Maximal Two-Guard Walks in a Polygon

Franz Aurenhammer[1], Michael Steinkogler[2], and Rolf Klein[3]

1 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria,** auren@igi.tugraz.at
2 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria,** steinkogler@igi.tugraz.at
3 **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-solved problem. We study the related question of how far two guards can reach from a given source vertex, 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.

1 Introduction

We address a structural question on polygons, which is motivated by optimizing so-called triangulation axes [1], but is also interesting in its own right: How many adjacent 'ear triangles' can be cut off from a polygon W, starting from a given vertex s? Equivalent is the following question: How far can two guards reach when they are to walk on W's boundary, starting from s in different directions and keeping mutually visible?

Visibility problems of this kind have been studied already in the 1990s, where Icking and Klein [5] 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. [2]. The algorithm in [5] 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 note 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 more detailed description of the results is given in [6].

2 Preliminaries

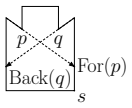
Throughout, we let W denote a simple polygon in the plane with n vertices, one of them being tagged as a source vertex, s. For two points x and y on the boundary, ∂W, of W, we write $x < y$ if x is reached before y when walking on ∂W from s in clockwise (CW) direction. For a vertex p of W, p^+ (repectively, p^-) is the CW successor (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, For(p) as the first intersection point with ∂W in the direction from p^- to p, and Back(p) as the first intersection point with ∂W in the direction from p^+ to p.

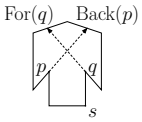
According to the aforementioned relation between walks and triangulations, we are only interested in *discrete* and *straight* walks. That is, the guards when moving on ∂W directly

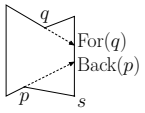
34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

'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 keeping 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 vice versa.

To characterize walkability, we will need a few more concepts, first introduced in [5]. We say that at a pair (p, q) of its vertices, W forms a:

- *Forward deadlock* if
$$\text{Back}(q) < p < q < \text{For}(p).$$


- *Backward deadlock* if
$$p < (\text{For}(q), \text{Back}(p)) < q.$$


- *CW wegde* if $p < q$ and there exists no vertex x of W with
$$q < \text{For}(q) < x < \text{Back}(p).$$


(A *CCW wegde* is defined in a symmetric way.)

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 [5] it has been shown that these obstacles to walkability are indeed the only ones. By adapting their result to our setting we get:

▶ **Theorem 2.1.** *Let (l, r), $l < r$, be a diagonal of W, and denote with Q the polygon bounded by (l, r) and the chains L and R. Then (l, r) is a walk in W iff (1) L and R are co-visible in Q, (2) Q forms no forward and no backward deadlock, (3) Q forms no CW wedge on L, and no CCW wedge on R.*

3 Extremal walks and obstacles

A walk (l, r) in W is termed *maximal* if it cannot be extended by a single guard move. More precisely, neither (l^+, r) nor (l, r^-) is a walk in W. For finding maximal walks, we will apply Theorem 2.1, 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 [5].

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 $\text{For}(q)$ and $\text{Back}(p)$ 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 neccessarily 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. Fortunately though, such 'induced' wedges cannot occur once the co-visibility condition is satisfied; see [6].

In conclusion, it suffices to consider obstacles formed by W rather than 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 adapting an algorithm in [7].

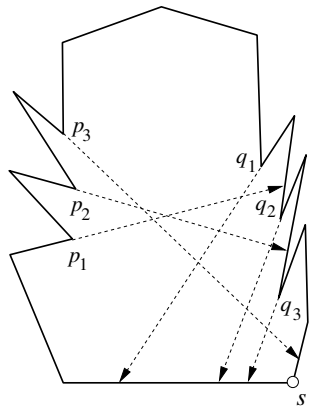
A deadlock (p, q) (either forward or backward) is called minimal if there is no other such deadlock (p', q') with $p' \leq p$ and $q' \geq q$. The *minimal backward deadlock* is unique, by the following property: If (p, q) and (p', q') are two backward deadlocks with $p < p'$ and $q < q'$, then (p, q') 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, respecticely, 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. [3].

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 For(p). We assume their availability in cyclic order around W.

Algorithm MFD

Trace relevant points x in CCW order from s:
if $x =$ For(p) and $p < x$ **then**
  Put p to a CW sorted list F
else if x is a reflex vertex q **then**
  Search F for the smallest p with 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 $p \geq x$



The proof of correctness of Algorithm MFD is omitted due to lack of space. The algorithm can be implemented 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 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; see [5].

(1) For each reflex p with $p >$ For(p): $l \leq p$.

(2) For each reflex p with $p <$ Back(p): $l \leq$ 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 $r >$ 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 (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 p_i's be sorted in CW order.

