk in $O(n \log n)$ time and all vertex pairs that admit a (straight) counter-walk in $O(n \log n + m)$, where m is $O(n^2)$.

Keywords: Visibility, watchman routes, motion planning, simple polygon, circular-arc graph.

1. Introduction

Given a simple polygon P, a pair of distinct points (s, g) on the boundary of P is said to admit a *walk* if two points (called the *guards*) can be moved from s to g, each on one of the two boundary chains, and the line segment connecting the two points is fully contained in P all the time. (s, g) is said to admit a *counter-walk* if one guard moves from s to g while the other moves from g to s say, in counterclockwise direction, and the line segment connecting them is fully contained in P all the time. The guards are allowed to backtrack locally during the (counter-)walk. We say the (counter-)walk is *straight* if no backtracking is necessary during the (counter-)walk. A simple polygon is (*straight*) (*counter-*) *walkable* if there exists at least an (s, g) that admits a (straight) (counter-)walk.

The 2-guard walkability problem was first introduced by Icking and Klein.[6] It was shown that to test if a given (s, g) on P admits a (straight) (counter-)walk can be done in $O(n \log n)$ time and in $O(n)$ space[a], where n is the number of vertices of P. Later, a linear-time straight (counter-)walkability test algorithm was presented by Heffernan.[4] In this paper, we consider a more general *polygon (straight) (counter-)walkability test* problem, i.e., given P, to test if there exists (s, g) that admits a (straight) (counter-)walk and, output one, if it exists. We show that this problem can be solved in $O(n \log n)$ time and linear space. We also obtain solutions to the *report-all-pairs* problem: To report all the point pairs on the boundary of P that admit a (straight) walk can be solved in $O(n \log n)$ time, and to report all vertex pairs that admit a (straight) counter-walk can be done in $O(n \log n + m)$ time, where m is $O(n^2)$.

2. Preliminaries

Let P be a simple polygon with n vertices. As in common parlance, P is used to denote the union of the boundary and of the interior. Given two distinct points p and q on the boundary of P, let $CCW(p, q)$ (resp. $CW(p, q)$) denote the polygonal chain from p counterclockwise (resp. clockwise) to q, with p and q included. The two polygonal chains $CW(s, g)$ and $CCW(s, g)$ are referred to as the *left* and *right chains*, denoted as $L_{(s,g)}$ and $R_{(s,g)}$ respectively (or L and R, if (s, g) is understood). We call a point $x \in P$ *visible* from a point $y \in P$ if the line segment $\overline{xy}$ is entirely contained in P. $L_{(s,g)}$ is said to be *weakly visible* from $R_{(s,g)}$, if for each point $p \in L$ there exists a point $q \in R$ such that q is visible from p. That R is weakly visible from L is defined analogously. If $L_{(s,g)}$ and $R_{(s,g)}$ are mutually weakly visible, we say (s, g) is *LR-visible*. Following the definition in Ref. [7] we say a simple polygon is *weakly star-shaped* if there exists a segment e (called *weakly visible segment*) inside the polygon such that each point of the polygon is visible from some point on e.

For a reflex vertex v adjacent to another vertex u, let v' be the first point on the boundary of P hit by the half line emanating from v in the same direction as $\overrightarrow{uv}$. We call $\overline{vv'}$ a *chord* of P. A chord $\overline{vv'}$ divides P into two parts and the one containing u, v and v' is the *component* of $\overline{vv'}$, denoted as $Comp(\overline{vv'})$. See Figure 1. By convention, when we write $\overline{vv'}$ as a chord, v is the reflex vertex and v' is the hit point. We distinguish two types of chords: A chord $\overline{vv'}$ with its component containing $CCW(v, v')$ is called a *ccw-chord*, and a *cw-chord* is defined analogously. For example, $\overline{vv'}$ in Figure 1 is a cw-chord. For convenience, it is assumed that P is in a general position in the sense that all chords have distinct hit points and none

[a]We note that, in the presentation of their algorithm, s and g cannot be on reflex vertices. As we'll show later, the restriction can be removed by modifying their algorithm without increasing the time and space complexity.

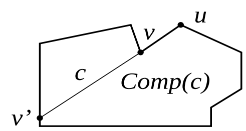

Fig. 1. A chord c and its component.

of the hit points coincides with any polygon vertex.

Let s^+ (resp. s^-) be the point on the boundary of P immediately counterclockwise (resp. clockwise) from s. g^+ and g^- are defined analogously. We say that there is a *deadlock*, denoted $DK(v_1, v_2)$, for (s, g), if there exist two reflex vertices v_1 and v_2 with their associated ccw-chords $\overline{v_1 v_1'}$, $\overline{v_2 v_2''}$ and cw-chords $\overline{v_1 v_1''}$, $\overline{v_2 v_2'}$, such that case (1) or (2) below applies (see Figure 2).

(1) $v_1' \in CCW(g, v_2) \subseteq L$, $v_2' \in CCW(v_1, g) \subseteq R$, and

- (1a) s is neither v_1 nor v_2, or
- (1b1) $s = v_1$ and (s^+, g) has a deadlock of case (1)(1a), or
- (1b2) $s = v_1$ and (s^+, g) is not LR-visible, or
- (1c1) $s = v_2$ and (s^-, g) has a deadlock of case (1)(1a), or
- (1c2) $s = v_2$ and (s^-, g) is not LR-visible.

(2) $v_1' \in CCW(s, v_2) \subseteq R$, $v_2' \in CCW(v_1, s) \subseteq L$, and

- (2a) g is neither v_1 nor v_2, or
- (2b1) $g = v_1$ and (s, g^+) has deadlock of case (2)(2a), or
- (2b2) $g = v_1$ and (s, g^+) is not LR-visible, or
- (2c1) $g = v_2$ and (s, g^-) has deadlock of case (2)(2a), or
- (2c2) $g = v_2$ and (s, g^-) is not LR-visible.

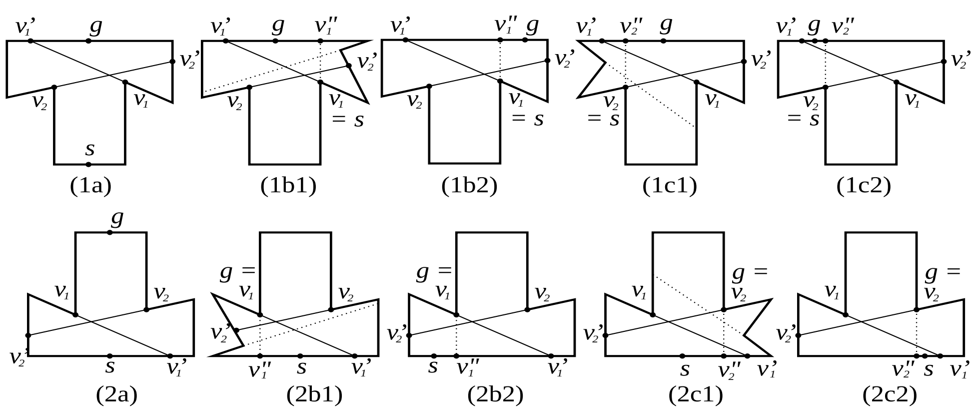

Fig. 2. Cases of deadlocks.

If case (1) is true, the deadlock is called an *s-deadlock* and if case (2) is true, the deadlock is called a *g-deadlock*. We notice that conditions (1b1) and (1c1) are actually equivalent, as can be seen by relabeling the vertices, and so are (2b1) and (2c1). Also, in condition (1b1), the deadlock for (s^+, g) is formed by the cw-chord $\overline{v_1 v_1''}$ and some other ccw-chord. And, in condition (1b2), the non-LR-visibility of (s^+, g) is due to the cw-chord $\overline{v_1 v_1''}$, because $s^- \in L_{(s^+,g)}$ is not visible from any point on $R_{(s^+,g)}$. Similar observations can be said about other conditions.

For two distinct reflex vertices v_1 and v_2, $CCW(v_1, v_2)$ defines a *wedge*, denoted $W(v_1, v_2)$, if (1) ccw-chord $\overline{v_1 v_1'}$ and cw-chord $\overline{v_2 v_2'}$ intersect with each other, and (2) $v_1 \in Comp(\overline{v_2 v_2'})$ and $v_2 \in Comp(\overline{v_1 v_1'})$. We say $W(v_1, v_2)$ is a *left wedge* of (s, g) if $v_1, v_2 \in L$, $v_1', v_2' \in R$, and

(1) $s \neq v_2$ and $g \neq v_1$, or

(2) $s = v_2$ and ((s^-, g) has an s-deadlock or is not LR-visible).

Similarly, $W(v_1, v_2)$ is a *right wedge* of (s, g) if $v_1, v_2 \in R$, $v_1', v_2' \in L$, and

(1) $s \neq v_1$ and $g \neq v_2$, or

(2) $s = v_1$ and ((s^+, g) has an s-deadlock or is not LR-visible).

See Figure 3. We notice that in condition (2) of the left wedge, if (s^-, g) has an s-deadlock, the deadlock is formed by the ccw-chord $\overline{v_2 v_2''}$ and some other cw-chord, and if (s^-, g) is not LR-visible, it is due to the ccw-chord $\overline{v_2 v_2''}$, for $s^+ \in R_{(s^-,g)}$ is not visible from any point on $L_{(s^-,g)}$. Similar observations can be said about condition (2) of the right wedge.

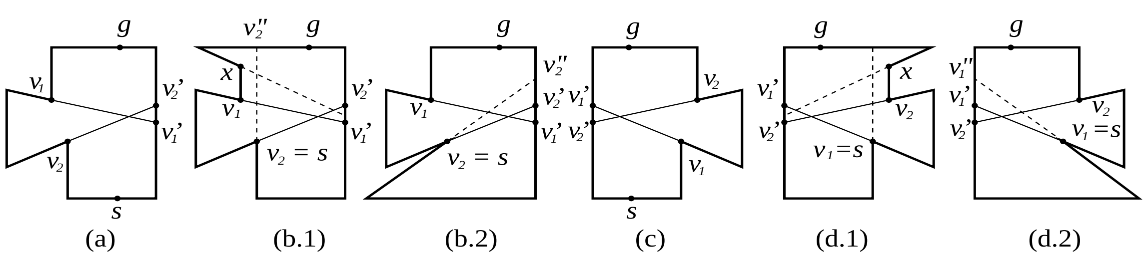

Fig. 3. (a) A case (1) left wedge. (b.1) A case (2) left wedge where (s^-, g) has s-deadlock $DK(v_2, x)$. (b.2) A case (2) left wedge where (s^-, g) is not LR-visible. (c) A case (1) right wedge. (d.1) A case (2) right wedge where (s^+, g) has s-deadlock $DK(x, v_1)$. (d.2) A case (2) right wedge where (s^+, g) is not LR-visible.

The following lemmas (Lemma 1 to Lemma 4) give the necessary and sufficient conditions for our polygon (straight) walkability test (in Sections 4 and 5), polygon (straight) counter-walkability test (in Section 6), and the all-pairs-reporting problem (in Section 7). These Lemmas are from Ref. [6] with some added conditions that were missing for the special case when s or g is on a reflex vertex. It is not difficult to see the added conditions are necessary. The sufficiency of the new set of conditions can be proved by modifying the constructive proof in Ref. [6] to show "walk instructions" can be constructed if the conditions hold. We leave the details in the Appendix A.

Lemma 1 *(s, g) admits a walk if and only if (s, g) is LR-visible and has no deadlocks.*

Lemma 2 *(s, g) admits a straight walk if and only if (s, g) is LR-visible, has no deadlocks, and has no left or right wedges.*

Lemma 3 *(s, g) admits a straight counter-walk if and only if s and g are visible, (s, g) is LR-visible, and none of the following four cases applies (see examples in Figure 4).*

(a) $\exists\, \overline{v_1 v_1'}, \overline{v_2 v_2'}$ *: (i) both of them are ccw-chords or cw-chords, (ii) $v_1, v_2' \in L$, $v_2, v_1' \in R$, (iii) $Comp(\overline{v_1 v_1'}) \cap Comp(\overline{v_2 v_2'}) = \emptyset$, and (iv) ($v_1 \neq s$ and $v_2 \neq g$) or ($v_1 \neq g$ and $v_2 \neq s$).*

(b)(b.1) $\overline{ss'}$ is a chord such that ($s' \in L$, $g \notin Comp(\overline{ss'})$, and $\exists\, q \in R$ not visible from s) or ($s' \in R$, $g \notin Comp(\overline{ss'})$, and $\exists\, q \in L$ not visible from s), or

(b.2) $\overline{gg'}$ is a chord such that ($g' \in L$, $s \notin Comp(\overline{gg'})$, and $\exists\, q \in R$ not visible from g) or ($g' \in R$, $s \notin Comp(\overline{gg'})$, and $\exists\, q \in L$ not visible from g).

(c) $\exists\, \overline{v_1 v_1'}, \overline{v_2 v_2'}$ *: (i) one is a cw-chord and the other is a ccw-chord, (ii) either ($v_1, v_2 \in L$, $v_1', v_2' \in R$) or ($v_1, v_2 \in R$, $v_1', v_2' \in L$), and (iii) $Comp(\overline{v_1 v_1'}) \cap Comp(\overline{v_2 v_2'}) = \emptyset$.*

(d)(d.1) $\overline{ss'}$ is a chord such that ($s' \in L$, $g \notin Comp(\overline{ss'})$, and $\exists\, q \in L$ not visible from s) or ($s' \in R$, $g \notin Comp(\overline{ss'})$, and $\exists\, q \in R$ not visible from s), or

(d.2) $\overline{gg'}$ is a chord such that ($g' \in L$, $s \notin Comp(\overline{gg'})$, and $\exists\, q \in L$ not visible from g) or ($g' \in R$, $s \notin Comp(\overline{gg'})$, and $\exists\, q \in R$ not visible from g).

