Partially Walking a Polygon

Franz Aurenhammer
Institute for Theoretical Computer Science, University of Technology, Graz, Austria
auren@igi.tugraz.at

Michael Steinkogler
Institute for Theoretical Computer Science, University of Technology, Graz, Austria
steinkogler@igi.tugraz.at

Rolf Klein
Universität Bonn, Institut für Informatik, Bonn, Germany
rolf.klein@uni-bonn.de

Abstract

Deciding two-guard walkability of an n-sided polygon is a well-understood problem. We study the following more general question: How far can two guards reach from a given source vertex while staying mutually visible, in the (more realistic) case that the polygon is not entirely walkable? There can be $\Theta(n)$ such maximal walks, and we show how to find all of them in $O(n \log n)$ time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Polygon, guard walk, visibility

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.60

Funding This work was supported by Project I 1836-N15, Austria Science Fund (FWF).

Acknowledgements We want to thank the anonymous referees for their detailed comments that have helped improving the presentation of this paper.

1 Introduction

We address the following structural question on polygons: How many adjacent ear triangles can be cut off from a polygon W, starting from a given vertex s? This question was originally motivated by optimizing so-called triangulation axes, a recently introduced skeletal structure for simple polygons [1]. An equivalent formulation of the problem, which is of interest in its own right, reads as follows: How far can two guards reach when they are to walk on W's boundary, starting from s in different directions and staying mutually visible?

Visibility problems of this kind have been studied already in the 1990s, where Icking and Klein [6] gave an $O(n \log n)$ time algorithm for deciding two-guard walkability of an n-sided polygon W, from a source vertex s to a target vertex t. A few years later, Tseng et al. [7] showed that one can find, within the same runtime, all vertex pairs (s, t) such that W is two-guard walkable from s to t. Their result was improved to optimal $O(n)$ time by Bhattacharya et al. [3]. The algorithm in [6] actually provides a walk for W in case of its existence but, on the other hand, only a negative message is returned in the (quite likely) case that the polygon is not entirely walkable.

The present paper elaborates on 'how far' in the latter case a polygon W is two-guard walkable – a natural question that has not been considered in the literature to the best of our knowledge. Such *maximal walks* are not unique, in general, which complicates matters. We present a strategy that finds, in $O(n \log n)$ time, all possible maximal walks that initiate

at a given source vertex s of W. A preliminary version of this paper appeared in [2]. For an account of related visibility questions on polygons, we refer to the survey article by Urrutia [8] on art gallery problems.

2 Preliminaries

We start with introducing the concepts and notations needed later in our considerations. Throughout, we let W denote a simple polygon in the plane with n vertices, one of them being tagged as a source vertex, s. For two points x and y on the boundary, ∂W, of W, we write $x < y$ if x is reached before y when walking on ∂W from s in clockwise (CW) direction. For a vertex p of W, p^+ denotes the CW successor vertex of p on ∂W. Similarly, p^- denotes the CW predecessor vertex of p on ∂W. When p is a reflex vertex (that is, a vertex where the interior angle in W is greater than π), then the two 'ray shooting points' for p in W can be defined, namely, $\mathrm{For}(p)$ as the first intersection point with ∂W of the ray from p^- through p, and $\mathrm{Back}(p)$ as the first intersection point with ∂W of the ray from p^+ through p; consult Figure 1.

According to the aforementioned relation between walks and triangulations, we are only interested in *discrete* and *straight* walks. That is, the guards when moving on ∂W directly 'jump' from a vertex to the respective neighboring vertex (only one guard is allowed to move at a time), and they never backtrack. A *walk in* W is now defined as a diagonal (l, r) of W, $l < r$, such that the first guard can move CW from s to l, and the second guard can move CCW from s to r, while staying visible to each other at each step. An obvious condition for W to be walkable till (l, r) is that the two boundary chains from s to l and to r, respectively (call them L and R), are *co-visible* in W. That is, each vertex on L is visible from some vertex on R, and each vertex on R is visible from some vertex on L.

