BACHELOR THESIS

DECONTAMINATING PLANAR REGIONS WITH FINITE AUTOMATON ROBOTS AND TILES

DORIAN RUDOLPH

Supervisor
Prof. Dr. Christian Scheideler

PADERBORN UNIVERSITY

Department of Computer Science
Theory of Distributed Systems

18. September 2018

Dorian Rudolph: *Decontaminating Planar Regions With Finite Automaton Robots and Tiles,* © 18. September 2018

SUPERVISOR:
Prof. Dr. Christian Scheideler

REVIEWERS:
Prof. Dr. Christian Scheideler
Prof. Dr. Friedhelm Meyer auf der Heide

MATRICULATION NUMBER:
7047011

TIME FRAME:
24. May 2018 — 24. October 2018

ABSTRACT

We consider the problem of decontaminating nanoscale regions using robots. These regions are modeled as finite induced subgraphs of the triangular lattice graph and are assumed to be contaminated by an arbitrarily fast spreading contaminant. Finite-state automaton robots are tasked with decontaminating the environment by moving a set of hexagonal tiles across it, which are able to block the contaminant. We develop algorithms for decontaminating several classes of environments and consider the number of tiles needed in comparison to the optimum as well as their runtime. For parallelograms, we are able to achieve that optimum. We achieve approximation factors of $O(1)$ for convex environments, $O(\log n)$ for n-node environments without holes, and $O(\sqrt{n})$ for environments with holes fitting inside a hexagon of radius $O(\sqrt{n})$. We show that the runtime of these algorithms can be improved linearly in the number of robots. We further prove that the problem of determining the minimum number of tiles sufficient for decontamination is $\mathcal{NP}$-hard.

A related problem is the decontamination of polygonal environments by sweeping them with barrier curves. The contaminant is assumed to spread instantly along all paths not blocked by a barrier. The *sweepwidth* of a polygon is defined as the minimum over the maximum total length of barriers during a decontamination sweep. Computing sweepwidth for a given polygon is known to be $\mathcal{NP}$-hard. We develop approximation algorithms for computing sweepwidth of an n-vertex polygon, achieving an approximation factor of $O(1)$ for convex polygons in $O(n)$ time, $O(\log n)$ in $O(n)$ for simple polygons, $O(\log n)$ in $O(n^{14}\log^4 n)$ and $O(\sqrt{n})$ in $O(n\log^4 n)$ for complex polygons. Further, we define a class of polygons consisting of aligned squares attached to each other in a tree-like fashion and provide an $O(1)$ approximation algorithm for sweepwidth in $O(n^3\log n)$ time. These algorithms construct decontamination sweeps explicitly.

ACKNOWLEDGMENTS

I would like to thank Lorena Rudolph, Nils Loose, and Kristian Hinnenthal for proofreading and my supervisor Christian Scheideler for helpful discussions.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Suppose some environment becomes *contaminated*. It is not difficult to think of many examples in the real world, e. g., virus infections or cancerous cells inside the human body, water contaminated with oil, radioactive material inside a nuclear reactor, tools coated with a corrosive substance after use, and cleaning up after accidents. In many situations, the environment is dangerous or impossible to access for humans. Therefore, using robots for decontamination is an attractive proposition, which we[1] investigate in this thesis.

We will begin this introduction by describing the motivation behind this research. Next, we introduce the formal models used in this thesis and describe related work, outlining what research has been done in our models and which other models exist. Finally, we give a brief overview of our results.

1.1 MOTIVATION

We already gave some examples where decontamination is a relevant problem. More specifically, we will consider this problem in the context of nanorobotics. This is motivated by significant progress in nanorobotics in recent years and its potential significance in various fields such as medicine and electronics. Many situations where decontamination is necessary are in the nanoscale (e. g., surgical operations inside the human body), hence requiring sufficiently small robots. In the future, nanorobots could even become attractive for larger scale operations due to a high degree of flexibility, provided the development of adequate manufacturing processes.