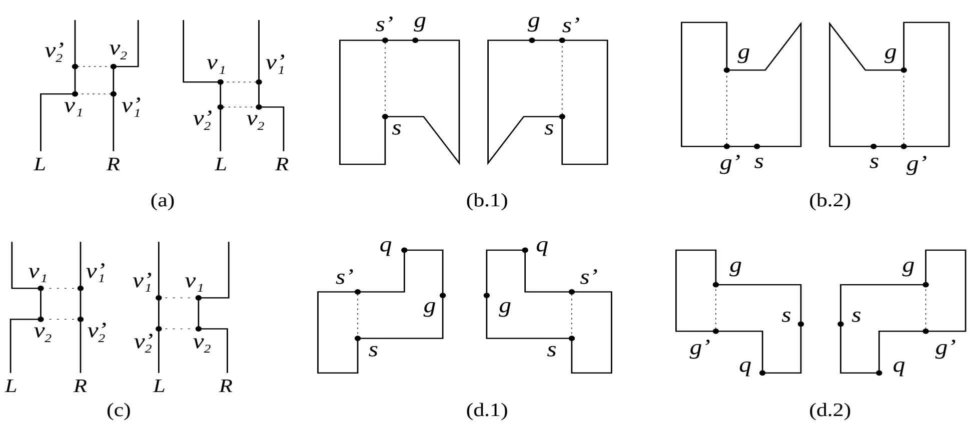

Fig. 4. Cases where no (straight) counter-walk exists.

Lemma 4 *(s, g) admits a counter-walk if and only if s and g are visible, (s, g) is LR-visible and neither case (a) nor (b) in Lemma 3 applies (see Figure 4 (a) and (b)).*

The following lemma gives an equivalent LR-visibility condition for a given (s, g). For its proof, refer to a similar lemma in Ref. [6].

Lemma 5 *Given s and g on the boundary of P, (s, g) is LR-visible if and only if there is no chord c such that $P - Comp(c)$ contains both s and g (i.e., if and only if $Comp(c)$ contains s or g, or both.)*

Some chords of P are useless in the sense that we can delete them without affecting the correctness of Lemma 5. To be precise, we say that a chord c is *redundant* if there is another chord c' such that $Comp(c')$ is contained in $Comp(c)$. The redundancy of chords was studied in Ref. [9], in which a linear time algorithm was given for eliminating all the redundant chords in P. Since it follows from Lemma 5 that LR-visibility is determined by non-redundant chords, we have

Corollary 1 *(s, g) is LR-visible if and only if, for each non-redundant chord c, $Comp(c)$ contains s or g, or both.*

In the next section, we introduce the *2-cut circular arcs problem* which, as will be seen later, is related to our problems.

3. 2-Cut Circular Arcs Problem

The 2-cut circular arcs problem is as follows: Given a set S of $n \geq 2$ non-empty proper[b] closed arcs $S = \{a_1, a_2, \ldots, a_n\}$ on a unit circle $\mathcal{K}$, determine if there exists a pair of two cut points (c_1, c_2), such that each arc contains at least one cut point, and if so, enumerate all such pairs[c], referred to as *2-cuttable* pairs. We'll show in the subsections that, if the endpoints of the arcs are sorted, computing one 2-cuttable pair can be solved in linear time, and computing all 2-cuttable pairs can be done in $O(n \log n)$ time. First, we give some definitions.

For convenience, for any two points e, f on $\mathcal{K}$, we use $CCW(e, f)$ to denote the portion on $\mathcal{K}$ from e (inclusive) counterclockwise to f (inclusive), if $e \neq f$. If $e = f$, we let $CCW(e, e) = CCW(f, f)$ be $\mathcal{K}$. $CCW(e, f)$ is called an *interval* (from beginning point e to ending point f). Each arc $a_i \in S$ is represented as $a_i = CCW(b_i, e_i)$. It is assumed, throughout this section, that the endpoints of arcs in S are sorted in the counterclockwise direction and arranged in a circular list. Define a *cut interval* C to be a non-empty interval $CCW(b, e)$ on $\mathcal{K}$ such that $b, e \in \{b_i | i = 1, \ldots, n\} \cup \{e_i | i = 1, \ldots, n\}$, $b \neq e$, and for all $c, d \in C - \{b, e\}$, the set of arcs intersected by the cut at c is the same as the set of arcs intersected by d. It can be seen that the number m of cut intervals is at most $2n$. We say a pair (E, F) of intervals is 2-cuttable if any (c, d) on (E, F)[d] is 2-cuttable.

If (c_1, c_2) is a 2-cuttable pair for S and (c_1, c_2') is not, where $c_2' \in CCW(c_2, c_1)$, we denote NEXT(c_1, c_2') as a 2-cuttable pair (d_1, d_2) such that $d_1 \in CCW(c_1, c_2)$, $d_2 \in CCW(c_2', c_1)$, and any (c, d) on $(CCW(c_1, d_1) - \{d_1\}, CCW(c_2', d_2)) \cup (CCW(c_1, d_1), CCW(c_2', d_2) - \{d_2\})$ is not a 2-cuttable pair. Similarly, if (c_1, c_2) is a 2-cuttable pair and (c_1', c_2) is not, where $c_1' \in CCW(c_1, c_2)$, we denote NEXT(c_1', c_2) as a 2-

[b] Arcs are *proper* if there is no arc fully containing another.

[c] Strictly speaking we find pairs of intervals $((p, q), (r, s))$ such that any pair of cut points (c_1, c_2) where $c_1 \in (p, q), c_2 \in (r, s)$ is a 2-cuttable pair.

[d] "(c, d) on (E, F)" means $c \in E$ and $d \in F$, where $c, d \in \mathcal{K}$ and E, F are two intervals.

cuttable pair (d_1, d_2) such that $d_1 \in CCW(c_1', c_2)$, $d_2 \in CCW(c_2, c_1)$, and any (c, d) on $(CCW(c_1', d_1) - \{d_1\}, CCW(c_2, d_2)) \cup (CCW(c_1', d_1), CCW(c_2, d_2) - \{d_2\})$ is not a 2-cuttable pair. NEXT is a well defined function from the following argument. For a 2-cuttable pair (c_1, c_2), if one cut, say c_2, is moved counterclockwise to some position c_2' on $CCW(c_2, c_1)$, such that it leaves some arcs not intersected by the other cut (c_1), then c_1 must be moved in the same direction (counterclockwise) to intersect those unintersected arcs in order that both cuts can possibly intersect again all the arcs. In such a way, one move of a cut may cause the other cut to be moved in the same direction, and this is repeated until no arcs are unintersected. The moves will stop finally because the pair never passes the original 2-cuttable pair, i.e., c_1 is not past the original c_2 and c_2 is not past the original c_1 (NEXT(c_1, c_2') could be (c_1, c_2)). The above argument also implies that NEXT$(p, q) = (p', q')$ can be computed in $O(k)$ time, where k is the number of endpoints of arcs on the intervals passed by (p, q), i.e., $CCW(p, p')$ and $CCW(q, q')$.

Lemma 6 *Let NEXT$(p, q) = (p', q')$.*

(a) If $p \neq p'$, then any $(a, b) \in (CCW(p, p') - \{p'\}, CCW(q', a))$ is not 2-cuttable.

(b) If $q \neq q'$, then any $(a, b) \in (CCW(p', b), CCW(q, q') - \{q'\})$ is not 2-cuttable.

Proof. If $p \neq p'$, there must be an arc, say a_1, with beginning point p' and not intersected by q', because otherwise $p = p'$. (a, b) in the lemma is not 2-cuttable because neither a nor b intersects a_1. Thus condition (a) is true. Condition (b) can be similarly proved. □

3.1. Computing a 2-Cuttable Pair

The problem can be solved by using, with some modification, an algorithm by Lee, et al.[8] which solves the following more general *minimum cuts for circular arcs* problem: given a set of n non-empty open arcs on a unit circle, find a set of cut points such that each arc contains at least one cut point and the number of cut points is minimized. The time and space complexity of their algorithm is $O(n)$ if the endpoints of arcs are sorted. Thus we have

Lemma 7 *To determine if there exists a 2-cuttable pair for a given set S of proper arcs and, output one, if it exists, can be done in linear time, after the endpoints of arcs in S are sorted.*

Another way to compute a 2-cuttable pair is using the procedure in the next subsection, which computes not only one but all the 2-cuttable pairs.

3.2. Computing All 2-Cuttable Pairs

The procedure below determines if there exists a 2-cuttable pair for S, and if so, computes all 2-cuttable pairs. The output consists of $O(n)$ 2-cuttable pairs of intervals.

PROCEDURE *2-CUTTABLE_(E_i, F_i)'s*

(1) [B_1, B_2 and A_3]

(a) Find two disjoint arcs a_1 and a_2;
if no such two arcs **then**
Let e and f respectively be the beginning and ending points of any arc d;
Let ff be the beginning point of the first arc not intersected by e clockwise from f, or let ff be e if that beginning point does not exist;
$B_1 \longleftarrow CCW(e,e)$; $B_2 \longleftarrow CCW(f,f)$; $A_3 \longleftarrow S - \{d\}$;
Go to (2);
else
$B_1 \longleftarrow a_1$; $B_2 \longleftarrow a_2$; $A_1 \longleftarrow \{a_1\}$; $A_2 \longleftarrow \{a_2\}$;
Divide the arcs in $A_3 = S - \{a_1, a_2\}$ into four subsets:
A'_0: arcs that intersect neither B_1 nor B_2,
A'_1: arcs that intersect B_1 but not B_2,
A'_2: arcs that intersect B_2 but not B_1,
A'_3: arcs that intersect both B_1 and B_2;

(b) **while** $A'_0 \cup A'_1 \cup A'_2 \neq \emptyset$ **do**
if $A'_0 \neq \emptyset$ **then** Report there is no 2-cuttable pair for S and Stop;
else
$B'_1 \longleftarrow B_1 \cap (\bigcap_{a \in A'_1} a)$; $B'_2 \longleftarrow B_2 \cap (\bigcap_{a \in A'_2} a)$;
if either B'_1 or B'_2 is empty **then** Report there is no 2-cuttable pair for S and Stop;
else
$B_1 \longleftarrow B'_1$; $B_2 \longleftarrow B'_2$; $A_1 \longleftarrow A_1 \cup A'_1$; $A_2 \longleftarrow A_2 \cup A'_2$;
Divide the arcs in A'_3 into four subsets $A''_0, A''_1, A''_2, A''_3$ similar to above;
$A'_0 \longleftarrow A''_0$; $A'_1 \longleftarrow A''_1$; $A'_2 \longleftarrow A''_2$; $A'_3 \longleftarrow A''_3$;

(c) $A_3 \longleftarrow A'_3$;
Let e and f be the beginning points of B_1 and B_2 respectively; $ff \longleftarrow f$;

(2) [2-cuttable (E_i, F_i)'s in (B_1, B_2)]
$i \longleftarrow 1$; $u \longleftarrow e$;
Scan B_1 counterclockwise from e to the ending point of B_1:
while not reaching the ending point of B_1 **do**

(a) **if** leaving an arc a at (its ending point) v **then**
Output $(E_i, F_i) = (CCW(u,v), CCW(ff, ff'))$, where ff' is the ending point of the first arc not intersected by u scanned counterclockwise from ff, or ff' is ff if that ending point does not exist;
if a is not intersected by ff **then**
$ff \longleftarrow$ the beginning point of a;
if $ff \notin F_i$ **then** $(u, ff) \longleftarrow \mathrm{NEXT}(u, ff)$; **else** $u \longleftarrow v$;
else $u \longleftarrow v$;
$i \longleftarrow i + 1$;

(b) **else if** entering an arc a at (its beginning point) v **then**
Output $(E_i, F_i) = (CCW(u,v), CCW(ff, ff'))$, where ff' is the ending point of a;
$u \longleftarrow v$; $i \longleftarrow i + 1$;

Output $(E_i, F_i) = (CCW(u$, the ending point of B_1), $CCW(ff$, the ending point of B_2));

END of *2-CUTTABLE_(E_i, F_i)'s*

There are two steps in the procedure. The goal of step (1) is to determine if there exists a 2-cuttable pair by computing two intervals B_1 and B_2 on $\mathcal{K}$ such that for any 2-cuttable pair, one cut must be in B_1 and the other in B_2. Also, we want the two intervals to be as small as possible. Our idea is borrowed from Ref. [7] for detecting if a polygon is weakly star-shaped and, if so, computing one weakly visible segment. In Ref. [7], two regions P_1 and P_2 in P are computed such that any weakly visible segment must have one endpoint in P_1 and the other in P_2. In fact, the arcs in our procedure correspond to the components in theirs, and B_1, B_2 correspond to P_1, P_2 respectively. Now, we explain how to compute B_1 and B_2. First, we determine if there exist two disjoint arcs a_1 and a_2. This can be done by computing, for each arc $a_i = (b_i, e_i)$, an arc denoted $\mathrm{nxt}(a_i) = (b_j, e_j)$, such that b_j is the first beginning point encountered during counterclockwise scan of $\mathcal{K}$ starting from e_i (exclusive) to b_i (inclusive). (This can be done in total $O(n)$ time for all arcs.) If there exists an arc a_i such that a_i and $\mathrm{nxt}(a_i)$ are disjoint, then there are two disjoint arcs. Otherwise, any two arcs are not disjoint. The correctness comes from the fact that if any arc a_i is disjoint from some arcs, $\mathrm{nxt}(a_i)$ must be one of these arcs.

