Approximating the Sweepwidth of Polygons with Holes

Dorian Rudolph[1]

1 **Paderborn University**
`dorian@mail.upb.de`

Abstract

Consider a contaminated polygon P with the goal of decontaminating P by sweeping it with barrier curves, where the contaminant spreads instantly along all paths not blocked by a barrier. The maximum length of curves in a sweep needed to decontaminate P is defined as its *sweepwidth*. This problem was introduced Karaivanov et. al (2014) [8] who also proved that computing the sweepwidth of even simple, orthogonal polygons is $\mathcal{NP}$-hard. Therefore, we propose a polynomial time $O(\log n)$-approximation algorithm for the sweepwidth of n-vertex polygons with holes. We accomplish this by rasterizing the polygon into a grid which allows a reduction to the well known *node search problem* on graphs. In order to obtain a polynomially sized rasterization, we first apply a *compression* technique to the polygon.

1 Introduction

Karaivanov et al. [8] introduced the problem of decontaminating an initially contaminated planar region by sweeping it with moving barriers in the form of curves while the contaminant instantly spreads along any path that is not blocked by a barrier. It can be seen as an extension of the *node search problem* introduced by Kirousis and Papadimitriou [9] where a graph has to be decontaminated with as few searchers (or pebbles) as possible. Next, we will formally define these problems.

Sweepwidth [8]. Let P be a closed n-vertex polygon with holes and no intersecting edges. $P \subset \mathbb{R}^2$ shall also denote the set of points on the polygon including its boundary. Every point of P is either *contaminated* or *decontaminated*. We sweep P using a set of moving *barriers* $b\colon [0,1] \to 2^P$, where the barriers at any time $t \in [0,1]$ consist of the points $b(t)$. All points in $b(t)$ become decontaminated. Initially, all points of P (except $b(0)$) are contaminated. A decontaminated point q becomes *recontaminated* at time t if there exists a path from a contaminated point p to q not intersecting $b(t)$. We say b decontaminates P if all points in P are decontaminated at time 1 (see [8] for a more formal definition). We restrict barriers to piecewise continuously differentiable *barrier curves* with a finite number of pieces and barriers. This allows us to describe a sweep by a function $b\colon [0,1]^2 \to P$ such that $b(s,t)$ is piecewise continuous in both curve parameter s and time t, and for any t, $b(\cdot,t)$, is piecewise continuously differentiable. The function $s \mapsto b(s,t)$ describes the barriers at time t. We measure the total length of barriers at time t as the sum of the arc lengths of all pieces of $b(\cdot,t)$. The *bottleneck length* of b is defined as the supremum over time of the sum of the lengths of barriers in $b(\cdot,t)$. An exemplary sweep is depicted in Fig. 1. We refer to the minimum bottleneck length of all decontamination sweeps of P as *sweepwidth* of P, denoted $sw(P)$. Karaivanov et al. show that each decontamination sweep can be transformed into a *canonical* sweep without increasing its bottleneck length, i. e., a sweep where all barriers consist of one or two straight line segments connecting two points on the boundary of P.

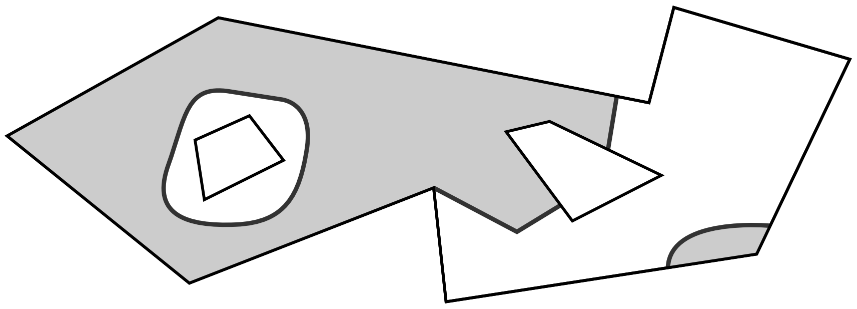

Figure 1 Incomplete sweep of a polygon with holes. The contaminated area is depicted as light gray, barriers as dark lines.