Here, we restrict ourselves to two-dimensional environments. They already offer a variety of potential applications and allow for the usage of techniques that might not be applicable in higher dimensions. Moreover, we believe it is important to understand the two-dimensional case before considering three (or maybe even higher) dimensions. In this thesis, we develop a theoretical model for decontamination, as well as algorithms in that model to perform decontamination.

Many models have been developed for nanorobots and nanomatter. Our model is based on a model for hybrid programmable matter, introduced by Gmyr et al. [16], consisting of active *robots* interacting with *passive* tiles. Tiles and robots inhabit the nodes of a triangular lattice graph. Related models often make use of only active elements. In practice, passive elements might be significantly cheaper to manufacture than active elements. Thus, both active and passive elements are present in our model. It is likely that nanorobots will only have limited computational capabilities. We therefore choose to implement robots as finite automata, i. e., they only have a finite number of states and need to compute their actions based on their state together with local information.

1 As customary in scientific writing, the pronoun *we* is used despite this thesis having only a single author.

The general idea of our model is that an environment is defined by a finite induced subgraph of the triangular lattice graph, which is initially considered to be fully contaminated. The robots' goal is now to remove the contaminant from the environment entirely, thereby performing *decontamination*. We assume the contaminant to spread arbitrarily fast (e. g., a gas or an intruder much faster than the robots). The robots therefore need to be able to block off already decontaminated regions to avoid immediate recontamination. To that end, they build barriers from tiles. We will try to minimize the number of tiles as well as the time needed to perform decontamination. Also, we will look at how this process can be sped up using multiple robots.

In the described model, the environment is essentially a polygon, as it is made up of tiles located on the lattice graph. This leads to a similar, purely geometric problem with the goal of decontaminating a polygon by *sweeping* it with a set of *barrier curves*, introduced by Karaivanov et al. [22]. They prove that calculating the minimal total curve length needed is $\mathcal{NP}$-hard. We will develop approximation algorithms to compute sweeps in polynomial time.

1.2 MODELS

We make use of three different, yet related models. First, we state the model targeting discrete robots and tiles. When using discrete tiles to decontaminate an environment, the intuitive approach is to build barriers. This lends itself particularly well to a geometric setting where a polygon is to be decontaminated using barrier curves. A similar problem has also been studied on graphs, in literature usually referred to as a *search game* (e. g., [24]). We will introduce variants called *node searching* and *edge searching*.

1.2.1 *Robot Model*

First, we present the model aimed at representing nanorobots. The following descriptions are adapted from [15] and extended by adding the restriction to a finite environment and a notion of (de)contamination.

We define the *environment* $\mathcal{E}$ as the graph induced by a finite set of connected nodes of the infinite triangular lattice graph. Nodes with a degree of less than six shall be denoted as *border nodes*. $\mathcal{E}$ is inhabited by a finite set of *active* agents (the *robots*) and *passive* hexagonal tiles. Robots and tiles occupy exactly one node. Each node may be occupied by at most one robot and one tile. Robots also have the ability to carry tiles. Carried tiles are not considered to occupy a node. We refer to nodes occupied by a tile or a robot carrying a tile as *tiled*.

Next, we will formally introduce the notions of *holes* and *outlines*. Let G be the graph induced by the set of nodes of the grid graph not belonging to $\mathcal{E}$ and $G_1, \dots, G_k$ its connected components. Exactly one G_i is infinite and corresponds to the *outside* of $\mathcal{E}$. Each finite G_i defines a *hole* in $\mathcal{E}$. We call $\mathcal{E}$ *simple* if it has no holes. For each G_i, there exists a unique (not necessarily simple) cycle C_i of all border nodes adjacent to G_i, obtained by traversing the polygon separating hexagons of $\mathcal{E}$ from hexagons of G_i (see black lines in Figure 1). We will refer to C_i as an *outline*.