To characterize walkability, we will need a few more concepts, first introduced in [6]. We say that W forms a *forward deadlock* at a pair (p, q) of its reflex vertices if we have

$$\mathrm{Back}(q) < p < q < \mathrm{For}(p).$$

Similarly, W forms a *backward deadlock* at (p, q) if

$$p < (\mathrm{For}(q), \mathrm{Back}(p)) < q.$$

Finally, W forms a *CW wedge* at (p, q), if $p < q$ and there exists no vertex x of W with

$$q < \mathrm{For}(q) < x < \mathrm{Back}(p).$$

(A *CCW wedge* is defined in a symmetric way.) See Figure 1 where these geometric concepts are illustrated.

It is not hard to see that the two guards cannot pass beyond deadlocks and wedges without losing visibility. This will be made specific in Section 4. Moreover, in the work [6] it has been shown that these obstacles to walkability are indeed the only ones. By adapting their result to our setting we get:

▶ **Theorem 1.** *Let (l, r), $l < r$, be a diagonal of W, and denote with Q the polygon bounded by (l, r) and the two chains L and R defined above. Then (l, r) is a walk in W iff the following three conditions are satisfied:*

(1) *L and R are co-visible in Q,*

(2) *Q neither forms a forward deadlock nor a backward deadlock (p, q) with $p \in L$ and $q \in R$, unless p or q is in $\{l, r\}$,*

(3) *Q forms no CW wedge on L, and no CCW wedge on R.*

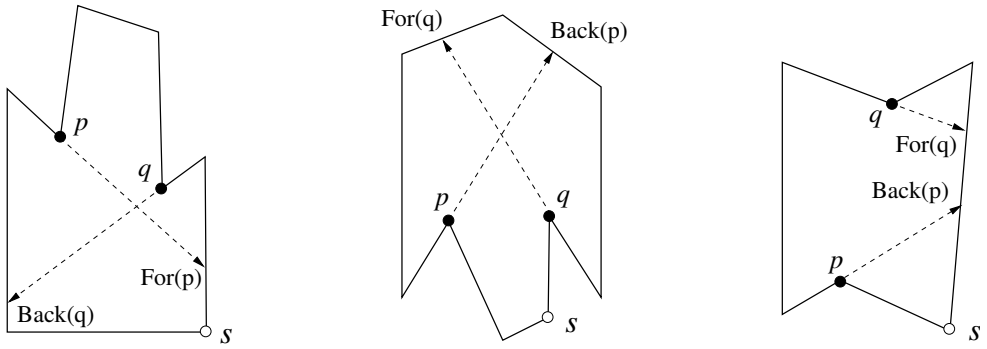


Figure 1 Forward deadlock (left), backward deadlock (middle), and CW wedge (right).

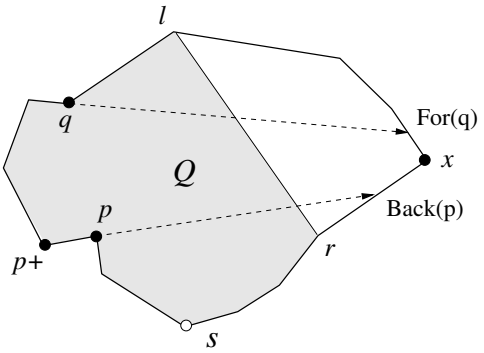


Figure 2 The line segment $\overline{lr}$ enables a CW wedge (p, q) in the shaded polygon Q.

3 Extremal walks and obstacles

Given a polygon W, our intention is to explore how far W is walkable from the source vertex s. That is, we want to find extremal positions for a diagonal (l, r) in W such that (l, r) is still a valid walk. A necessary (but not sufficient) condition is that (l, r) cannot be extended by a single guard move. More adequately, a walk (l, r) in W is termed *maximal* if there is no other walk (l', r') in W such that $l' \geq l$ and $r' \leq r$. For finding maximal walks, we will apply Theorem 1, but we have to do so with care since conditions (1) to (3) refer to a (yet unknown) polygon Q, rather than to the input polygon W as in [6].