Node search [9]. Let $G = (V, E)$ be an undirected graph. All edges $e \in E$ are initially considered contaminated. To decontaminate G, we can *place* and *remove searchers* on nodes. We refer to this sequence of moves as *node-search strategy.* Nodes with a searcher are considered *guarded.* An edge $e = (v, w) \in E$ is *decontaminated* if v and w are guarded. e is recontaminated if there exists a path W of unguarded nodes from u or v to a node incident to a contaminated edge. Let the *node-search number* of G, $ns(G)$, be the minimum number of searchers needed to decontaminate all edges of G.

Related Work. For the polygon decontamination problem defined above, Karaivanov et al. [8] construct optimal sweeps for rather simple classes of polygons and prove that computing sweepwidth is $\mathcal{NP}$-hard for simple, orthogonal polygons. To the best of our knowledge, no approximation algorithms for the sweepwidth of general polygons are known. That problem is generalized by Markov et al. [12] to the *directed sweepwidth* where barriers must start and end on predefined parts of the boundary. They also give a rather involved lower bound for the sweepwidth based on the sweepwidth of three non-intersecting subshapes. Various other search problems for polygons have been analyzed in literature, including observing the entire polygon (*art gallery problem*) [13], or agents having to catch [10] or spot [1] an intruder. Another related problem is sweeping terrains with aerial vehicles [3]. If restricted to a single curve and simple polygons, sweepwidth is equivalent to the *elastic ring-width* [14], solved in time $O(n^2 \log n)$ [6]. Hence, ring-width constitutes an upper bound on sweepwidth. However, that bound can be arbitrarily bad, e. g., for an arbitrarily narrow T-shaped polygon.

Regarding different search problems on graphs, we refer the reader to the survey [4].

Our Contribution. We develop a polynomial time $O(\log n)$-approximation algorithm for the sweepwidth of n-vertex polygons with holes. We do this by rasterizing the polygon as a grid graph and computing its node-search number. It will follow that $O(1)$-approximation of sweepwidth is at most as hard as $O(1)$-approximation of the node-search number.

2 Algorithm

We will construct a sweep of P with a bottleneck length of $O(\log n \cdot sw(P))$ by rasterizing P using hexagonal cells (each cell is a closed set). Their adjacency graph G_P is an induced subgraph of the infinite triangular lattice graph [1]. First, we will argue $sw(P) = \Theta(ns(G_P))$ (using an appropriate scale). Afterwards, we describe a *compression* technique to construct a polygon P' with $sw(P) = \Theta(sw(P'))$ and polynomially sized $G_{P'}$. We can compute an $O(\log n)$-approximation of $ns(G_{P'})$ in polynomial time, also yielding an $O(\log n)$-approximation of $sw(P)$.

2.1 Rasterization

Cell size. Let R_P be the diameter of the largest inscribed circle in P. We define the *size* of cells, i. e., the length of edges in the lattice, as $r_P := R_P/n$. As $sw(P) \geq R_P$ [8], we can guard $O(n)$ cells throughout the entire sweep using curves of length $O(sw(P))$. To compute R_P, we construct the Voronoi diagram of the edges of P in time $O(n \log n)$ with Fortune's algorithm [5]. One of the Voronoi nodes must be the center of the largest inscribed circle.

Cell categories. We can now describe the rasterization process. There are different types of cells that will need to be treated differently by the algorithm. To that end, we introduce categories for cells intersecting P. Each cell not completely outside P belongs to exactly one of the following categories:

(a) *Blocked cell*: cell that either contains a vertex of P, or is completely inside P and is adjacent to two cells on opposing sides that are both intersected by at least one edge (dark gray in Fig. 2).
(b) *Full cell*: cell fully inside P that is not a blocked cell (white).
(c) *Empty cell*: any other cell (light gray).