If we can not find two disjoint arcs, each arc must intersect all the other arcs and hence, two endpoints of the arc form a 2-cuttable pair. In this case, we pick an arbitrary arc with beginning point e and ending point f; let B_1 and B_2 both be $\mathcal{K}$ but represented as $CCW(e, e)$ and $CCW(f, f)$ respectively; let $A_3 (\neq \emptyset)$ be the set of remaining arcs; and we finish step (1). Otherwise, let the two disjoint arcs be our initial B_1 and B_2. It is clear that for any 2-cuttable pair, one cut must be in B_1 and the other in B_2. Now, if there is some arc c not intersected with a_1 or a_2, then there is no 2-cuttable pair for S because of mutually disjoint a_1, a_2, and c. Otherwise, we try to make B_1 and B_2 smaller by intersecting B_1 with the arcs that are not intersected with B_2 to get B_1', and intersecting B_2 with the arcs that are not intersected with B_1 to get B_2'. If either B_1' or B_2' is empty, then there is no 2-cuttable pair for S because there exist three mutually disjoint arcs. Otherwise, let B_1' and B_2' be our new B_1 and B_2 respectively. Again, it is clear that for any 2-cuttable pair, one cut must be in B_1 and the other in B_2. With the new smaller B_1, B_2 and the unprocessed arcs (i.e., the arcs intersecting both old B_1 and old B_2), we repeat this process to make B_1, B_2 possibly even smaller, in a loop in step (1) (b). Actually, in step (1) (b) the following properties are maintained in each iteration:

($\mathcal{P}_1$) B_1 and B_2 are disjoint.

($\mathcal{P}_2$) Each arc in A_0' is disjoint from both B_1 and B_2.

($\mathcal{P}_3$) Each arc in A_1' is disjoint from B_2 and has one endpoint in B_1. Similarly, each arc in A_2' is disjoint from B_1 and has one endpoint in B_2.

($\mathcal{P}_4$) Each arc in A'_3 has one endpoint in B_1 and the other in B_2.

($\mathcal{P}_5$) If $B'_1 = \emptyset$, then there exist two disjoint arcs in A'_1; if $B'_2 = \emptyset$, then there exist two disjoint arcs in A'_2; if either B'_1 or B'_2 is empty, then there is no 2-cuttable pair for S.

($\mathcal{P}_6$) Each arc in A_1 is disjoint from B_2 and each arc in A_2 is disjoint from B_1.

($\mathcal{P}_7$) For any 2-cuttable pair, one cut must be in B_1 and the other in B_2.

Finally, we can either decide there is no 2-cuttable pair for S or have computed B_1, B_2 and A_3 such that (i) B_1 is the intersection of the arcs in A_1, (ii) B_2 is the intersection of the arcs in A_2, (iii) $A_1 \cup A_2 \cup A_3 = S$, where A_3 is the set of arcs that intersect both B_1 and B_2, and A_1, A_2, A_3 are mutually disjoint sets, and (iv) for any 2-cuttable pair, one cut must be in B_1 and the other in B_2. Thus the goal of step (1) is achieved.

We now describe how to implement step (1) in time complexity $O(n \log n)$. The technique described below is again borrowed from Ref. [7]. We consider one iteration in step (1) (b). Currently, procedure has computed $B_1 = CCW(b_1, e_1)$, $B_2 = CCW(b_2, e_2)$, and four subsets: A'_0, A'_1, A'_2,and A'_3. It will output two modified B_1, B_2, and also four subsets of A'_3: $A''_0, A''_1, A''_2, A''_3$. We partition A'_3 into two subsets $A_{3,1}$ and $A'_{3,1}$, where $A_{3,1}$ is the set of arcs intersected by e_1 and $A'_{3,1}$ is the set of arcs intersected by b_1. Arcs in $A_{3,1}$ (resp. $A'_{3,1}$) are ordered according to the positions of their endpoints on B_1 and represented by a binary tree T_1 (resp. T'_1) such that searching and deleting can be done in $O(\log n)$ time per operation. Similarly, arcs in A'_3 is partitioned into $A_{3,2}$ and $A'_{3,2}$ by their intersection with e_2 or b_2, and, according to the order of their endpoints on B_2, represented by two binary trees T_2 and T'_2. Assuming $A'_0 = \emptyset$ and $A'_1 \cup A'_2 \neq \emptyset$, two phases are followed.

In the first phase, we intersect all chords of A'_1 with B_1. Let $A'_1 = \{a_{(1)}, a_{(2)}, \ldots, a_{(i)}, \ldots, a_{(h)}\}$. We start by intersecting $a_{(1)}$ with B_1. If the intersection $B_{(1)} = a_{(1)} \cap B_1$ is empty, then S does not have any 2-cuttable pair. Otherwise, we delete all arcs from T_1 or T'_1 that become disjoint from $B_{(1)}$ and move them from A'_3 to A''_2. This can be done by searching the endpoint of $a_{(1)}$ in T_1 or T'_1. In general, we intersect $a_{(i)}$ with $B_{(i-1)} = (\cap_{j=1}^{i-1} a_{(j)}) \cap B_1$, and if the intersection $B_{(i)} = a_{(i)} \cap B_{(i-1)}$ is empty, then S does not have any 2-cuttable pair; otherwise, we delete all arcs from T_1 or T'_1 that become disjoint from $B_{(i)}$ and move them from A'_3 to A''_2. Finally, we have $B'_1 = B_{(h)}$.

In the second phase, we intersect all chords of A'_2 with B_2. Let $A'_2 = \{a_{(1)}, a_{(2)}, \ldots, a_{(i)}, \ldots, a_{(l)}\}$. We start by intersecting $a_{(1)}$ with B_2. If the intersection $B_{(1)} = a_{(1)} \cap B_2$ is empty, then S does not have any 2-cuttable pair. Otherwise, we delete all arcs from T_2 or T'_2 that become disjoint from $B_{(1)}$. Among these deleted arcs, we move those from A''_2 to A''_0 that are also deleted from T_1 or T'_1 in the first phase, and move the rest from A'_3 to A''_1. In general, we intersect $a_{(i)}$ with $B_{(i-1)} = (\cap_{j=1}^{i-1} a_{(j)}) \cap B_2$, and if the intersection $B_{(i)} = a_{(i)} \cap B_{(i-1)}$ is empty, then S does not have any 2-cuttable pair; otherwise, we delete all arcs from T_2 or T'_2 that become disjoint from $B_{(i)}$, and among these deleted arcs, we first move those from A''_2 to

A_0'' that are also deleted from T_1 or T_1' in the first phase and then move the rest from A_3' to A_1''. Finally, we have $B_2' = B_{(l)}$ and $A_3'' = A_3'$.

The first phase takes $O(k \log n)$ time, where k is the number of arcs "absorbed" by B_1 plus the number of arcs deleted from T_1 and T_1'. Similar complexity is required for the second phase. Thus, step (1), by using the two phases, can be carried out in $O(n \log n)$ time.

Now we consider step (2) of the procedure, the goal of which is to compute all 2-cuttable pairs on (B_1, B_2). Clearly, if A_3 is empty, then (B_1, B_2) is 2-cuttable and, B_1, B_2 are largest intervals such that (B_1, B_2) is 2-cuttable. In such a case, all the 2-cuttable pairs for S can be expressed together as (B_1, B_2). Suppose now that A_3 is not empty. It can be seen that not all pairs on (B_1, B_2) are 2-cuttable. Let e be the beginning point of B_1. If B_1 and B_2 are disjoint, let ff be the beginning point of B_2. Otherwise, let ff be the beginning point of the first arc not intersected by e scanned clockwise from the beginning point of B_2, or let ff be e if that beginning point does not exist. Then, if e and ff are distinct, (e, ff) is 2-cuttable, while (e, x) is not, for any $x \in CCW(e, ff) - \{ff\}$. And, on the other hand, if $e = ff$, which happens when $B_1 = B_2 = \mathcal{K}$ and $e(= ff)$ is on the beginning point of the common intersection of all the arcs, then (e, ff) is again 2-cuttable. Let $E_1, E_2, \ldots, E_i, \ldots, E_p$ be the cut intervals contained in B_1, such that E_{i+1} follows E_i in the counterclockwise sense and $B_1 = \cup_{i=1}^{p} E_i$. Since only beginning points or ending points of arcs in A_3 can effect the 2-cuttability of a cut pair, our approach is, for each cut interval E_i on B_1, compute an interval F_i on B_2 such that (E_i, F_i) is 2-cuttable and F_i is the largest such interval. (In step (2), if E_i has empty F_i, this pair is not generated and index i is used for the next pair of intervals.) We start with E_1 with beginning point e, and look for F_1 which must have beginning point on ff. Since arcs in A_3 are proper, for two intervals E_i and E_{i+1} on B_1, there are only four cases for their corresponding F_i and F_{i+1} on B_2:

(1) The beginning point of F_{i+1} is at that of F_i, and $F_i \subseteq F_{i+1}$.

(2) The ending point of F_{i+1} is at that of F_i, and $F_{i+1} \subseteq F_i$.

(3) The beginning point of F_{i+1} is on F_i excluding the beginning point of F_i, and the ending point of F_i is on F_{i+1} excluding the ending point of F_{i+1}. This may happen if the common point of E_i and E_{i+1} is the beginning point of an arc and the ending point of some other arc.

(4) F_i and F_{i+1} are disjoint and F_{i+1} follows F_i in the counterclockwise scan of B_2. This happens when E_i and E_{i+1} are disjoint and may happen if the common point of E_i and E_{i+1} is the beginning point of an arc and the ending point of some other arc.

For these reasons, we can skip some scans for finding F_i's in step (2) so that the output ($O(n)$ pairs of intervals) can be computed in $O(n)$ time. Figure 5 shows three examples, where in (b) and (c) S contains two disjoint arcs and hence B_1 and B_2 obtained in step (1) are disjoint; while in (a) no two disjoint arcs exists and hence $B_1 = CCW(e, e) = \mathcal{K}$ and $B_2 = CCW(f, f) = \mathcal{K}$. We should mention

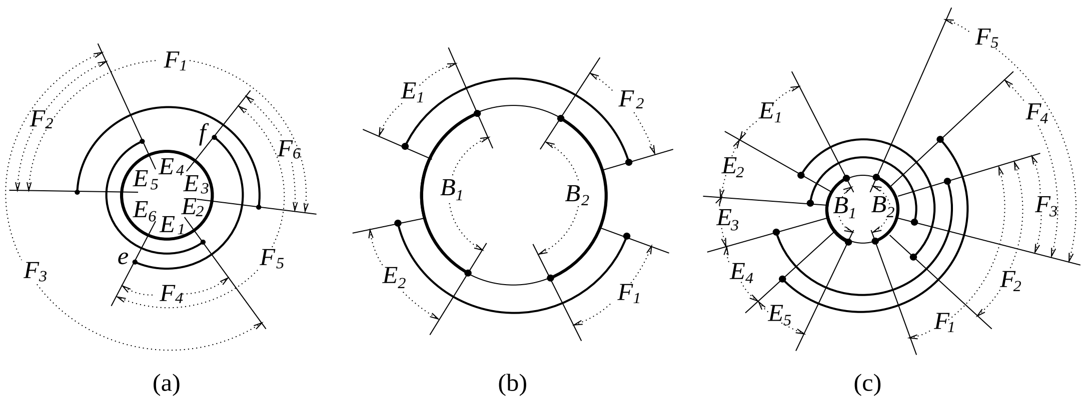

Fig. 5. Step (2) of PROCEDURE 2-CUTTABLE_(E_i, F_i)'s: (a) No two disjoint arcs. (b,c) With two disjoint arcs.

that the common endpoint of any E_i and E_{i+1} has two different intervals F_i and F_{i+1} on B_2 for it to pair with as 2-cuttable pair. If one of the two intervals F_i and F_{i+1} is contained in the other, such endpoint should be deleted from E_i or E_{i+1}, whichever has smaller interval on B_2 to pair with. Otherwise, such endpoint, say s, should be deleted from both E_i and E_{i+1}, and have its own 2-cuttable interval pair $(\{s\}, CCW($ the beginning point of F_i, the ending point of $F_{i+1})$. This step can be done in additional $O(n)$ time for all such endpoints but we leave it to until when we apply the procedure to solve the walkability problems.

We conclude that

Lemma 8 *To determine if there exists a 2-cuttable pair for a given set S of n proper closed arcs, and if so, compute all 2-cuttable pairs can be done in $O(n \log n)$ time.*

4. Polygon Walkability Test

Our polygon walkability test algorithm first finds a pair (s_0, g_0), if any, that satisfies the LR-visibility condition of Lemma 1. If there is no such pair, P is not walkable. If there is no deadlock for (s_0, g_0), we are done with a polygon which is walkable and report the pair (s_0, g_0) which admits a walk. Otherwise, we allow (s_0, g_0) to move counterclockwise, i.e., move s_0 and/or g_0 counterclockwise on the boundary of P, to a location (s, g) that avoids the deadlock detected and satisfies the LR-visibility condition. Since the new (s, g) may also have deadlocks, we do this counterclockwise moving iteratively, until we reach a pair (s, g) that has no deadlock and is LR-visible, or until (s, g) has passed (g_0, s_0) (i.e., s passed g_0 and g passed s_0). In the former case, (s, g) admits a walk and P is walkable. In the latter case, P is reported to be not walkable. We give the general structure of the algorithm below.