To this end, for (1) we observe that the chains L and R are co-visible in Q iff they are co-visible in W: The line segment $\overline{lr}$ lies entirely within W, so the part of ∂W different from ∂Q does not obstruct the view within Q.

Concerning (2), we notice that forward deadlocks formed by Q do not depend on the shape of $\partial W \setminus \partial Q$, and thus trivially are also forward deadlocks formed by W. By contrast, for a backward deadlock (p, q) formed by Q, the points For(q) and Back(p) in Q may not be the same as in W. (Namely, if at least one of them lies on $\overline{lr}$). But since these points are larger than p and smaller than q, (p, q) is also a backward deadlock in W.

No such property holds for the wedges in (3), however. A wedge (p, q) formed by Q is not necessarily also formed by W: The segment $\overline{lr}$ can obstruct the view to vertices x on $\partial W \setminus \partial Q$ that prevent (p, q) from being a wedge in W. Figure 2 illustrates this situation.

Fortunately though, such 'induced' wedges cannot occur as long as the co-visibility condition is satisfied:

▶ **Observation 2.** *Assume that the diagonal $\overline{lr}$ of W induces a wedge in the polygon Q bounded by $\overline{lr}$ and the chains L and R. Then L and R are not co-visible.*

Proof. Without loss of generality, let the induced wedge, (p, q), be a CW wedge; see Figure 2 again. Then for the reflex vertex p we have $p < \text{Back}(p)$, and because (p, q) is induced by $\overline{lr}$ we also have $r > \text{Back}(p)$. But this implies that the vertex p^+ (which belongs to the chain L) is not visible from any point on the CCW chain from p to r. In particular, p^+ is not visible from any vertex on the chain R, which ranges from s to r. ◀

In summary, we can conclude that it suffices to consider the obstacles formed by the input polygon W, rather than the obstacles formed by Q.

For maximal walks, obstacles with extremal positions are relevant (in case of the presence of obstacles at all, which we will assume in the sequel). A *minimal CW wedge* on the chain L is a wedge (p, q) on L where the vertex q is smallest possible. For a *minimal CCW wedge* (p, q) on R, in turn, the vertex p has to be largest possible. Such extremal wedges need not be unique. A representative can be found in $O(n \log n)$ time, by a simple adaption of an algorithm given in [7], which finds all non-redundant wedges of a polygon. (We therefore do not elaborate on the details here.)

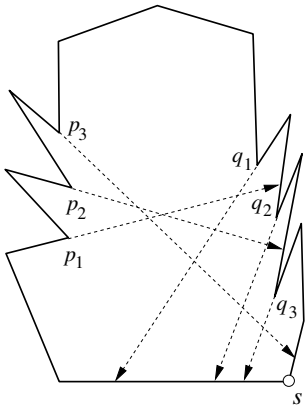
A deadlock (p, q) (either forward or backward) is called minimal if there is no other such deadlock (p', q') with $p' \leq p$ and $q' \geq q$. The *minimal backward deadlock* is unique, by the following property:

▶ **Observation 3.** *If (p, q) and (p', q') are two backward deadlocks with $p < p'$ and $q < q'$, then (p, q') is a backward deadlock as well.*

To find this minimal deadlock, we simply let p and q run through the reflex vertices of W, starting from s in CW and CCW direction, respectively, until the deadlock inequalities for p as well as for q are fulfilled at the same time. This can be done in $O(n)$ time, if W has been preprocessed accordingly in $O(n \log n)$ time using ray shooting; see Chazelle et al. [4].

Minimal forward deadlocks, on the other hand, are not unique in general. This is one of the reasons why maximal walks need not be unique. In fact, W can contain $\Theta(n)$ minimal forward deadlocks (p_i, q_i); see the figure below for $i = 1, 2, 3$. The following algorithm reports all of them. The points on ∂W relevant for this task are the reflex vertices p of W plus their ray shooting points $\text{For}(p)$. We assume their availability in cyclic order around W.