All cells between the outermost blocked cells belonging to the same edge pair that do not contain a vertex shall be *redefined* as *empty cells* (see striped cells in Fig. 3).

If m cells intersect P, then we can easily compute the categories in time polynomial in m. Let G_P be the graph induced by full cells. We will argue that there are $O(n)$ blocked cells and that placing barriers around them separates cells belonging to different connected components of G_P.

▶ **Lemma 1.** *There are $O(n)$ blocked cells.*

▶ **Theorem 2.** *It holds that $sw(P) = \Theta(ns(G_P) \cdot r_P)$.*

Proof sketch. In order to prove $sw(P) = O(ns(G_P) \cdot r_P)$, we construct a sweep by first placing barriers at a distance of two around all blocked cells and decontaminate the inside of these barriers. One can show that this separates cells belonging to different connected components of G_P. Hence, we can decontaminate the resulting regions of P separately. Regions without full cells can easily be decontaminated with a curve of length $O(r_P)$. For each component of G_P, consider an optimal node-search strategy. Whenever a searcher is on a node of G_P, we place barriers at a distance of 2 around the corresponding cell. This can be shown to decontaminate regions of full cells including adjacent empty cells.

G_P has a connected component C such that $ns(G_P) = ns(C) = \Omega(n)$, as separate connected components can be decontaminated one after another. $ns(C) = \Omega(n)$ follows from the fact that it takes $\Omega(n)$ searchers to decontaminate an $\Omega(n) \times \Omega(n)$ parallelogram of cells, which is contained in the largest inscribed circle. Let $P_C \subset P$ be the polygon of cells in C. It is straightforward to prove $sw(P_C) = \Theta(ns(P_C) \cdot r_P)$. $sw(P) = \Omega(ns(G_P) \cdot r_P)$ follows. ◀

[1] We use hexagonal cells since then G_P is planar, which would not be the case for square cells due to diagonal adjacencies.

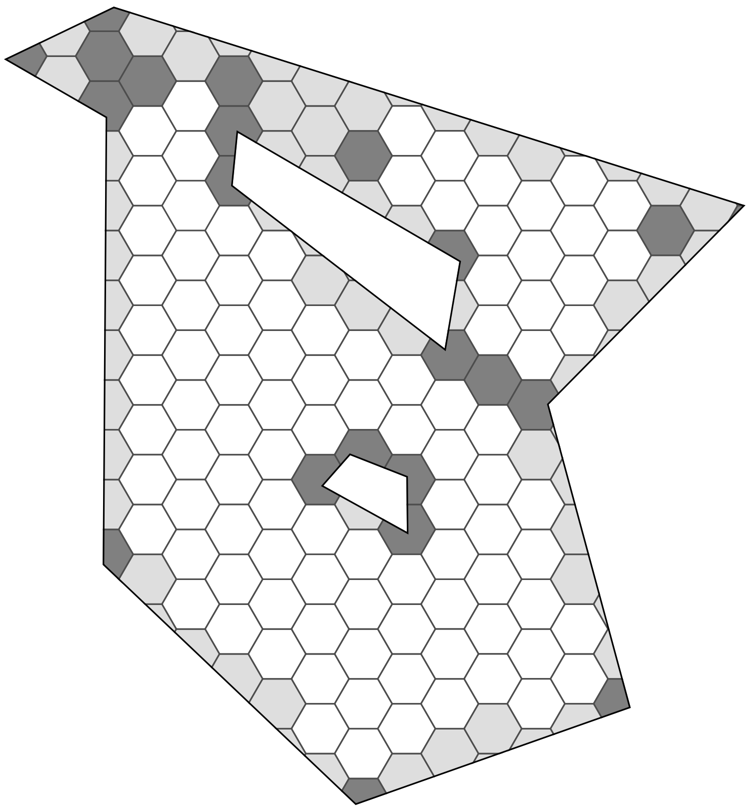

Figure 2 Polygon rasterized into a hexagonal grid. Blocked cells are depicted dark gray, empty cells light gray, and full cells white.