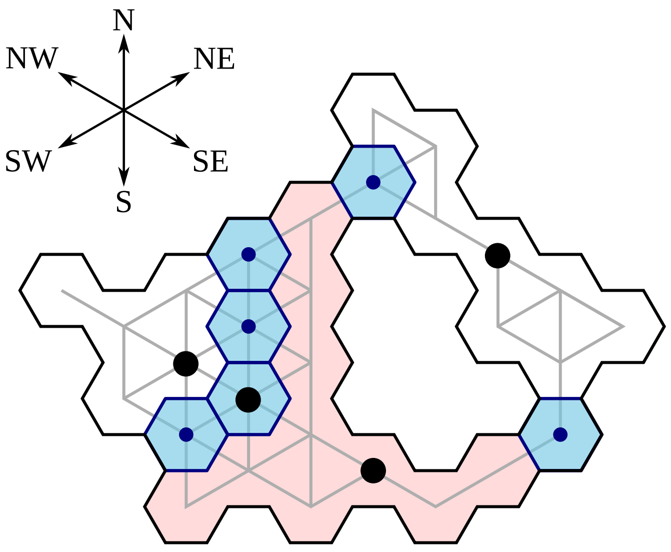


Figure 1: A connected configuration (tiles as blue hexagons, robots as black circles, environment graph as gray lines and its outlines as black polygon, contamination as red area). Removing any tile would lead to immediate recontamination of the adjacent area.

A *configuration* consists of the positions of tiles and robots. It is considered *connected* if all connected components of the graph induced by tiled nodes contain a border node, and all robots are located on a border node, a node containing a tile, or are adjacent to a node containing a tile. We require the configuration to remain connected throughout the execution. See Figure 1 for an example of a connected environment. The idea behind connectivity is that robots and tiles might be floating on a liquid. As such, they should always form a connected structure together with the environment's outline and not "float freely".

Each robot acts as a *deterministic finite automaton*, which is shared by all robots, and operates in *look-compute-move* cycles. In the *look* phase, the robot can observe the node it occupies and its neighbors. In case of border nodes, it can determine which neighbors are accessible, i.e., part of $\mathcal{E}$. For each adjacent node, it can detect whether it is occupied by a tile or a robot, and, in the latter case, the current state of that robot, including whether it carries a tile. Note that robots are not able to detect whether a node is contaminated, since this would not always be possible in a real context, where the contaminant might not spread throughout the entire environment or could also just be a single intruder, for example. In the *compute* phase, the robot can use information from the look phase together with its state to determine its next move. It may further change its own state and the state of all robots at adjacent nodes. In the *move* phase, the robot can either take a tile from its current node, place a tile it is carrying at that node, move to an adjacent node while possibly carrying a tile with it, or pass a tile it is carrying to a neighboring robot that does not yet carry a tile. Each robot can carry at most one tile.

We do not require a robot to know its global orientation, but to be able to maintain its relative orientation with respect to its original orientation. Furthermore, robots share a common chirality, i. e., an understanding of what a clockwise (or counterclockwise) rotation is.

We assume the standard asynchronous model from distributed computing in which robots are *activated* in an arbitrary sequence of *activations*. Whenever a robot is activated, it performs exactly one look-compute-move cycle before the next robot is activated. We measure the time complexity of our algorithms in the number of *rounds*, where a round is over whenever each robot has been activated at least once.

Initially, all non-tiled nodes in $\mathcal{E}$ are *contaminated*. If a robot carrying a tile moves onto a contaminated node, that node will be *decontaminated*. After each activation, all nodes reachable from contaminated nodes via paths containing no tiled nodes will be contaminated as well. Figure 1 shows an example of a partially contaminated configuration. The goal is that once the robots have terminated, all tiles should be decontaminated. We seek to minimize the number of tiles required to decontaminate $\mathcal{E}$, $td(\mathcal{E})$, as well as the number of rounds. Note that in the context of $td(\mathcal{E})$, robots are allowed to be constructed specifically for $\mathcal{E}$. It is therefore possible that no robot can decontaminate all environments $\mathcal{E}$ with $td(\mathcal{E})$ tiles.

1.2.2 *Geometric Model*

The outlines of the environment in the previous model as seen in Figure 1 already indicate how such an environment may be viewed as a polygon. In this section, we look at how to decontaminate a polygon by *sweeping* it with *barrier curves*. To that end, we define what a *sweep* of a shape is and state some existing results from [22]. Intuitively, one can think of these curves as barriers restricting the contaminant. By moving these barriers closer together, we gradually reduce the area wherein the contaminant remains until the shape is fully decontaminated.