Algorithm MFD
for all relevant points x in CCW order from s **do**
  if $x = \text{For}(p)$ and $p < x$ **then**
    Insert p into a CW sorted list F
  else if x is a reflex vertex q **then**
    Search F for the smallest p with $\text{Back}(q) < p$
    if p exists and is unmarked **then**
      Mark p
      Report the forward deadlock (p, q)
    end if
  end if
  x = next relevant point
  Delete from F all vertices p with $p \geq x$



In a nutshell, the algorithm scans the boundary of W in counterclockwise direction, maintaining reflex vertices with forward rayshots on the scanned part of the boundary in a CW sorted list. When a reflex vertex is encountered, this list is used to search for forward deadlocks formed by the current vertex and vertices in the list.

▶ **Lemma 4.** *Algorithm* MFD *reports all minimal forward deadlocks* (p, q) *in* W*, and no other pair.*

Proof. Let (p, q) be a minimal forward deadlock. Then q is reflex and $q < \text{For}(p)$ holds. So the list F contains p when q is processed, by the CCW order of processing. Moreover, because (p, q) is minimal, p is the smallest vertex in F with $\text{Back}(q) < q$, and p is unmarked. Therefore, the algorithm will report (p, q). Conversely, assume that (p, q) gets reported. Then we know $\text{Back}(q) < p$, and because p is in F we know $q < \text{For}(p)$. Also, $p < q$ holds by the deletion criterion in the last line. Therefore (p, q) is a forward deadlock. Concerning minimality, observe first that there cannot be a forward deadlock (p', q') with $p' < p$ and $q' \geq q$. Otherwise, F contains p' when (p, q) is reported, because we have $q \leq q' < \text{For}(q')$. Because of $p' < p$, the algorithm would have reported (p', q) rather than (p, q), or nothing at all if p' is marked. There also is no forward deadlock (p', q') with $p' = p$ and $q' > q$. Otherwise, because of $q' > q$, (p', q') has been reported already. So $p' = p$ is marked, and (p, q) does not get reported. ◀

The algorithm can be implemented to run in $O(n \log n)$ time. It scans $O(n)$ relevant points, each being processed in constant time apart from the actions on F, which take $O(n \log n)$ time in total when a balanced search tree for F is used.

4 Constraints from obstacles

Minimal wedges and deadlocks, and also the required co-visibility, give rise to constraints on the vertices l and r for a maximal walk (l, r) in the polygon W. We will discuss the constraints on l in some detail. The situation for r is symmetric.

We have to distinguish between *absolute* and *conditional* constraints. Among the former is the list below. The first two constraints stem from the co-visibility of L and R, and have been taken from [6]. For the last two constraints, compare Figure 1.

(1) For each reflex vertex p with $p > \text{For}(p)$: $l \leq p$.
(2) For each reflex vertex p with $p < \text{Back}(p)$: $l \leq \text{Back}(p)$.
(3) For the minimal CW wedge (p, q) on L: $l \leq q$.
(4) For the minimal backward deadlock (p, q): $l \leq p$.

The conditional constraints read as follows:

(I) For each p in (1): If $r > p$ then $l < p^-$.
(II) For each p in (2): If $r > \text{Back}(p)$ then $l \leq p$.
(III) For (p, q) in (3): If $r > q$ then $l < q$.

For convenience, we subsume the absolute constraints (1) - (4) into a single one, $l \leq x$ (where x is the smallest right-hand side value), and turn it into a conditional constraint:

(IV) If $r \geq s$ then $l \leq x$.

Finally, the minimal forward deadlocks lead to absolute constraints which deserve special attention. Whereas in the case of a backward deadlock (p, q), neither guard can walk beyond these vertices, we have the following observation for the avoidance of a forward deadlock:

▶ **Observation 5.** *To avoid the forward deadlock* (p, q)*, only one of the bounds* $l \leq p$ *and* $r \geq q$ *needs to hold.*

Assume now that k minimal forward deadlocks $(p_1, q_1), \ldots, (p_k, q_k)$ exist, and let the vertices p_i be sorted in CW order.

▶ **Lemma 6.** *Each of the following $k+1$ pairs of bounds for (l, r) avoids all minimal forward deadlocks:* $(p_1, s), (p_2, q_1), \ldots, (p_k, q_{k-1}), (s^-, q_k)$.