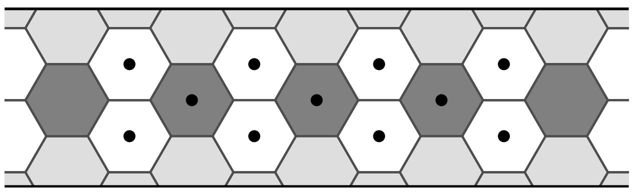

Figure 3 Cells between outermost blocked cells of an edge pair are redefined as empty (marked with dots).

Since G_P is planar, we can compute an $O(1)$-approximation of its *treewidth* $tw(G_P)$ in time $O(m \log^4 m)$ [7] if G_P has m nodes. As $ns(G_P) = \Omega(tw(G_P))$ and $ns(G_P) = O(tw(G_P) \log m)$ [2], we have an $O(\log m)$-approximation for $ns(G_P)$.

2.2 Polygon Compression

G_P may still contain arbitrarily many cells if P contains long, narrow sections. Thus, we will *compress* intervals along the x- and y-axes such that the resulting polygon P' can be rasterized using a polynomial number of cells. In the following, we will describe how to compress an interval along the x-axis where P has no vertices. We only consider maximal such intervals with a length greater than $(n+6)R_P$ and compress them down to that length, thereby bounding the distance between vertices. Compressed intervals will not overlap.

Let $I' := [x'_0, x'_1] \subset \mathbb{R}$, $x'_1 - x'_0 > (n+6)R_P$ be maximal such that no vertices of P have

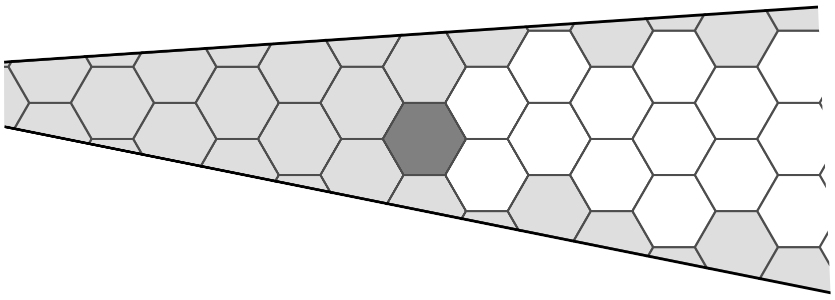

Figure 4 Blocked cells between two polygon edges.

their x-coordinate inside I'. That interval will look like the top of Fig. 5, i.e., there are k pairs of subsegments of edges, bordering part of the polygon. For each pair, we compute their minimum distances $d_1, \dots, d_k$ which are bounded by R_P and the distance of the two points of the intersection of the edge with the line $x = x_0$ or $x = x_1$. Then, we cut out $I := [x_0, x_1] := [x'_0 + 2R_P, x'_1 - 2R_P]$, and replace that part as illustrated at the bottom of Fig. 5. More specifically, each pair of edges is replaced by a rectilinear path of width d_i from left to right, beginning with the lowest edge. The opening between the edges is constricted to d_i on both sides. For each pair of edges i, let y_i (y'_i) be the y-coordinate of the intersection of the lower edge with the left (right) boundary of I. W.l.o.g., we may assume $y_i \le y'_i$. We then replace the lower edge by a path constructed as follows. Begin in (x_0, y_i) and move right until either reaching x_1 (see d_1 in Fig. 5) or an edge of pair $i-1$ (see d_3). We can clearly move right until at least $x_1 - \sum_{j=1}^{i-1} d_i \ge x_1 - n \cdot R_P$. Then we move straight up to y'_i and continue to (x_1, y'_i). Since $x_1 - x_0 \ge (n+2) \cdot R_P$, we move a distance of at least R_P right before moving up. The upper edge is replaced analogously such that that the distance between parallel segments is d_i. There will be an interval I'' with a size of at least $x_1 - x_0 - (n+1) \cdot R_p \ge R_P$ inside I where all edges are parallel to the x-axis. We compress I'' to a length of R_P. We obtain a polygon P' in polynomial time by performing the above steps on P both in x- and y-direction.