The following definitions are adapted from [22]. Let $S \subset \mathbb{R}^2$ be a compact, path-connected region, i. e., S is a closed and bounded subset of the Euclidean plane such that for any two points $p, q \in S$, there exists a continuous function $f\colon [0,1] \to S$ with $f(0) = p$, $f(1) = q$. We refer to such a function as a *path* or *curve*, depending on the context. Since we are only interested in polygons, S will always be the set of points on a finite non-self-intersecting closed polygon, i. e., a polygon with a finite amount of vertices whose boundary does not cross itself. It is obvious that S is compact and path-connected for these polygons. Initially, S is considered contaminated.

We sweep S using a set of moving *barrier points* $b\colon [0,1] \to 2^S$. The barrier points $b(t)$ are decontaminated at time t. A decontaminated point q becomes *recontaminated* if there exists a path from a contaminated point p not intersecting $b(t)$. In order to define when a region S is decontaminated by a sweep b, it is more convenient to think of the contaminant as an arbitrarily fast moving intruder we want to catch that cannot jump over the barriers. Therefore, S is *decontaminated* if and only if for every continuous curve $\sigma\colon [0,1] \to S$ and continuous non-decreasing function $\tau\colon [0,1] \to [0,1]$ with $\tau(0) = 0$ and $\tau(1) = 1$, there exists some $t_0 \in [0,1]$ such that $\sigma(t_0)$ is part of a barrier in $b(\tau(t_0))$. Think of σ as the path the intruder takes in trying to evade our barriers. Instead of moving arbitrarily fast, we can also imagine the intruder to be able to change the movement speed of the barriers, possibly to a temporary standstill, i. e., the barriers are at $b(\tau(t))$ when the searcher is at $\sigma(t)$. t_0 would then correspond to the point in time when the intruder is caught.

We restrict barriers to piecewise continuously differentiable *barrier curves*. This allows us to describe a sweep by a function $b\colon [0,1]^2 \to S$ such that $b(s,t)$ is piecewise

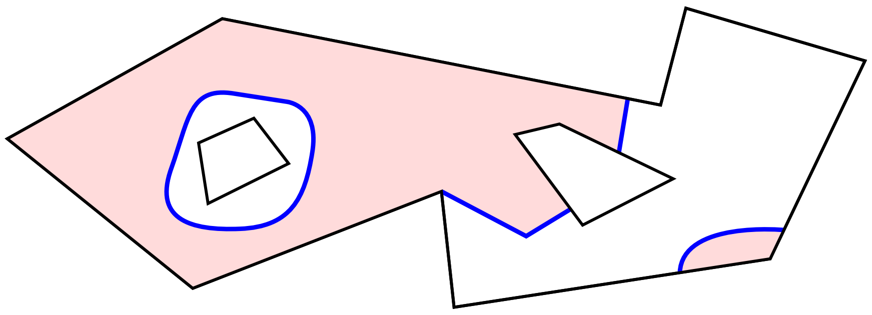


Figure 2: Incomplete sweep of a polygon (red area contaminated, barriers blue).

continuous in both curve parameter s and time t, and for any t, $b(\cdot, t)$, is piecewise continuously differentiable. Therefore, we can measure the total length of barriers at time t as the sum of the arc lengths of all pieces. $b(\cdot, t)$ consists of possibly multiple piecewise continuously differentiable curves, each representing a barrier at time t. The *bottleneck length* of b is defined as the supremum over all lengths of $b(\cdot, t)$. An exemplary sweep is depicted in Figure 2.

We refer to the minimum bottleneck length of a decontamination sweep of S as *sweepwidth* of S, denoted $sw(S)$. Generally, we will seek to find a decontamination sweep whose minimum bottleneck length is as close to $sw(S)$ as possible.

1.2.3 *Graph Model*

In this section, we define the *node/edge searching game* as introduced in [24]. The following definitions are adapted from there.