▶ **Lemma 4.1.** *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 q_i's have to be sorted in CW order. So, for each index $i \geq 2$, the observation above 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 found 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 4.1 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 bounds a and b, respectively, there are two sets C_L and C_R of conditional constraints, of the form

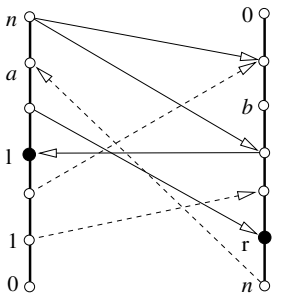
$$r \geq y_i \implies l \leq x_i \quad \text{and} \quad l \leq x_j \implies r \geq y_j$$

respectively, with all values 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.) We want to compute the maximal pair (l, r) such that

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

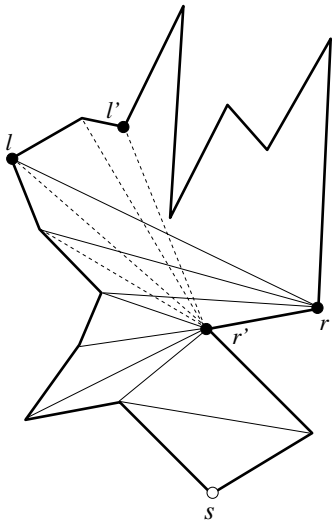
We say that $c_i \in C_L$ is *active* at a value r if $r \geq y_i$ holds. Similarly, $c_j \in C_R$ is *active* at l if we have $l \leq x_j$. The constraint fulfilling algorithm now simply alternates in scanning through the sorted sets C_L and C_R (in ascending order of y_i-values, and in descending order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

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 first vertex $r' \geq r$ such that (l, r') is visible in the polygon W. (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 $a_1, \ldots, a_m$ (and thus $b_1, \ldots, b_m$) 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



▶ **Lemma 5.1.** *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 $\text{CFF}(l, r, C_L, C_R) = \text{CFF}(l, b_1, C_L, C_R)$.

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 $(l, r) = \text{CFF}(l_{\text{old}} - 1, b_1, C_L, C_R)$ holds by the former equality. So we have $(l, r) = \text{CFF}(l', b_1, C_L, C_R)$ for $l_{\text{old}} > l' > l$, implying that there is no walk (l', r') for these l'-values.

There also is no walk (l, r') with $r' < r_{\text{cand}}$, because the bounding pair (a_i, b_i) as well as the constraints in C_R need to be respected. Concerning r_{vis}, if $r_{\text{vis}} > n$ then no pair (l, r')

with $r' \geq r_{\text{cand}}$ is visible, and thus no such pair can be a walk. Further, if $r_{\text{vis}} > y$ 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 $r_{\text{vis}} \leq \min\{n, y\}$ then (l, r_{vis}) is a walk, because the pair is visible and fulfills all the constraints. The pair gets reported unless $r_{\text{vis}} \geq r_{\text{rep}}$, in which case (l, r_{vis}) is not maximal because a larger pair has been reported already. ◀

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 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 [4]. Clearly, the while loop is executed only $O(n)$ times, which gives a runtime of $O(n \log n)$ for this part, and thus for Algorithm MAXWALKS overall.

6 Concluding remarks

We conclude the paper with a few brief comments.

The polygon example above shows that maximal walks differ in (combinatorial) length, in general. It is also revealed that minimum forward deadlocks are not the only reason why maximal walks are not unique.

The number of maximal walks is trivially bounded by n, because no two of them can have the same l-vertex (or the same r-vertex).

Algorithm MAXWALKS provides each maximal walk in the form of a target pair (l, r), but not the way how the guards actually move on ∂W. Such a movement can be computed in $O(n)$ time, by applying the algorithm in [5] to the (already preprocessed) subpolygon of W defined by s and (l, r). Notice, however, that a fixed target pair (l, r) may leave to the guards different ways to perform the walk. Different ways to triangulate W from s to (l, r) then result, though the dual of any such triangulation has to be a path.

References

1 W. Aigner, F. Aurenhammer, and B. Jüttler. On triangulation axes of polygons. *Information Processing Letters* 115 (2015), 45-51.

2 B. Bhattacharya, A. Mukhopadhyay, and G. Narasimhan. Optimal algorithms for two-guard walkability of simple polygons. *Proc. Workshop on Algorithms and Data Structures, WADS 2001*. Springer Lecture Notes in Computer Science 2125, 438–449.

3 B. Chazelle, H. Edelsbrunner, M. Grigni, L.J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica* 12 (1994), 54–68.

4 L.J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Computer and System Sciences* 39 (1989), 126–152.

5 C. Icking and R. Klein. The two guards problem. *Int'l J. Computational Geometry & Applications* 2 (1992), 126–152.

6 M. Steinkogler. Maximal walks in polygons. Master Thesis, Institute for Theoretical Computer Science, University of Technology, Graz, Austria, 2017.

7 L.H. Tseng, P. Heffernan, and D.T. Lee. Two-guard walkability of simple polygons. *Int'l J. Computational Geometry & Applications* 8 (1998), 85–116.