▶ **Lemma 3.** *P' has $O(n^4)$ vertices and intersects a polynomial number of cells of size $r_{P'} = O(R_P/n^4)$.*

Consequently, the size of $G_{P'}$ is polynomial in n. What remains to show is that the compression steps did not change sweepwidth by much.

▶ **Lemma 4.** *Let $\tilde{P}$ be the result of compressing P along one axis. Then $sw(\tilde{P}) = \Theta(sw(P))$.*

Proof sketch. Given a canonical decontamination sweep of P, we construct a decontamination sweep b of P which does not maintain barriers inside each interval $\hat{I} := [x_0 - R_P, x_1 + R_P]$, where x_0, x_1 are defined as above. Instead, b only uses barriers inside $\hat{I}$ for sweeping the region between edge pairs and subsequently places barriers just outside $\hat{I}$ to block off that region if necessary. We can show that this construction increases bottleneck length by at most a constant factor. Next, we construct a sweep $\tilde{b}$ of $\tilde{P}$ from b, where b and $\tilde{b}$ have the same barriers outside compressed regions and sweep corresponding compressed regions at the same time. This allows us to show $sw(\tilde{P}) = O(sw(P))$. $sw(P) = O(sw(\tilde{P}))$ follows analogously. ◀

The above lemmas directly imply the final theorem.