ALGORITHM *WALKABILITY_TEST*

(1) Compute all the chords of P;

(2) Find an LR-visible (s_0, g_0) as (s, g);
if not found **then** Report P is not walkable and Stop;

(3) **repeat**
if there is a deadlock **then**
Move (s, g) counterclockwise to avoid it;
Move (s, g) further counterclockwise, if necessary, to maintain LR-visibility;
until either $((s, g)$ is deadlock-free and LR-visible) **or** $((s, g)$ is past $(g_0, s_0))$;
if the first condition is true **then** Report P is walkable and output (s, g);
else Report P is not walkable;

END of *WALKABILITY_TEST*

Step (1) computes all the chords which are used in our algorithm to test LR-visibility and deadlocks conditions. This step can be done in $O(n \log n)$ time and $O(n)$ space by making use of the following result due to Guibas et al.[3]

> **Ray-shooting Query:** *Given a triangulation of P (which can be done in linear-time,[1]) an $O(n)$ space data structure can be built on P in $O(n)$ time such that given any point x inside P and any direction u, the first hit point of the ray in the direction u from x can be computed in $O(\log n)$ time.*

Their algorithm improves the original ray-shooting algorithm by Chazelle and Guibas[2] which has $O(n \log n)$ preprocessing time. The endpoints of the chords and vertices of P are then sorted in the order as they appear in the counterclockwise scan of the boundary of P starting from any point, which can be done in $O(n \log n)$ time.

To compute an LR-visible (s_0, g_0) in step (2), we first remove redundant chords[9] from the chords of P computed in step (1). Then, we transform our problem into the 2-cut circular arcs problem given in Section 3 in the following way. The boundary of P is mapped to a unit circle, and the polygonal boundary of the component of each non-redundant chord mapped into an arc on the circle such that the relative position of endpoints of non-redundant chords is the same as that of endpoints of arcs on the circle. Since there are at least two and at most $O(n)$ non-redundant chords in P, we have at least two arcs in the arc set S of size $O(n)$. Clearly, the mapping can be carried out in linear time when the boundary of P is traversed counterclockwise. After that, we determine if there exists a 2-cuttable pair for S in the 2-cut circular arcs problem and, if so, compute a 2-cuttable pair (c_1, c_2), in linear time, as discussed in Section 3.1. From the definition of a 2-cuttable pair and Corollary 1, a 2-cuttable pair for S corresponds to an LR-visible pair for P. If there is no LR-visible pair, we decide that P is not walkable by Lemma 1.

Now we give the details of step (3). If the (s_0, g_0) obtained in step (2) is deadlock-free, then (s_0, g_0) admits a walk by Lemma 1 and we are done with a walkable P. Otherwise, we may have s-deadlocks and/or g-deadlocks. To detect an s-deadlock, we do the following (see Figure 6 where $DK(v_1, v_2)$ is detected).

PROCEDURE *DEADLOCK_DETECTION*

(1) [initialization]

$l \longleftarrow s$; $r \longleftarrow s$;

(2) [case (1b2) deadlock?]

if s is a vertex with cw-chord $\overline{v_1 v_1''}$ and ccw-chord $\overline{v_1 v_1'}$, and
$g \in CCW(v_1, v_1'') - \{v_1''\}$ **then**
Scan $CW(s, v_1')$ from s for the first vertex v_2 with cw-chord $\overline{v_2 v_2'}$ such that
$v_2' \in CCW(v_1, g)$;
if no such vertex **then** $l \longleftarrow v_1'$; goto (4); **else** return $DK(v_1, v_2)$;

(3) [case (1c2)/(1c1) deadlock?]

if s is a vertex with cw-chord $\overline{v_2 v_2'}$ and ccw-chord $\overline{v_2 v_2''}$ **then**
Scan $CCW(s, v_2') - \{s\}$ from s for the first vertex v_1 with ccw-chord $\overline{v_1 v_1'}$
such that $v_1' \in CW(v_2, g)$;
if no such vertex **then** $r \longleftarrow v_2'$; goto (4);
if $g \in CW(v_2, v_2'') - \{v_2''\}$ **then** return $DK(v_1, v_2)$;
else
Scan $CW(s, v_2'') - \{s\}$ from s for the first vertex v_3 with cw-chord $\overline{v_3 v_3'}$
such that $v_3' \in CCW(s, g)$;
if found **then** return $DK(v_1, v_2)$;
else $l \longleftarrow v_2''$; $r \longleftarrow v_1$; goto (4);

(4) [left scan for a $\overline{v_2 v_2'}$ of case (1a) deadlock]

Scan $CW(l, g) - \{l\}$ from l for the first vertex v_2 with cw-chord $\overline{v_2 v_2'}$ such that
$v_2' \in CCW(r, g)$;
if no such vertex **then** stop;

(5) [right scan for a $\overline{v_1 v_1'}$ of case (1a) deadlock]

Scan $CCW(r, v_2')$ from r for the first vertex v_1 with ccw-chord $\overline{v_1 v_1'}$ such that
$v_1' \in CCW(g, v_2)$;
if found **then** return $DK(v_1, v_2)$; **else** $l \longleftarrow v_2$; $r \longleftarrow v_2'$; goto (4);

END of *DEADLOCK_DETECTION*

For convenience, we identify the *forbidden chain* pertaining to the deadlock cases in Figure 2. The boundary portions $CCW(v_1, v_1'') - \{v_1''\}$ in cases of (1b2) and (2b2), $CW(v_2, v_2'') - \{v_2''\}$, in cases of (1c2) and (2c2), and $CW(v_1, v_2) - \{v_1, v_2\}$ in all deadlock cases, are called the *forbidden chains*.

Suppose the procedure has detected an s-deadlock $DK(v_1, v_2)$. The deadlock is one of the four cases (1a), (1b2), (1c1), and (1c2) in the definition of a deadlock (see Figure 2)—case (1b1) is equivalent to case (1c1) as mentioned before and is detected in our procedure as case (1c1). It can be seen that if $s \neq v_1$, i.e., $DK(v_1, v_2)$ is of case

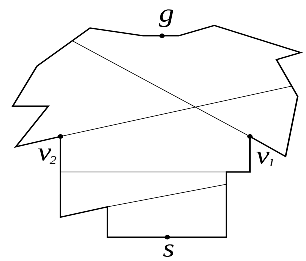

Fig. 6. Detection of s-deadlocks.

(1a), (1c1), or (1c2), then any (a, g) with a on the forbidden chain $CW(v_1, v_2) - \{v_1, v_2\}$ has deadlock $DK(v_1, v_2)$ too. Also, if s is v_1, i.e., $DK(v_1, v_2)$ is of case (1b2), then any (s, b) with b on the forbidden chain $CCW(v_1, v_1'') - \{v_1''\}$ has deadlock $DK(v_1, v_2)$ or is not LR-visible. Thus, for a deadlock of case (1a), (1c1), or (1c2), we move s out of the forbidden chain $CW(v_1, v_2) - \{v_1, v_2\}$ and put it at v_1 to avoid the deadlock, and for a deadlock of case (1b2), we move g out of the forbidden chain $CCW(v_1, v_1'') - \{v_1''\}$ and put it at v_1'' to avoid the deadlock.

If no s-deadlock is detected, we use a similar algorithm to detect any g-deadlock and avoid it (by moving s/g counterclockwise). Whenever an s-deadlock or g-deadlock is detected and resolved, a new (s, g) may not be LR-visible. In this case we perform an *advancing* step to ensure LR-visibility of (s, g), by moving (s, g) counterclockwise to (s', g') such that (s', g') is LR-visible. Advancing process is equivalent to computing $\mathrm{NEXT}(c_1, c_2)$ in the 2-cut circular arcs problem, where (c_1, c_2) corresponds to (s, g) and $\mathrm{NEXT}(c_1, c_2)$ corresponds to (s', g'). If the advancing moves (s, g) past (g_0, s_0), we report P is not walkable.

Moving (s, g) to avoid a deadlock while maintaining LR-visibility may result in a new deadlock, for which we have to repeat the above deadlock avoidance and advancing process. We proceed until either we obtain an (s, g) that is deadlock-free and LR-visible, or we have moved (s, g) past (g_0, s_0).

We show that step (3) of our algorithm can be done in $O(n)$ time, so that the time complexity of the algorithm is $O(n \log n)$. To maintain LR-visibility takes overall $O(n)$, as can be seen by considering the computation of NEXT in the 2-cut circular arcs problem. The deadlock detection takes overall $O(n)$ time if we take into consideration the following observations and modify deadlock detection procedure accordingly. Consider s-deadlock detections. For the clockwise scan (in steps (2), (3) and (4) of the deadlock detection procedure) in detecting an s-deadlock $DK(v_1, v_2)$ for (s, g), we scanned $CW(s, v)$ for some v on L. Let this scanned portion excluding v be denoted I. We then moved s to v_1 or moved g to v_1'', which may be followed by moving (s, g) counterclockwise either because of a g-deadlock or because of non-LR-visibility. Then, if we need to detect s-deadlock again for the current LR-visible pair (s, g), we do not have to scan I in the clockwise scan, because there is no deadlock $DK(p, q)$ for the current (s, g) such that p lies on I. For example, in Figure 6, there is no $DK(p, q)$ for (s', g'), where (s', g')

is the (s, g) position after avoiding $DK(v_1, v_2)$ and maintaining LR-visibility, and $p \in CW(s, v_2) - \{v_2\}$, because otherwise p would have been v_2. In general, the portion that we had scanned in the clockwise scan need not be scanned later. The same arguments apply to the detection of g-deadlocks. For each chord scanned, to determine if it is involved in forming a deadlock takes $O(1)$ time (once the endpoints of chords and vertices of P have been sorted in step (1) of our algorithm). As for the counterclockwise scan (in steps (3) and (5)), a chord may be checked several times to see if it is involved in forming a deadlock with some other chords, which may make our deadlock detections take $O(n^2)$ overall time. To overcome this, we do the following. We always search s- and g-deadlocks simultaneously, by alternating steps–that is, by interspersing instructions for searching an s-deadlock with instructions for searching a g-deadlock. In the search, when a deadlock is found, the work of both searches is "charged" to the successful search; if none is found, the walkability-test algorithm is done. Since the chords examined in a successful search for a current (s, g) will not be checked again for a new (s, g) (because the clockwise-scanned portion won't be visited again, as mentioned above, and the counterclockwise-scanned portion is passed over when moving the current (s, g) to the new (s, g) to avoid the deadlock found), successful searches in walkability-test algorithm take overall $O(n)$ chord examinations and therefore the total time spent for deadlock detections is $O(n)$.

The following subsection establishes the correctness of the algorithm.

4.1. Correctness

If the algorithm reports that P is walkable, it is correctly so because the final (s, g) has no deadlocks and is LR-visible. If, on the other hand, P is reported to be not walkable, we need to show that no (s, g) admits a walk. We claim that during the counterclockwise movement of (s, g) in step (3), any pair on the portions that (s, g) has passed does not admit a walk.

Lemma 9 *Let the deadlocks detected by the algorithm be $DK_0, DK_1, \ldots, DK_i, \ldots,$ DK_m, $m > 0$. Let (s_{i+1}, g_{i+1}), $0 \le i \le m$, be the (s, g) position after avoiding DK_i and maintaining LR-visibility. Then any (a, b) where a or b is on $CCW(s_0, s_{m+1}) \cup$ $CCW(g_0, g_{m+1}) - \{s_{m+1}, g_{m+1}\}$ does not admit a walk.*

Proof. First observe that if a or b lies in some forbidden chain $CW(v_1, v_2) - \{v_1, v_2\}$ of a deadlock $DK(v_1, v_2)$ in these DK_i's, then (a, b) does not admit a walk, because (a, b) either has deadlock $DK(v_1, v_2)$ or is not LR-visible (see Figure 2). So we assume in the following that neither a nor b in the claim lies in any of these forbidden chains, referred to as type $F1$ forbidden chains for convenience. Also observe that, by our algorithm, DK_i can be one of the deadlock cases (1a), (1b2), (1c1), (1c2), (2a), (2b2), (2c1), and (2c2).

We now show that if neither a nor b lies in any forbidden chain $CCW(v_1, v_1'') -$ $\{v_1''\}$ (called type $F2$ forbidden chain) in case (1b2) or (2b2) deadlock $DK(v_1, v_2)$ in the DK_i's, then (a, b) is not LR-visible and does not admit a walk. Let (s'_{i+1}, g'_{i+1}) be the (s, g) position right after avoiding DK_i. Suppose $a \in CCW(s'_k, s_k)$ and $b \in CCW(g'_l, g_l)$. Clearly, $s'_k \neq s_k$ and $g'_l \neq g_l$, by our assumption that neither a

nor b lies in any forbidden chain of type $F1$ or $F2$. Suppose (s'_k, g'_k) corresponds to (p,q) in the equivalent 2-cut circular arcs problem. Then, (s_k, g_k) corresponds to a 2-cuttable pair (p', q') which is NEXT(p,q) and $p \neq p'$. By Lemma 6, $l \leq k$. Similarly, we have $k \leq l$. So $k = l$. For $k = l$, any (a,b) is not LR-visible except at (s_k, g_k), which has deadlock DK_k.