Proof. By minimality of the considered deadlocks, we know that the vertices q_i will be sorted in CW order as well. So, for each index $i \geq 2$, Observation 5 tells us that the constraint $l \leq p_i$ avoids the deadlocks $(p_i, q_i), \ldots, (p_k, q_k)$, and the constraint $r \geq q_{i-1}$ avoids the remaining deadlocks $(p_1, q_1), \ldots, (p_{i-1}, q_{i-1})$. Moreover, the constraint $l \leq p_1$ suffices to avoid all k deadlocks, and $r \geq s$ is trivially fulfilled. The same is true for $r \geq q_k$ and $l \leq s^-$, respectively. ◀

In summary, there are $O(n)$ constraints in total, which can be identified in $O(n \log n)$ time by the results in Section 3.

5 Computing all maximal walks

Section 4 tells us that the goal is to fulfill the constraints in (I) - (IV) simultaneously, though for each of the bounding pairs in Lemma 6 separately. This gives all possible maximal walks – granted the visibility of the reported vertex pairs. But let us come back to the issue of visibility later in this section.

For a *fixed* bounding pair (a, b), the constraint satisfaction problem can be transformed into the following standard form: For two variables l and r, with absolute bounds a and b, respectively, we have two sets of conditional constraints: Namely, a set C_L containing constraints for l, of the form

$$r \geq y_i \implies l \leq x_i$$

and a set C_R containing constraints for r, of the form

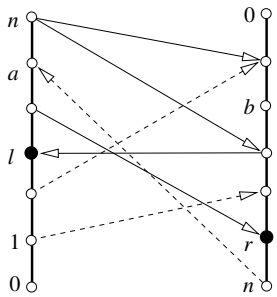
$$l \leq x_j \implies r \geq y_j \ .$$

We may assume that all x-values and y-values are in $\{0, 1, \ldots, n\}$. That is, the vertices $w_0, w_1, \ldots, w_n$ of W, $w_0 = w_n = s$, are identified with their indices. This is no loss of generality, because only their relative positions (rather than the geometric positions) on ∂W matter. We want to compute the (unique) maximal pair (l, r) such that

$$l \leq a, \ r \geq b, \text{ and all constraints } c \in C_L \cup C_R \text{ are fulfilled.}$$

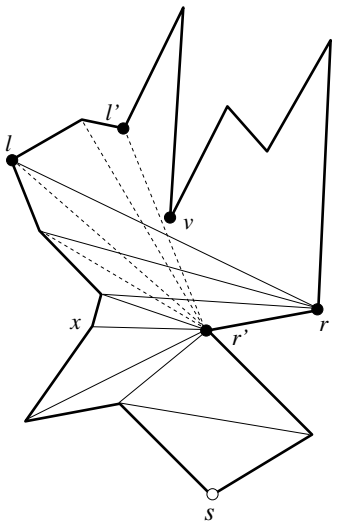
We say that a constraint $c_i \in C_L$ is *active* at a value r if $r \geq y_i$ holds. Similarly, a constraint $c_j \in C_R$ is *active* at l if we have $l \leq x_j$. The constraint fulfilling algorithm, CFF, now simply alternates in scanning through the sorted sets C_L and C_R (in ascending order of y_i-values, and in descending order of x_j-values, respectively), and adjusts the values of l and r according to the constraints that become active. In the figure below, active/inactive constraints are indicated with full/dashed arrows.

Algorithm CFF(a, b, C_L, C_R)
$l = a$, $r = b$
repeat
$x = \min\{x_i \mid c_i \in C_L \text{ is active at } r\}$
$l = \min\{l, x\}$
$y = \max\{y_j \mid c_j \in C_R \text{ is active at } l\}$
$r = \max\{r, y\}$
until $r = y$ or $r = b$
Return the pair (l, r)



Suppose that a function VIS(l, r) is available which returns the smallest vertex $r' \geq r$ such that (l, r') is visible in the polygon W. (That is, $\overline{lr'}$ is the first possible diagonal of W that emanates from vertex l. If r' does not exist then $n + 1$ is returned.) We now present an algorithm that uses CFF and VIS as subroutines, and is capable of computing, in $O(n \log n)$ time, all maximal walks that exist in W. Let $P = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ be the given set of bounding pairs. We assume that $a_1, \ldots, a_m$ (and thus $b_1, \ldots, b_m$) are in increasing order. In the polygon below, (l, r) and (l', r') are the two possible maximal walks.