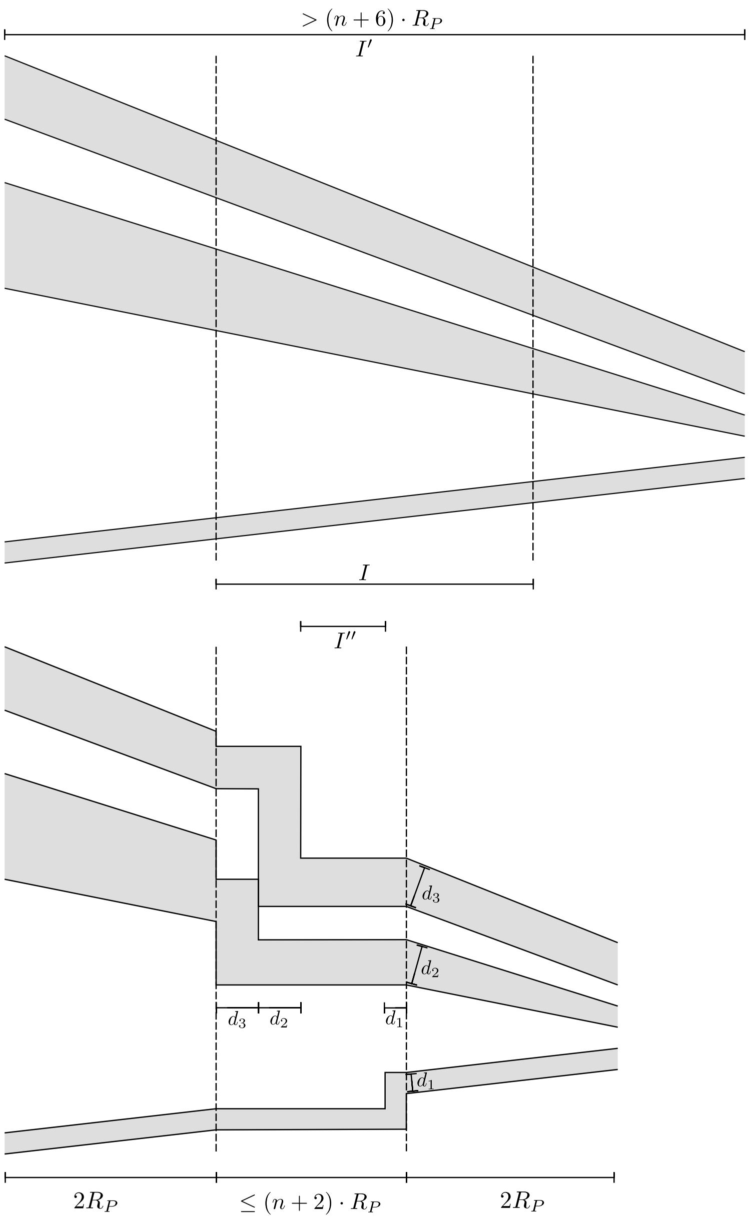

Figure 5 Illustration of polygon compression. d_1, d_2, d_3 are the minimum distance between their respective lines. The area inside P is depicted gray.

▶ **Theorem 5.** *There exists a polynomial time $O(\log n)$-approximation algorithm for computing $sw(P)$.*

The exact computation of $ns(G_P)$ is in $\mathcal{NP}$ [11], which implies the following corollary.

▶ **Corollary 6.** *$O(1)$-approximation of the sweepwidth of is in $\mathcal{NP}$.*

▶ Remark. If P is simple, it can be shown that $sw(P) = O(R_P \log n)$, which immediately gives an $O(\log n)$-approximation.

3 Conclusion

We constructed a polynomial time approximation algorithm for the relatively novel problem of computing a polygon's sweepwidth. Our proofs imply explicit constructions of sweeps. However, the runtime can be considered prohibitively bad. Improving approximation factors and runtime, potentially also for simple polygons, might therefore be interesting future work.

Acknowledgments. I thank Christian Scheideler and Kristian Hinnenthal for their help and advice.

References

1 Binay Bhattacharya, Tsunehiko Kameda, and John Z. Zhang. Surveillance of a polygonal area by a mobile searcher from the boundary: Searchability testing. In *2009 IEEE International Conference on Robotics and Automation*, pages 2461–2466, May 2009.

2 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1 – 45, 1998.

3 Alon Efrat, Mikko Nikkilä, and Valentin Polishchuk. Sweeping a terrain by collaborative aerial vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, pages 4–13, New York, NY, USA, 2013. ACM.

4 Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236 – 245, 2008. Graph Searching.

5 Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1):153, Nov 1987.

6 Jacob E. Goodman, János Pach, and Chee K. Yap. Mountain climbing, ladder moving, and the ring-width of a polygon. *The American Mathematical Monthly*, 96(6):494–510, 1989.

7 Qian-Ping Gu and Gengchun Xu. Near-linear time constant-factor approximation algorithm for branch-decomposition of planar graphs. In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science*, pages 238–249, Cham, 2014. Springer International Publishing.

8 Borislav Karaivanov, Minko Markov, Jack Snoeyink, and Tzvetalin S. Vassilev. Decontaminating planar regions by sweeping with barrier curves. In *26th Canadian Conference on Computational Geometry, CCCG 2014*, pages 206–211, 2014.

9 Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47:205–218, 1986.

10 Kyle Klein and Subhash Suri. Catch me if you can: Pursuit and capture in polygonal environments with obstacles. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 2010–2016. AAAI Press, 2012.

11 Andrea S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40:224–245, 1993.

12 Minko Markov, Vladislav Haralampiev, and Georgi Georgiev. Lower bounds on the directed sweepwidth of planar shapes. 2015.

13 Jorge Urrutia. Chapter 22 - art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 973 – 1027. North-Holland, Amsterdam, 2000.

14 Chee-Keng Yap. How to move a chair through a door. *IEEE Journal on Robotics and Automation*, 3(3):172–181, June 1987.