Consider the situation when a or b lies in some type $F2$ forbidden chains in case (1b2) or (2b2) deadlocks. If (a,b) has such deadlock, (a,b) does not admit a walk. Otherwise, we show in the following that (a,b) is not LR-visible and thus does not admit a walk either. Without loss of generality, assume b is on a type $F2$ forbidden chain $CCW(v_1, v''_1) - \{v''_1\}$ in case (1b2) deadlock DK_i. For any a that is not in $Comp(\overline{v_1 v''_1})$, clearly (a,b) is not LR-visible. Otherwise, we have two cases: (1) $a \in CCW(s_0, v_1)$ and (2) $a \in CCW(v''_1, g_{m+1}) - \{g_{m+1}\}$. For case (1), since by assumption that a is not on any type $F1$ forbidden chain, either $a \in CCW(s'_k, s_k)$ for some $k \leq i$ and $s'_k \neq s_k$, or a is on some type $F2$ forbidden chain in case (2b2) deadlock $DK_h = DK(v_3, v_4)$ for some $h < i$. In the former case, (a,b) is not LR-visible, as can be seen by letting (s'_k, g'_k) be (p,q) in Lemma 6 (a). In the latter, (a,b) is not LR-visible, because the cw-chord $\overline{v_3 v''_3}$ has the component containing neither a nor b. Therefore, for case (1), (a,b) is not LR-visible. Similarly for case (2) we have $a \in CCW(g'_k, g_k)$ for some $k > i$ and $g'_k \neq g_k$, or a is on some type $F2$ forbidden chain in case (1b2) deadlock $DK_h = DK(v_3, v_4)$ for some $h > i$. In the former case, (a,b) is not LR-visible, as can be seen by letting (g'_k, s'_k) be (p,q) in Lemma 6 (a). In the latter, (a,b) is not LR-visible, because the cw-chord $\overline{v_3 v''_3}$ has the component containing neither a nor b. Therefore, for case (2), (a,b) is again not LR-visible. □

Consider the pair which is the (s,g) position after avoiding the last detected deadlock in our algorithm and maintaining LR-visibility. (The pair is (s_{m+1}, g_{m+1}) in the lemma.) If the pair is not past (g_0, s_0), it admits a walk. Otherwise, no (a,b), where a or b is on $CCW(s_0, s_{m+1}) \cup CCW(g_0, g_{m+1})$, admits a walk, by the above lemma. In such case, since $CCW(s_0, s_{m+1}) \cup CCW(g_0, g_{m+1})$ is the whole boundary of P, we conclude that no pair (s,g) on the boundary of P admits a walk. Therefore, we report P is not walkable.

5. Polygon Straight Walkability Test

Lemma 10 *(a) If (s,g) admits a straight walk, then any wedge of P contains s or g, or both. (b) If any wedge of P contains s or g or both, then (s,g) does not have any case-(1) left wedge or case-(1) right wedge.*

Proof. (b) can be easily proved by the definitions of case-(1) left wedge and case-(1) right wedge. For (a), suppose there exists a wedge $W(v_1, v_2)$ with ccw-chord $\overline{v_1 v'_1}$ and cw-chord $\overline{v_2 v'_2}$, such that neither s nor g lies in $W(v_1, v_2)$. In case both s and g lie in $P - Comp(\overline{v_1 v'_1})$ or both in $P - Comp(\overline{v_2 v'_2})$, then (s,g) is not LR-visible. If it is not the case, then $W(v_1, v_2)$ is a left wedge or a right wedge for (s,g). In either case the condition of Lemma 2 is violated, and thus (s,g) does not admit a straight walk. □

Consider positioning (s,g) on the boundary of P such that (s,g) satisfies the necessary condition of Lemma 10 (a). If we transform each wedge into an arc similar to what we did in Section 3 for polygonal boundary of a component, we have a 2-cut circular arcs problem. The arcs are to be intersected by two cuts because wedges have to contain s or g, which implies that (s,g) does not have case-(1) left wedge or case-(1) right wedge, by Lemma 10 (b). There could be $O(n^2)$ number of wedges in P, but some wedges of P are useless in the sense that we can delete them without affecting the correctness of Lemma 10. We say that a wedge is *non-redundant* if it does not fully contain another wedge. It is clear only non-redundant wedges determine the conditions of Lemma 10. The following procedure finds the set of all non-redundant wedges W, which is of size $O(n)$.

PROCEDURE *NON-REDUNDANT_WEDGES*

(1) Start with u of any ccw-chord $\overline{uu'}$; Mark the chord;

(2) Scan boundary of P counterclockwise from u until back to u:
if u' of a marked chord $\overline{uu'}$ is scanned **then** Unmark that chord;
if u of a ccw-chord $\overline{uu'}$ is scanned **then** Mark the chord;
if v of a cw-chord $\overline{vv'}$ is scanned **and** the chord intersects some of the previously marked chords
then
Among these chords, choose the u such that $CCW(u,v)$ is the smallest;
Output $CCW(u,v)$ as a non-redundant wedge;
Unmark chords that were marked and scanned before u (inclusive);

(3) **if** there exists some marked chord **then**
repeat the procedure once using the same starting vertex
but considering only chords that do not form a wedge in the first run;

END of *NON-REDUNDANT_WEDGES*

Note that, in step (2), after a non-redundant wedge is found, we unmark chords that were marked before — in fact, if any of these chords can be paired with a later chord to form a wedge, the wedge must be redundant because it contains the non-redundant wedge just found. Step (3) is needed because there may be some non-redundant wedge missed by step (2) if our start vertex u in step (1) is contained in a wedge.

The time complexity for the procedure is $O(n \log n)$ if we maintain a binary tree of the vertices (u') of marked chords $(\overline{uu'})$ (whose search, insert, and delete operations can be done in $O(\log n)$ time) such that for each $\overline{vv'}$, one of the marked chords that pairs with $\overline{vv'}$ to form the "smallest" wedge in step (2) can be found by searching the tree in $O(\log n)$ time.

Recall that in the walkability test, we need to find one (s,g) that is LR-visible and deadlock-free according to Lemma 1. LR-visibility is maintained by equivalently keeping two cuts intersecting a set S of arcs in the 2-cut circular arcs problem. For the straight walkability test, we first find one (s,g) that, in addition to being LR-visible and deadlock-free, satisfies the necessary condition of Lemma 10 (a). If we apply the walkability test algorithm by adding some more arcs, each corresponding to a non-redundant wedge in W, into S (and removing non-proper arcs from them), then keeping two cuts intersecting the new S is equivalent to maintaining LR-visibility and also satisfying the necessary condition of Lemma 10 (a). We then modify the remaining part, i.e., the deadlock detection and avoidance, to make sure the final (s, g) does not have case-(2) left/right wedge, as follows. If there is a deadlock for the current (s, g), we just follow the original deadlock avoidance. Now, suppose there is no deadlock detected. We detect if s is on v_2 of some $W(v_1, v_2) \subseteq L$ or on v_1 of some $W(v_1, v_2) \subseteq R$. If not, (s, g) has no case-(2) left/right wedge, and so (s, g) admits a straight walk. Otherwise, consider first the case-(2) left wedge (Figure 3 (b)). This is the case when (s^-, g) is not LR-visible, i.e., s to s^- clockwise leaves a component not containing g, or (s^-, g) has an s-deadlock (this can be tested when we were detecting deadlocks for (s, g)). If so, we move g to v_1 to avoid the wedge. Next, consider the case-(2) right wedge. We observe that a case-(2) right wedge $W(v_1, v_2)$ in which (s^+, g) has an s-deadlock is actually detected as a deadlock $DK(v_1, x)$ as shown in Figure 3 (d.1) and thus the wedge is avoided when we do the deadlock avoidance. So we just consider case-(2) right wedge $W(v_1, v_2)$, in which (s^+, g) is not LR-visible (Figure 3 (d.2)). In this case, we move g to v_1'' ($\overline{v_1 v_1''}$ is the cw-chord) to avoid the wedge. Thus, case-(2) left/right wedge is avoided for current (s, g).

It can be seen the time complexity remains the same as for polygon walkability test. The proof of correctness is similar to that for polygon walkability test by observing from the lemma below that when we move g to avoid a left/right wedge in the above discussion, the part g passed does not allow pairs to admit a straight walk.

Lemma 11 *(1) If (s, g) has a case-(2) left wedge $W(v_1, v_2)$ (see Figure 3 (b)), then, for $b \in CCW(g, v_1) - \{v_1\}$ and $a \in CCW(s, b)$, (a, b) does not admit a straight walk. (2) If (s, g) has a case-(2) right wedge $W(v_1, v_2)$ in which (s^+, g) is not LR-visible and $\overline{v_1 v_1''}$ is a cw-chord (see Figure 3 (d.2)), then, for $b \in CCW(g, v_1'') - \{v_1''\}$ and $a \in CCW(s, b)$, (a, b) does not admit a straight walk.*

Proof. Consider (1). If $a \neq s$, then (a, b) in the lemma has case (1) left wedge $W(v_1, v_2)$ or is not LR-visible. Otherwise, (a, b) has case-(2) left wedge $W(v_1, v_2)$ or is not LR-visible. By Lemma 2, (a, b) does not admit a straight walk. Now, consider (2). If $a \neq s$, then (a, b) is not LR-visible. Otherwise, (a, b) has case-(2) right wedge $W(v_1, v_2)$. Again, (a, b) does not admit a straight walk by Lemma 2. □

We conclude that polygon straight walkability test can be done in $O(n \log n)$ time and in linear space.

6. Polygon (Straight) Counter-walkability Test

The following lemma gives us a method for testing polygon straight counter-walkability in $O(n \log n)$ time.

Lemma 12 *A simple polygon P is straight counter-walkable if and only if it has no two components that are disjoint.*

Proof. For the *only if* part, suppose P has two components that are disjoint. Then, one of the cases (a–d) of Lemma 3 (see Figure 4) must be true for any (s, g)

that is LR-visible. Thus P is not straight counter-walkable by Lemma 3.

For the *if* part, if P has no disjoint components, each non-redundant chord must intersect the component of any other chord. Let the two endpoints of some non-redundant chord be s and g. Clearly, s and g are visible. L and R are visible, for otherwise there must exist a chord with component unintersected by $\overline{sg}$. Also none of the cases (a–d) of Lemma 3 applies for the (s, g). So (s, g) admits a straight counter-walk and P is straight counter-walkable. □

Our polygon straight counter-walkability test algorithm first computes all the non-redundant chords. Then, in $O(n)$ time, it tests if there exist two disjoint components to determine the straight counter-walkability, according to the above lemma. If P is straight counter-walkable, one (s, g) that admits a straight counter-walk can be computed by choosing the two endpoints of any non-redundant chord, as in the *if*-part proof of the above lemma. That concludes our $O(n \log n)$-time and linear-space algorithm for the polygon straight counter-walkability test.

Now we begin the discussion of polygon counter-walkability test. One of the necessary conditions for (s, g) to admit a counter-walk is LR-visibility. The following lemma gives the relationship between a weakly star-shaped polygon and a polygon which contains an LR-visible (s, g).

Lemma 13 *If P is weakly star-shaped then there exist two points s and g on the boundary of P such that (s, g) is LR-visible.*

Proof. Suppose P is weakly star-shaped. By definition, there exists a weakly visible segment $\overline{ab}$ inside P such that each point on the boundary of P is visible from some point on $\overline{ab}$. Let s and g be the first hit points on the boundary of P by $\overrightarrow{ba}$ and $\overrightarrow{ab}$ respectively. For each point p on L, let $q \in \overline{ab}$ be a point visible from p, and let p' be the first hit point on R by $\overrightarrow{pq}$. Then, p and p' are visible. Namely, each point on L is visible from some point on R. Similarly, each point on R is visible from some point on L. Thus, (s, g) is LR-visible. □

The following lemma suggests an $O(n \log n)$ time algorithm for polygon counter-walkability test.

Lemma 14 *A simple polygon P is counter-walkable if and only if P has a weakly visible segment $\overline{sg}$ with s and g on the boundary of P such that neither case (a) nor (b) in Lemma 3 applies.*

Proof. From Lemma 4, it suffices to show $\overline{sg}$ is a weakly visible segment if and only if s and g are visible and (s, g) is LR-visible. Consider the *only if* part. If $\overline{sg}$ is a weakly visible segment, then clearly s and g are visible. That (s, g) is LR-visible can be seen by the proof of the above lemma. For the *if* part, suppose s and g are visible and (s, g) is LR-visible. By the latter condition, each point $p \in L$ is visible from some $q \in R$. Then it can be seen p is visible from the intersection point of $\overline{pq}$ and $\overline{sg}$. Namely, each point on L is visible from some point on $\overline{sg}$. Similarly, each point on R is visible from some point on $\overline{sg}$. So each point on the boundary of P is visible from some point on $\overline{sg}$, and hence $\overline{sg}$ is a weakly visible segment. □

To test if P has a weakly visible segment and, if so, compute one can be done in $O(n \log n)$ time.[7] The algorithm is based on the equivalence that a segment inside P is a weakly visible segment if and only if it intersects $Comp(c)$ for each non-redundant chord c. And, the algorithm is almost equivalent to computing B_1 and B_2 in step (1) of the procedure in Section 3.2 (for computing all 2-cuttable pairs for the 2-cut circular arcs problem), as given below.