Algorithm MAXWALKS(P, C_L, C_R)
$l = a_m$, $r = b_1$
$r_{\text{rep}} = n + 1$
while $l \geq 0$ and $r < r_{\text{rep}}$ **do**
$(l, r) = \text{CFF}(l, r, C_L, C_R)$
$i = \min\{\lambda \mid a_\lambda \geq l\}$
$r_{\text{cand}} = \max\{b_i, r\}$
$r_{\text{vis}} = \text{VIS}(l, r_{\text{cand}})$
$y = \max\{\varrho \mid \text{all } c \in C_L \text{ active at } \varrho \text{ admit } l\}$
if $r_{\text{vis}} \leq \min\{n, y\}$ and $r_{\text{vis}} < r_{\text{rep}}$ **then**
Report (l, r_{vis})
$r_{\text{rep}} = r_{\text{vis}}$
end if
$l = l - 1$
end while



Before providing a proof of correctness, we give a short explanation of this algorithm. All the bounding pairs (a_i, b_i) need to fulfill the constraints that are active there, so the algorithm starts by fulfilling the constraints for the vertex pair (a_m, b_1), as these constraints have to be fulfilled in any case. Then the boundary chain of W from a_m 'down to' s is scanned in CCW direction, while fulfilling all constraints on both chains. After each constraint fulfillment it is checked whether a candidate vertex pair lies 'below' a bounding pair (a_i, b_i), while also ensuring visibility within W and maximality among walks.

▶ **Lemma 7.** *Algorithm* MAXWALKS *is correct.*

Proof. The value of r changes only when Algorithm CFF is called, and thus r cannot decrease. The first call of CFF is with the bounding pair (a_m, b_1), and the subsequent calls are with (l, r) for $l < a_m$. As soon as we have $r > b_1$, some constraint in C_R is responsible for this. So putting the bound r for the next call means no additional restriction. This implies that, for all l, we have the equality $\mathrm{CFF}(l, r, C_L, C_R) = \mathrm{CFF}(l, b_1, C_L, C_R)$.

We now look at one iteration of the while loop, under the assumption that Algorithm MAXWALKS worked correctly so far. That is, all maximal walks (l', r') with $l' \geq l$ have been reported, and no other walks. Let l_{old} be the value of l before the iteration. Then $(l, r) = \mathrm{CFF}(l_{\mathrm{old}} - 1, b_1, C_L, C_R)$ holds by the former equality. So we have $(l, r) = \mathrm{CFF}(l', b_1, C_L, C_R)$ for $l_{\mathrm{old}} > l' > l$, implying that there is no walk (l', r') for these l'-values.

There also is no walk (l, r') with $r' < r_{\mathrm{cand}}$, because the bounding pair (a_i, b_i) as well as the constraints in C_R need to be respected. Concerning r_{vis}, if $r_{\mathrm{vis}} > n$ then no pair (l, r') with $r' \geq r_{\mathrm{cand}}$ is visible, and thus no such pair can be a walk. Further, if $r_{\mathrm{vis}} > y$ then some constraint in C_L is active at r_{vis} but does not admit l, so (l, r_{vis}) is not a walk either. On the other hand, if $r_{\mathrm{vis}} \leq \min\{n, y\}$ then (l, r_{vis}) is a walk, because the pair is visible and fulfills all the constraints. The pair gets reported unless $r_{\mathrm{vis}} \geq r_{\mathrm{rep}}$, in which case (l, r_{vis}) is not maximal because a larger pair has been reported already. ◀

Turning to runtime considerations now, we can make the following observations. CFF can be implemented such that the bounding pair of the last call is remembered. This way each constraint in $C_L \cup C_R$ is handled only once: If a call has been with (l, r), the next call will be with (l', r') where $l' < l$ (and thus $r' \geq r$). Thus only $O(n)$ time is spent in total for all calls to CFF from Algorithm MAXWALKS.

Computing the thresholds y in MAXWALKS can also be done in total $O(n)$ time. We remember the previous value of y, and scan down from this value as long as all active constraints of C_L are fulfilled by l. The first violating constraint then gives the new value for y.

The function VIS can be performed in logarithmic time using the techniques in Guibas and Hershberger [5], in a way similar as already done in Icking and Klein [6]: Basically, finding the desired vertex r_{vis} can be reduced to finding the first vertex on a shortest path between two polygon vertices. Clearly, the while loop is executed only $O(n)$ times (because the value of l is decremented in each iteration), which gives a runtime of $O(n \log n)$ for this part, and thus for Algorithm MAXWALKS overall.

We now can conclude the main result of this paper:

▶ **Theorem 8.** *Let W be a simple polygon with n vertices. For a given vertex s of W, there can be $\Theta(n)$ maximal two-guard walks in W starting from s, and these walks can be computed in $O(n \log n)$ time.*

6 Concluding remarks

A few comments related to the results in this paper are in order.

The polygon example in Algorithm MAXWALKS shows that maximal walks may differ in (combinatorial) length. The walk (l, r) involves 6 steps by the left guard and 3 steps by the right guard, so 9 steps in total, whereas the walk (l', r') involves 8 steps by the left guard and 2 steps by the right guard, and thus allows one more step in total.

The same example also reveals that minimum forward deadlocks are not the only reason why maximal walks are not unique: The reason why there are two walks for the shown

polygon is the vertex v, which can be 'approached' by the (mutually visible) guards in two different ways.

In Section 3 we have seen that minimum forward deadlocks can lead to $\Omega(n)$ different maximal walks. On the other hand, the number of maximal walks trivially cannot exceed n, because no two of them can have the same l-vertex, or the same r-vertex, by maximality.

Algorithm MAXWALKS provides each maximal walk in the form of a target pair (l, r), but the algorithm does not specify the way the two guards actually move on ∂W. Such a movement can be computed in $O(n)$ additional time: Since we know that the subpolygon Q of W defined by s and (l, r) is entirely walkable, we can simply apply the algorithm in [6] to the polygon Q (which has already been preprocessed with W).

Notice, however, that a fixed target pair (l, r) may still leave the guards different ways to perform the walk. Different ways to triangulate W from s to (l, r) then result. For example, in the polygon example in Algorithm MAXWALKS, it would be possible to include one more diagonal with endpoint r into the solid-line trianguation, namely, the diagonal $\overline{rx}$. The dual of any such triangulation has to be a path, though, as the triangulation is constructed by repeatedly cutting off adjacent ear triangles, one triangle per guard step.

References

1 Wolfgang Aigner, Franz Aurenhammer, and Bert Jüttler. On triangulation axes of polygons. *Information Processing Letters*, 115(1):45–51, 2015.

2 Franz Aurenhammer, Michael Steinkogler, and Rolf Klein. Maximal two-guard walks in a polygon. In *34nd European Workshop on Computational Geometry, EuroCG 2018*, pages 17–22, 2018.

3 Binay Bhattacharya, Asish Mukhopadhyay, and Giri Narasimhan. Optimal algorithms for two-guard walkability of simple polygons. In *Workshop on Algorithms and Data Structures, WADS 2001. Lecture Notes in Computer Science*, volume 2125, pages 438–449. Springer, 2001.

4 Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

5 Leonidas J Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

6 Christian Icking and Rolf Klein. The two guards problem. *International Journal of Computational Geometry & Applications*, 2(03):257–285, 1992.

7 L.H. Tseng, Paul Heffernan, and Der-Tsai Lee. Two-guard walkability of simple polygons. *International Journal of Computational Geometry & Applications*, 8(01):85–116, 1998.

8 Jorge Urrutia. Art gallery and illumination problems. In *Handbook of Computational Geometry*, pages 973–1027. Elsevier, 2000.