(1) Let each component of some non-redundant chord in our P map to an arc in S, such that the endpoints of non-redundant chords are in the same order as that for arcs.

(2) If there exists no two disjoint components, any non-redundant chord is a weakly visible segment. This corresponds to a 2-cuttable pair with two cuts at endpoints of an arc.

(3) Otherwise, two regions P_1 and P_2 corresponding to B_1 and B_2 are such that the polygonal boundary of P_1 (i.e., intersection of P_1 with the boundary of P) corresponds to B_1, and the polygonal boundary of P_2 corresponds to B_2. Indeed the output has $P_1 = \cap_{c \in C_1} Comp(c)$, $P_2 = \cap_{c \in C_2} Comp(c)$, and C_3, where C_1, C_2, and C_3 are each a set of non-redundant chords corresponding to the arc sets A_1, A_2, and A_3 respectively.

From these relations, we observe the following: P_1 and P_2 are both non-empty and mutually disjoint, and all chords in C_3 intersect both P_1 and P_2. And, a segment inside P is a weakly visible segment if and only if it intersects both P_1 and P_2, and the components of chords in C_3. Let $CCW(b_1, e_1)$ and $CCW(b_2, e_2)$ be the polygonal boundary of P_1 and P_2 respectively. Also, let π_1 be the geodesic path inside P between b_1 and e_2, and π_2 be the geodesic path inside P between e_1 and b_2. If P_1 and P_2 are not visible from each other, i.e., each point in one region is not visible from any point in the other region, then no segment can intersect the two regions simultaneously. It follows that P is not weakly star-shaped and, by Lemma 14, P is not counter-walkable. Otherwise, π_1 and π_2 are two disjoint convex chains.

We now show, with the above construction, how to determine if there exists (s, g) such that condition of Lemma 14 is true. If (2) above is true, we know P is counter-walkable since condition of Lemma 14 is true. We now assume (3) above is true. First, we find two disjoint components $Comp(a)$ and $Comp(b)$ such that a and b are both ccw-chords or both cw-chords in $O(n)$ time as follows.

(1) Remove all cw-chords and redundant ccw-chords from the chords of P, and then determine if there exist two component-disjoint ccw-chords.

(2) Determine if there exist two component-disjoint cw-chords similarly.

If two component-disjoint ccw- or cw-chords a and b were not found, we extend both endpoints of a weakly visible segment to touch the boundary of P and let the two touch points be s and g. Then $\overline{sg}$ satisfies the condition of Lemma 14 and so P is counter-walkable. Otherwise, we compute C_a as the set of chords of the same type as a (inclusive) and each with component disjoint from $Comp(b)$, in $O(n)$ time. Similarly, we get C_b as the set of chords of the same type as b

(inclusive) and each with component disjoint from $Comp(a)$. It is not difficult to see that if $C_a \neq \{a\}$ and $C_b \neq \{b\}$, then case (a) in Lemma 3 applies for any (s,g) that is LR-visible and thus P is not counter-walkable. Otherwise, we consider without loss of generality the case when $C_a = \{a\}$ and let a be $\overline{v_a v_{a'}}$. It can be seen that the polygonal boundary of $Comp(a)$, ψ_a, is contained in (or equal to) either $CCW(b_1, e_1)$ or $CCW(b_2, e_2)$. Without loss of generality consider the case when $\psi_a \subseteq CCW(b_1, e_1)$. If $(s,g) \in (CCW(b_1,e_1), CCW(b_2,e_2))$ admits a counter-walk, s must be v_a because otherwise (s,g) is not LR-visible or case (a) in Lemma 3 applies. So, if (s,g) admits a counter-walk, v_a must be b_1. Thus, we let s be $v_a = b_1$. If we can find $g \in CCW(b_2,e_2)$ such that s and g are visible, $\overline{sg}$ intersects all components of chords in C_3, and case (b) in Lemma 3 does not apply, then (s,g) admits a counter-walk and P is counter-walkable. The existence of such a g can be determined as follows. After we compute all the 2-cuttable pairs in the equivalent 2-cut circular arcs problem (Section 3.2), we can get 2 disjoint chains $CCW(x_1,x_2)$ and $CCW(y_1,y_2)$ such that $x_1 = b_1$ and, for any $q \in CCW(y_1,y_2)$, (b_1,q) is LR-visible. Since s is b_1, we need to determine if there exists $g \in CCW(y_1,y_2)$ $\subseteq CCW(b_2,e_2)$ such that s and g are visible so that $\overline{sg}$ intersect all components in C_3. Observe that if $\pi_1 \neq \overline{b_1 e_2}$, then case (b) in Lemma 3 must be true for any $(s,g) \in (\{b_1\}, CCW(b_2,e_2))$. It follows that P is not counter-walkable. Otherwise, let $y_1' \in CCW(b_2,e_2)$ be such that $\overline{sy_1'}$ is the tangent from s to π_2. If y_1' is not on $CCW(b_2,y_2)$, then there is no g such that both visibility and LR-visibility of $(s = v_a = b_1, g)$ are met. In this case P is again not counter-walkable. Otherwise, g must be in $G = CW(y_2,y_1) \cap CW(y_2,y_1')$ in order that s and g are visible and (s,g) is LR-visible, i.e., $\overline{sg}$ is a weakly visible segment. We still need to find a $g \in G$ such that case (b) in Lemma 3 does not apply. For this, we scan $CW(e_2,y_1) \cap CW(e_2,y_1')$ clockwise from e_2 for the first vertex v such that it is visible to $s = v_a = b_1$ but its following vertex is not. If such v found is not on G, i.e., $v \in CW(e_2,y_2) - \{y_2\}$, then case (b) in Lemma 3 must be true for our s and any $g \in G$ visible from s and so P is not counter-walkable. Otherwise, $\overline{sy_2}$ is a weakly visible segment and neither case (a) nor (b) in Lemma 3 applies. By Lemma 14, P is counter-walkable with (b_1,y_2) admitting a counter-walk. Thus, we have a polygon counter-walkability test algorithm that runs in $O(n \log n)$ time and linear space.

7. Reporting All Pairs

We consider the following two problems:

(i) Given P, report all the point pairs that admit a (straight) walk.

(ii) Given P, report all the vertex pairs that admit a (straight) counter-walk.

We say a pair (C_1,C_2) of two chains on the boundary of P is LR-visible if, for any (a,b) on (C_1,C_2), (a,b) is LR-visible.

7.1. Reporting Pairs that Admit a (Straight) Walk

Given P, a pair (s,g) that admits a walk must be LR-visible and have no deadlocks by Lemma 1. Our approach for reporting all pairs that admit a walk is to compute all pairs that are LR-visible and then, eliminate from them the pairs that have deadlocks. First, consider computing all (s, g)'s that are LR-visible. By solving the equivalent problem of computing all 2-cuttable pairs for arcs (Section 3.2), we can determine if there exists an LR-visible pair (s, g) and, if so, compute all the LR-visible (s, g)'s expressed as a set of $O(n)$ LR-visible (S_i, G_i)'s, where S_i and G_i are two chains on the boundary of P and (S_i, G_i) corresponds to a 2-cuttable pair of intervals in the 2-cut circular arcs problem. The transformation is similar to what we do in Section 4 and the equivalence of the two problems can be easily verified. Let the set of the LR-visible pairs obtained be $U = \{(S_1, G_1), (S_2, G_2), \ldots, (S_i, G_i), \ldots, (S_p, G_p)\}$, where each (S_i, G_i) corresponds to (E_i, F_i) in the procedure of Section 3.2. Also, let S and G be the two chains corresponding to B_1 and B_2 (respectively) of that procedure.

If U is empty, i.e., there is no LR-visible pair, then P is not walkable. Otherwise, we now need to take into consideration the deadlocks—we want to remove, for each (S_i, G_i), any $(a, b) \in (S_i, G_i)$ that has deadlock. In the following, we discuss how to remove (a, b)'s that have s-deadlocks. To remove (a, b)'s that have g-deadlocks can be similarly done. We assume S and G are disjoint, while non-disjoint S and G will be discussed later. Let us examine the five possible cases of an s-deadlock $DK(v_1, v_2)$ (see Figure 2). It can be shown that (1) any (a', b') does not admit a walk if a' lies in the forbidden chain $F1 = CW(v_1, v_2) - \{v_1, v_2\}$, (2) if $DK(v_1, v_2)$ is of case (1b2) (resp. (1c2)), then any (v_1, b') (resp. (v_2, b')) does not admit a walk if b' lies in the forbidden chain $F2 = CCW(v_1, v_1'') - \{v_1''\}$ (resp. $F3 = CW(v_2, v_2'') - \{v_2''\}$), and (3) if $DK(v_1, v_2)$ is of case (1b1) (resp. (1c1)), then any (v_1, b') (resp. (v_2, b')) does not admit a walk. Therefore, the removing of (a, b)'s that have s-deadlocks consists of the following three tasks. (1) Remove points in S_i on $F1$ of some s-deadlock $DK(v_1, v_2)$. (2) Remove $(v_1, b') \in (S_i, G_i)$ for b' on $F2$ of case-(1b2) deadlock $DK(v_1, v_2)$ and remove $(v_2, b') \in (S_i, G_i)$ for b' on $F3$ of case-(1c2) deadlock $DK(v_1, v_2)$. (3) Remove point v in S_i that is not on some $F1$ but both v^+ and v^- are. For example, in Figure 7, we (1) remove from S_i three chains $CW(v_3, v_4) - \{v_3, v_4\}$, $CW(v_3', v_3) - \{v_3', v_3\}$, and $CW(s', v_6) - \{v_6\}$ because of the three type $F1$ forbidden

chains $DK(v_3, v_4)$, $DK(v_3', v_3)$, and $DK(v_5, v_6)$, (2) remove s from S_i and add another LR-visible pair $(\{s\}, G_i - F2)$, because $DK(v_1, v_2)$ is a case-(1b2) deadlock, and (3) remove v_3 from S_i. In general, after the removal of forbidden chains, the new S_i, if not empty, is a set of *admissible* sub-chains $S_{i1}, S_{i2}, \ldots, S_{ix}$. Before giving the procedure for our tasks, we observe the following. For $(a, b) \in (S_i, G_i) = (CCW(s, s'), CCW(g, g'))$ and an s-deadlock $DK(v_1, v_2)$ of (a, b) with cw-chords $\overline{v_2 v_2'}$, $\overline{v_1 v_1''}$, and ccw-chords $\overline{v_1 v_1'}$, $\overline{v_2 v_2''}$, we have

(1) $v_2' \in CCW(v_2, g)$ because otherwise (S_i, G_i) is not LR-visible due to $\overline{v_2 v_2'}$ (by observing that $P - Comp(\overline{v_2 v_2'})$ would contain $g \in G_i$ and some point in S_i, violating Lemma 5. Similarly, we have $v_1' \in CCW(g', v_2)$.

(2) If $v_1 \neq s'$, then $DK(v_1, v_2)$ is not of case (1b2), because otherwise (S_i, G_i) is

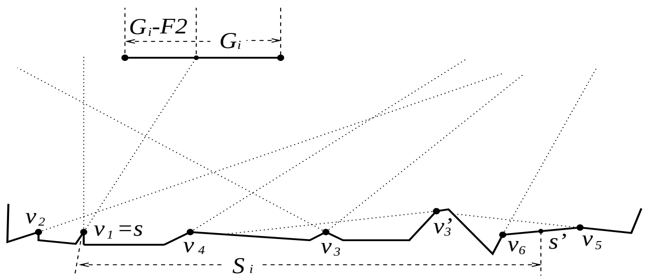

Fig. 7. Removing forbidden chains in S_i.

not LR-visible due to $\overline{v_1 v_1''}$. If $a = v_1 = s'$, then $DK(v_1, v_2)$ may be of case (1b2), but any $(v_1, b') \in (\{v_1\}, G_i - F2) = (\{v_1\}, G_i \cap CW(v_1, v_1''))$ has no deadlock of case (1b2). If $G_i - F2$ above is empty and s' is the beginning point of S_{i+1}, then any (v_1, b') has no deadlock of case (1b1).

(3) If $v_2 \neq s$, then $DK(v_1, v_2)$ is not of case (1c2), because otherwise (S_i, G_i) is not LR-visible due to $\overline{v_2 v_2''}$. If $a = v_2 = s$, then $DK(v_1, v_2)$ may be of case (1c2), but any $(v_2, b') \in (\{v_2\}, G_i - F3) = (\{v_2\}, G_i \cap CCW(v_2, v_2''))$ has no deadlock of case (1c2). If $G_i - F3$ above is empty and s is the ending point of S_{i-1}, then any (v_2, b') has no deadlock of case (1c1).

For disjoint S and G, the following procedure removes forbidden chains of s-deadlocks.
PROCEDURE *ADMISSIBLE_CHAINS*

(1) $b_S \longleftarrow$ beginning point of S; $e_S \longleftarrow$ ending point of S;
$b_G \longleftarrow$ beginning point of G; $e_G \longleftarrow$ ending point of G;

(2) **if** (b_S, b_G) has a deadlock **then**
Using deadlock avoidance of Section 4, move (b_S, b_G) counterclockwise
until (b_S, b_G) is deadlock-free and LR-visible or passes original (b_G, b_S);
In the latter case, report P is not walkable and stop;
Let the set of (new) LR-visible pairs be $U = \{(S_1, G_1), \ldots, (S_q, G_q)\}$;
$b_S \longleftarrow$ beginning point of S_1; $b_G \longleftarrow$ beginning point of G_1;

(3) Scan $CCW(b_S, b_G)$ from b_S to b_G:
if a vertex $u \in CCW(b_s, e_s)$ with cw-chord $\overline{uu'}$ and $u' \in CCW(u, b_S)$
then insrt(BT, $\overline{uu'}$);
if a vertex u' of a cw-chord $\overline{uu'}$ in BT **then** delete(BT, $\overline{uu'}$);
if a vertex v_1 with ccw-chord $\overline{v_1 v_1'}$, and cw-chord $\overline{v_1 v_1''}$
then if srch(BT, v_2)
then rm(U, $CW(v_1, v_2) - \{v_1, v_2\}$);
if v_1 is the ending point of some S_i
then $S_i' \longleftarrow \{v_1\}$; $G_1' \longleftarrow G_i \cap CW(v_1, v_1'')$;
Remove v_1 from S_i;
if $G_i' \neq \emptyset$ **then** Add (S_i', G_i') to U right after (S_i, G_i);
else **if** v_1 is the beginning point of some S_{i+1} **then** mark v_1 in S_{i+1};
if v_2 is the beginning point of some S_i

then $S'_i \longleftarrow \{v_2\}$; $G'_1 \longleftarrow G_i \cap CCW(v_2, v''_2)$;
Remove v_2 from S_i;
if $G'_i \neq \emptyset$ **then** Add (S'_i, G'_i) to U right before (S_i, G_i);
else **if** v_2 is the ending point of some S_{i-1} **then** mark v_2 in S_{i-1};

(4) For each $S_i = \{S_{i1}, \ldots, S_{ij}, \ldots, S_{ik}\}$ which is not added into U in step (3):
Remove any S_{ij} which is a single vertex;

(5) For each S_i which was added into U in step (3):
if $S_i = S_{i+1}$ **then** Remove both (S_i, G_i) and (S_{i+1}, G_{i+1});
if the vertex is marked in step (3) **then** Remove (S_i, G_i);

END of *ADMISSIBLE_CHAINS*

We start by imagining a pair (b_S, b_G) on the two beginning points of S and G. If there is a deadlock for the pair, we move it counterclockwise, avoiding deadlocks and maintaining LR-visibility, as discussed in Section 4, to obtain (b_S, b_G) that is deadlock-free and LR-visible, or find out P is not walkable. Suppose P is walkable and suppose the deadlock-free and LR-visible (b_S, b_G) is on (S_k, G_k). We delete (S_i, G_i) from U for $1 \leq i \leq k-1$, trim off the portion that (b_S, b_G) had passed on S_k, G_k, and possibly some G_i's that intersect b_G, and then rename the pairs $(S_k, G_k), (S_{k+1}, G_{k+1}), \ldots, (S_p, G_p)$ in U as $(S_1, G_1), (S_2, G_2), \ldots, (S_q, G_q)$, respectively, where $q = p - k + 1$. Next, we scan counterclockwise from the beginning point of S_1 to the beginning point of G_1, i.e., $CCW(b_S, b_G)$ in step (3), searching for forbidden chains. Each forbidden chain $F1 = CW(v_1, v_2) - \{v_1, v_2\}$ of an s-deadlock $DK(v_1, v_2)$ is formed by a cw-chord $\overline{v_2 v'_2}$ intersecting a ccw-chord $\overline{v_1 v'_1}$. Notice that, since b_S and b_G do not lie in any forbidden chain because of the above deadlock-avoidance, we only consider $\overline{v_2 v'_2}$ with v_2 on $CCW(b_S, e_S)$. And because of observation (1) above, we only consider $\overline{v_1 v'_1}$ with v_1 on $CCW(b_S, b_G)$. To find forbidden chains of this kind, each time a possible $\overline{v_2 v'_2}$ is found in the scan, we insert it into a binary tree BT of such chords ordered according to v_2's (by insrt(BT, $\overline{v_2 v'_2}$) in step (3)), and each time a possible $\overline{v_1 v'_1}$ is found, we search BT using v'_1 to find a $\overline{v_2 v'_2}$ which intersects $\overline{v_1 v'_1}$ to form a deadlock and such that $CW(v_1, v_2)$ is the largest among chords in BT intersecting $\overline{v_1 v'_1}$ (by srch(BT, v_2) which returns true along with the information of the desired chord, if a forbidden chain is found; otherwise, it returns false). If such a forbidden chain is found, we remove the part from each S_i that is in the forbidden chain (by rm($U, CW(v_1, v_2) - \{v_1, v_2\}$)). Then, we consider two possible cases, the first one being case-(1b2) deadlock $DK(v_1, v_2)$ for (v_1, b') and the second one, case-(1c2) deadlock $DK(v_1, v_2)$ for (v_2, b'). The first case happens only when v_1 is the ending point of an S_i (observation (2)). In this case, we remove v_1 from S_i and add another LR-visible pair $(\{v_1\}, G_i - F2)$ into U. The second case happens only when v_2 is the beginning point of an S_i, (observation (3)). In this case, we remove v_2 from S_i and add another LR-visible pair $(\{v_2\}, G_i - F3)$ into U.

We have taken care of the first two required tasks of removing pairs. The third task of removing the point pairs (a', b')'s that have case-(1b1) or (1c1) deadlocks is done by steps (4) and (5) which we explain. First suppose vertex a' of such an

(a', b') is not on the endpoints of some original S_i. Because a'^+ and a'^- are in some $F1$'s and have been removed in step (3), a' must be in a sub-chain consisting of only a'. Step (4) thus removes such a'. Next, suppose a' is on an endpoint of some original S_i. We consider the case when a' is also on an endpoint of S_{i-1} or S_{i+1} and the case when a' is not. For the first case, without loss of generality, let a' be on the ending point of S_{i-1} and on the beginning point of S_i. Again, a'^+ and a'^- are in some $F1$'s and have been removed in step (3). a' was also removed from both S_{i-1} and S_i. By observations (2) and (3) above, both G'_{i-1} and G'_i in step (3) are not empty and hence $(\{a'\}, G'_{i-1})$ and $(\{a'\}, G'_i)$ were added into U. Such two new LR-visible pairs are next to each other and step (5) removes them. For the second case, we have similar situation as in the first case, except that only one new LR-visible pair $(\{a'\}, G'_i)$ might be added. If a' is on the beginning point of S_i, then a' was marked in step (3) when a' was considered as a v_1. If, on the other hand, a' is on the ending point of S_i, then a' was marked in step (3) when a' was considered as a v_2. So, the new pair $(\{a'\}, G'_i)$ must have a' marked before, and step (5) removes this pair.

In the above discussion, we assume that S and G are disjoint in the beginning. If that is not the case (for example, see Figure 5 (a)), it is not difficult to see the idea of removing forbidden chains still works, but in step (3), we need to scan the whole boundary of P from b_S counterclockwise to e_S looking for possible $\overline{v_2 v_2'}$'s.

The time complexity is $O(n \log n)$ by using the binary tree BT in which each insertion or search can be done in $O(\log n)$. A similar procedure can be derived to remove the point pairs that have g-deadlocks. Our final result, if P is walkable, is $U = \{(S_1, G_1), \ldots, (S_i, G_i), \ldots, (S_r, G_r)\}$, where each S_i is a set of disjoint sub-chains $S_{i1}, S_{i2}, \ldots, S_{ix}$ and each G_i is a set of disjoint sub-chains $G_{i1}, G_{i2}, \ldots, G_{iy}$, such that each $(a, b) \in (S_{ih}, S_{ik})$, for $1 \le h \le x$ and $1 \le k \le y$, admits a walk. Although the sum of the number of admissible chains on all G_i's can be $O(n^2)$ (because G_i's may overlap), the total number of distinct admissible chains on the union of G_i's is $O(n)$. By keeping an ordered set of the latter $O(n)$ admissible chains and representing each G_i as a subset of the ordered set, our output takes $O(n)$ space. Finally, as we mentioned in Section 3.2 for computing all 2-cuttable pairs of an arc set, we may need to remove some endpoints from the original S_i's because such endpoint may have two different chains to pair with. Doing this takes additional $O(n)$ time. We conclude that all the point pairs that admit a walk can be computed in $O(n \log n)$ time using $O(n)$ space.

We now turn to the problem of reporting all point pairs that admit a straight walk. We first introduce non-redundant wedges as additional arcs into the original set of arcs of the equivalent 2-cut circular arcs problem for walkability discussed above, as we did in Section 5, and it is clear that the above idea for computing LR-visible pairs also applies for the new set of arcs, so that we can compute $O(n)$ LR-visible and case-(1) left/right wedge-free pairs of intervals in $O(n \log n)$ time. Then we can just follow the above procedure for computing admissible chains and obtain the output in a similar form with the same time and space complexity. One problem is that the output may contain point pairs that have case-(2) left/right wedges. We show below that such cases never happen.

Take one pair (S_i, G_i) in the output with $S_i = \{S_{i0}, S_{i1}, \ldots, S_{ij}, \ldots, S_{ix}\}$ and $G_i = \{G_{i0}, G_{i1}, \ldots, G_{ik}, \ldots, G_{iy}\}$. For a reflex vertex s in S_{ij} and any g in some G_{ik}, we first consider case-(2) left/right wedges in which (s^-, g) is not LR-visible (Figure 3 (b.2)) or (s^+, g) is not LR-visible (Figure 3 (d.2)). If (s, g) has a case-(2) left wedge in which (s^-, g) is not LR-visible, then s must be on the beginning point of S_i because otherwise S_i and G_i would not be LR-visible. But for such s and g, (s^+, g) has a case-(1) left wedge which contradicts that (S_i, G_i) does not contain a case-(1) left/right wedge. A similar contradiction happens if (s, g) has case-(2) right-wedges, in which (s^+, g) is not LR-visible. Thus, we see that no such left/right wedge exists. Next consider the other possibility of case-(2) left/right wedges in which (s^-, g) (Figure 3 (b.1)) or (s^+, g) (Figure 3 (d.1)) has an s-deadlock. If (s, g) has case-(2) left wedge in which (s^-, g) has an s-deadlock, then s must be at the beginning point of S_{ij}, because otherwise S_{ij} would intersect an $F1$ forbidden chain of that deadlock and would not be an admissible chain. But for such (s, g), (s^+, g) has a case-(1) left wedge which contradicts that (S_i, G_i) does not have a case-(1) left/right wedge. A similar contradiction happens if (s, g) has case-(2) right-wedges in which (s^+, g) has an s-deadlock. Thus, such left/right wedge does not exist either.

7.2. Reporting Vertex Pairs that Admit a (Straight) Counter-walk

We first consider the problem for counter-walkability and then that for straight counter-walkability. To report all vertex pairs that admit a counter-walk, we need to find all vertex pairs (s, g)'s such that each (s, g) is LR-visible, s and g are visible, and neither case (a) nor case (b) in Lemma 3 applies, by Lemma 4. First, to get (s, g)'s that satisfy LR-visibility condition, we compute all the LR-visible (S_i, G_i)'s like we do in the previous subsection in $O(n \log n)$ time. Second, to get (s, g)'s such that vertices s and g are visible, we use the algorithm by Hershberger[5] to compute all visible pairs of vertices in P, in $O(m)$ time, where m is the number of visible vertex pairs which is $O(n^2)$ but can be as small as $O(n)$. Actually, we can modify the algorithm in Ref. [5] without increasing the time complexity to get visible vertex pairs that do not have case (b) of Lemma 3, as follows. Consider a visible vertex pair (s, g) that has case (b.1) of Lemma 3 in which $s' \in L$ (see Figure 4 (b.1)). Let $\overline{ss'}$ be the cw-chord associated with s. Then, there must be a vertex $u \in CCW(s, g)$ that is the first reflex vertex counterclockwise from s such that u^+ is not visible from s. Since this u can be tested when computing the set V_s of vertices visible from s, we can remove from V_s the pair (s, p) for each $p \in CCW(u^+, s')$. Such (s, p) pairs have case (b.1) of Lemma 3 in which $s' \in L$, and are the only pairs containing s such that this case applies. Visible vertex pairs that have other cases of Lemma 3 (b) can be similarly removed.

Combining both, we can get the set V of all the vertex pairs (s, g)'s such that, for each pair, both visibility and LR-visibility conditions are met and case (b) of Lemma 3 does not apply. Now we try to remove from V the pairs to which case (a) of Lemma 3 applies. First, if there exist no two disjoint components $Comp(a)$

and $Comp(b)$ such that a and b are both ccw-chords or both cw-chords, we need not consider case (a) of Lemma 3 because it does not apply. Otherwise, following the definitions and description for polygon counter-walkability test in Section 6, we know that P is counter-walkable only if $C_a = \{a\}$ or $C_b = \{b\}$. So after computing C_a and C_b, if neither $C_a = \{a\}$ nor $C_b = \{b\}$, we report P is not counter-walkable. If $C_a = \{a\}$, in order for a pair to admit a counter-walk, one point in the pair must be on v_a. In this case, we remove the pairs from V that do not contain v_a. The other case when $C_b = \{b\}$ is similarly handled. If V is empty after the removal, we report P is not counter-walkable. Otherwise V is the set of all vertex pairs that admit a counter-walk. And thus, reporting all vertex pairs for counter-walkability can be done in $O(n \log n + m)$, where m is $O(n^2)$.

We now consider computing all vertex pairs, (s, g)'s, that admit a straight counter-walk. By Lemma 3, we need to find all vertex pairs (s, g)'s such that each (s, g) is LR-visible, s and g are visible, and none of the cases (a–d) in Lemma 3 applies. First, we test if P is straight counter-walkable (in Section 6) in $O(n \log n)$ time. If it is, then P has no two disjoint components, by Lemma 12. In this case, it is clear that we need not consider cases (a) and (c) in Lemma 3 because neither applies. Then, we can compute V like we do for the counter-walkability described above to get all the vertex pairs that satisfy both visibility and LR-visibility conditions, and do not have cases (b) and (d) of Lemma 3— removing vertex pairs to satisfy case (d) of Lemma 3 is similar to what we do for removing vertex pairs to satisfy case (b) of Lemma 3. Thus, reporting all vertex pairs that admit a straight counter-walk can be done in $O(n \log n + m)$, where m is $O(n^2)$.

8. Conclusion

In summary, we have proved

Theorem 1 *The polygon (straight) (counter-)walkability test can be done in $O(n \log n)$ time and linear space.*

Theorem 2 *Reporting all point pairs that admit a (straight) walk can be solved in $O(n \log n)$ time and linear space.*

Theorem 3 *Reporting all vertex pairs that admit a (straight) counter-walk can be solved in $O(n \log n + m)$ time, where m is $O(n^2)$.*

We mention that if a pair of two points admits a (counter-)walk, the minimum length walk of it can be computed by an algorithm in Ref. [6] which outputs "walk instructions" for the two guards in $O(n \log n + k)$, where $k = O(n^2)$ is the size of the output. For straight (counter-)walk, the walk instructions are of size $O(n)$ and can be computed in $O(n \log n)$ time[6] or in $O(n)$ time.[4]

There are some other related walkability problems for further study:

(1) Among all the pairs that admit a (straight) (counter-)walk, in which walk is the maximum distance between the two guards minimized[6]?

(2) Suppose there are $k \geq 3$ guards on the boundary of a simple polygon. Can they move in the same direction (either clockwise or counterclockwise on the

boundary) such that the whole boundary is traversed and the guards are always "visible"? Here, the visibility can be defined as either each guard is visible from all the other guards or each guard is visible from at least one of the other guards.

References

1. B. Chazelle, "Triangulating a simple polygon in linear time," *Proc. of the 31-st Annual IEEE Symposium on Foundation of Computer Science*, 1990, 220–229.
2. B. Chazelle and L. Guibas, "Visibility and intersection problems in plane geometry," *Discrete and Computational Geometry*, 4(6), 1989, 551–581.
3. 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, 209–233.
4. P. Heffernan, "An optimal algorithm for the two-guard problem," *Proc. of the Ninth ACM Symposium on Computational Geometry*, 1993, 348-358.
5. J. Hershberger, "An optimal visibility graph algorithm for triangulated simple polygons," *Algorithmica*, 4, 1989, 141–155.
6. C. Icking and R. Klein, "The two guards problem," *Int'l J. Comput. Geometry & Application,* 2,3 (Sept. 1992), 257-285.
7. Y. Ke, "Detecting the weak visibility of a simple polygon and related problems," Tech. Report, The Johns Hopkins University, 1987.
8. D. T. Lee, M. Sarrafzadeh and Y. F. Wu, "Minimum cuts for circular-arc graphs," *SIAM J. of Comput.*, 19(6), 1990, 1041–1050.
9. S. Suri, "Minimum link paths in polygons and related problems," Ph.D. thesis, The Johns Hopkins University, August 1987.

Appendix A: Correctness Proof of Lemmas 2.1–2.4

Here we prove the correctness of Lemma 3 (conditions for (s, g) admitting a straight counter-walk). Lemma 1, Lemma 2, and Lemma 4 can be proved in a similar way. For Lemma 1 (conditions for a walk) and Lemma 2 (conditions for a straight walk), the following perturbation scheme[4] helps to explain the conditions in addition to those given in the original lemma in Ref. [6] for the cases when s and/or g are on a reflex vertex.

> **v^+/v^- (Perturbation) Technique:** *If s is on a reflex vertex while g is not, we use the original lemma in Ref. [6] to test if (s^+, g) or (s^-, g) admits a (straight) walk. (s^+, g) or (s^-, g) admits a (straight) walk if and only if (s, g) admits a (straight) walk.*

We now describe the v^+/v^- method; its use allows one to approach the (straight) walk problem as if no potential vertex s or g were reflex.

Imagine that we are trying to determine (straight) walkability with respect to a reflex vertex s and a convex vertex g. Both guards start at s. Because s is reflex, both guards cannot begin moving at the same moment; if the guard on $L(R)$ moves first, it must continue until the other endpoint of the ccw- (cw-) chord before the guard on $R(L)$ begins to move. Replace s with two vertices s^+ and s^- as follows. The vertex s^+ lies on the edge of L which is adjacent to s, and s^+ is infinitessimally close to s; similarly s^- lies on the other adjacent edge of s. Note the following: there exists a (straight) walk from s to g if and only if there exists a (straight) walk from s^+ to g or there exists a (straight) walk from s^- to g. Therefore, for each reflex vertex v, create a pair of "replacement" vertices v^+ and v^-.

Define P^+ as the union of P with its replacement vertices. In the straight walk algorithms that follow, we will traverse P^+, looking for possible (straight) walkable pairs (s, g). We will consider only convex and replacement (not reflex) vertices as condidates for (straight) walkable pairs. The reflex vertices will also be examined in the algorithms, but only for the cw and ccw chords they produce. The algorithms' final step will be to call a pair (straight) walkable for reflex s and convex g if and only if (s^+, g) or (s^-, g) is (straight) walkable (similarly for g reflex or for s and g both reflex).

Note that while the (straight) walk algorithms use P^+, the (straight) counter walk algorithms use the original polygon P.

Now we consider Lemma 3. We first give the original lemma in Ref. [6] for the conditions of (s, g) admitting a straight counter-walk, where s or g does not lie on a reflex vertex.

Lemma A.1 (s, g) *admits a straight counter-walk if and only if s and g are visible, (s, g) is LR-visible, and none of the following cases applies (see Figure 4 (a) and (c)).*

(1) $\exists\, \overline{v_1 v_1'}, \overline{v_2 v_2'}$:

- *(a) both of them are ccw-chords or cw-chords,*
- *(b)* $v_1, v_2' \in L$, $v_2, v_1' \in R$,
- *(c)* $Comp(\overline{v_1 v_1'}) \cap Comp(\overline{v_2 v_2'}) = \emptyset$, *and*
- *(d)* ($v_1 \neq s$ *and* $v_2 \neq g$) *or* ($v_1 \neq g$ *and* $v_2 \neq s$).

(2) $\exists\, \overline{v_1 v_1'}, \overline{v_2 v_2'}$:

- *(a) one is a cw-chord and the other is a ccw-chord,*
- *(b) either* ($v_1, v_2 \in L$, $v_1', v_2' \in R$) *or* ($v_1, v_2 \in R$, $v_1', v_2' \in L$), *and*
- *(c)* $Comp(\overline{v_1 v_1'}) \cap Comp(\overline{v_2 v_2'}) = \emptyset$.

Proof. The necessity part is clear—if any one of the conditions is violated, (s, g) can not admit a straight counter-walk while still maintaining the visibility. The sufficiency part is proved by showing that walking instructions can be constructed if the conditions hold. First assume both visibility and LR-visibility of (s, g) are satisfied. We consider for the left guard on vertex $p \in L$ in a straight counter walk, the position of its partner (the right guard) on R. There are four cases.

case (1) If there is a ccw-chord $\overline{qq'}$ with $q \in R$ and with component not containing p, then p's partner must be on $CCW(q, g)$. Otherwise, the left guard will need backtracking when its partner goes past q later. This is true for all such $\overline{qq'}$'s.

case (2) If there is a cw-chord $\overline{qq'}$ with $q \in R$ and with component not containing p, then p's partner must be on $CW(q, s)$. Otherwise, the left guard had been backtracked because it has passed p when its partner was previously below q. This is true for all such $\overline{qq'}$'s.

case (3) If there is a ccw-chord $\overline{rr'}$ with $r \in L$ and with component not containing p, then p's partner must be on $CW(r', s)$. Otherwise, the right guard will need backtracking when the left guard goes past r later. Again, this is true for all such $\overline{rr'}$'s.

case (4) If there is a cw-chord $\overline{rr'}$ with $r \in L$ and with component not containing p, then p's partner must be on $CCW(r', g)$. Otherwise, the right guard had been backtracked because it has passed r' when its partner was previously at r. And it is true for all such $\overline{rr'}$'s.

It is clear that, for the left guard on $p \in L$, there are some "allowable intervals" for its partner (the right guard), outside which it is not possible to have a straight counter-walk. If (s, g) does not have any of the configurations shown in Figure 4 (a) and (c), i.e., cases (1) and (2) in Lemma A, it can be seen these allowable partner intervals for p have a common intersection and form an interval $CCW(lo(p), hi(p))$, denoted $[lo(p), hi(p)]$, on R. So we assume the condition that neither case (1), nor case (2) applies holds. Since the partner of a guard on p must be visible from p, we can restrict the partner interval to the portion that is visible from p. And, it can be shown this portion is indeed a nonempty and connected part $CCW(L(p), H(p))$ on R, which we denote as $[L(p), H(p)]$. Similarly, if we consider the right guard on a vertex $p \in R$ and its partner's allowable positions, we have for each vertex $p \in R$ a partner interval $CW(L(p), H(p))$, denoted $[L(p), H(p)]$, on L. It can be proved that for vertices v's on L, $L(v)$'s are ordered counterclockwise on R, the same sequence as v's on L. Also, for $p \in L$ and $q \in R$, $p \in [L(q), H(q)]$ if and only if $q \in [L(p), H(p)]$. According to these, we can assign a walk partner position on R for any v on L, and for vertices on R that have not been assigned a walk partner position, choose $L(q) \in L$ as its walk partner position. Let the vertices and partner positions on L be $a_0 = g, a_1, a_2, \ldots, a_k$ counterclockwise and let those on R be $a'_0 = s, a'_1, a'_2, \ldots, a'_k$ counterclockwise. If l and r represent the two guards on L and R respectively, a straight counter-walk instruction is that the guards proceed such that their walk segment $\overline{lr}$ rotates about the intersection point $\overline{a_i a'_i} \cap \overline{a_{i+1} a'_{i+1}}$ for $i = 0, 1, \ldots, k$. □

What needs to be modified for the lemma if we allow s and/or g to be on reflex vertices is given next. It is clear that both visibility and LR-visibility of (s, g) are still needed, so we assume they hold. Also, if s or g is neither v_1 nor v_2 as shown in cases Figure 4 (a) and (c), we assume neither case applies (because either of them does not allow a straight counter-walk, as can be seen from the above lemma and proof).

First suppose s is on a reflex vertex and let $t \in L$ be the first vertex clockwise from s. Following the arguments in the proof of the above lemma, we observe that, for each vertex $p \in L$, its partner's allowable interval on R is not affected, except possibly when p is t. This is because, if $p = t$, the portion on R which is visible from p (the allowable partner's interval for p must be in this visible portion) may degenerate to a single point s, as shown in Figure A.1, when the cw-chord $\overline{sq}$ has q not on R. Then, $p = t$ does not have an allowable partner's interval $[L(t), H(t)]$

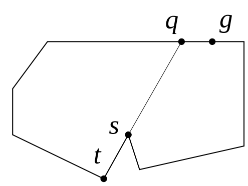

Fig. A.1. The portion on R visible from t is just s.

in case $[lo(t), hi(t)]$ does not have intersection with s. Consider in the proof of the above lemma, the four cases and the definition of $[lo(p), hi(p)]$ on R for $p \in L$. We see that if a chord $\overline{qq'}$ in case (1) does exist, then, when letting p be t such that the visible portion on R from t is just a single point s, the allowable partner's position $CCW(q, g)$ will not contain s. This is what we have as the first condition in (b.1) of Lemma 3. We also see that if a chord $\overline{rr'}$ in case (4) does exist, then, when letting p be t such that the visible portion on R from t is just a single point s, the allowable partner's position $CCW(r', g)$ will not contain s. This is what we have as the first condition in (d.1) of Lemma 3.

Now suppose s is on a reflex vertex and let $t \in R$ be the first vertex counterclockwise from s. Again, we observe that, for each vertex $p \in R$, its partner's allowable interval on L is not affected, except possibly when p is t. And, in case $p = t$, the portion on L visible from t may be a single point s. So, it is possible that $p = t$ does not have an allowable partner's interval $[L(t), H(t)]$ in case $[lo(t), hi(t)]$ does not have intersection with s. Such cases can be found similar to cases (1)–(4) in the above lemma, or by constructing configurations analogous to that for the first conditions of (b.1) and (d.1) of Lemma 3. These new cases are the second conditions of (b.2) and (d.2) in Lemma 3.

Next, if considering now when g is on a reflex vertex, we can symmetrically get four new cases that we put in (b.2) and (d.2) of Lemma 3.

This explains why we have conditions (b) and (d) of Lemma 3 in addition to the conditions of Lemma A if we allow s/g to be on a reflex vertex.