We start off with the edge searching game. Let G be an undirected graph. Initially, all edges of G are considered *contaminated*. A set of *searchers*, each of which can be placed onto at most one node of G, will be used to decontaminate the graph. Nodes may contain multiple searchers. A strategy $\mathcal{S}$ consists of a sequence of moves of the following types:

(a) *Placing* a searcher, which is not on a node, onto a node.
(b) *Sliding* a searcher from a node along an edge.
(c) *Removing* a searcher from a node.

An edge e is *cleared* (or *decontaminated*) by sliding a searcher along that edge. A node is considered *guarded* while there is a searcher on it. It remains clear as long as there is no *unguarded path* (i. e., a path of unguarded nodes) from e to a contaminated edge. Once such a path emerges after a move, e is *recontaminated*.

Note that if e is the only contaminated edge incident to a node v, a searcher may be slid along e from v without causing recontamination. Otherwise, an additional searcher on v would be needed to prohibit immediate recontamination.

The decontamination (or search) is complete once all edges are decontaminated. The maximum number of searchers placed onto G at the same time is referred to as the *edge-search number* of $\mathcal{S}$, or $es(G, \mathcal{S})$. A strategy $\mathcal{S}^*$ *clearing* (or *decontaminating*) G is optimal if its edge-search number is minimal among all such strategies. We set $es(G) := es(G, \mathcal{S}^*)$ the edge-search number of G.

Similarly, we define the node searching game with the difference that only moves (b) and (c) are allowed and edges are decontaminated when there is a searcher on both incident nodes. The *node-search number ns* is defined analogously to *es*.

1.3 RELATED WORK

As previously mentioned, there exist various models aimed at nanoscale computing. In *tile self-assembly* [34], semi-passive tiles attach to each other following certain rules growing into larger structures. Other models consider sets of only active agents [9, 21, 39], cooperating to solve problems such as *shape formation*. Especially the *amoebot* model by Derakhshandeh et al. [9] is quite similar to our robot model, the main difference being that ours also includes passive agents. In our robot model, shape formation [15, 16] and recognizing certain side ratios of a parallelogram [17] have been explored. Note that we restrict ourselves to finite environments whereas previous work operated on an infinite triangular lattice graph.

A very similar problem to ours is investigated by Dereniowski and Urbańska [12]. They give a distributed monotone connected online strategy for searching an unknown partial grid with n nodes using $O(\sqrt{n})$ searchers and prove that in their model, any algorithm can be forced to use $\Omega(\sqrt{n})$ searchers when $O(\log n)$ searchers would be optimal for an offline strategy. Note that the choice of partial grids in comparison to our environments is arbitrary as mutual embeddings are trivial. The key difference lies in the restriction to monotone connected strategies, i.e., the decontaminated area must be connected and recontamination is disallowed since that largely prevents exploration of the environment. Hence, their competitiveness lower bound of $\Omega(\sqrt{n}/\log n)$ does not translate to our model. Their algorithms require the memory of each searcher to be polynomial in the network size, whereas we are restricted to constant memory in our agents.

Various related problems are also discussed by Altshuler et al. [1]. They begin by analyzing a cooperative cleaning protocol for a static observable contaminant in the infinite grid graph, i. e., the contaminant does not spread and robots can see which tiles are contaminated. They also consider expanding contaminants, either deterministically over time or according to a stochastic process, and give lower bounds for the number of agents needed to clean certain regions. A similar problem using two-dimensional cellular automata has been explored in [8]. This differs from our model where recontamination happens instantly. Lastly, they look at the cooperative hunters problem where a number of drones equipped with GPS and sensors of limited range seek to find a set of evading targets. Note that evading targets with limited maximum velocity are similar to contaminated areas that expand over time.

While we only consider the (triangular) grid graph, a large amount of research has also been done for more general cases and different graph topologies, usually referred to as *graph searching* in literature. A multitude of scenarios including online or offline, distributed or centralized settings have been considered. A survey including further references can be found in [13]. Note that for these problems, agents are usually controlled by a centralized algorithm with complete knowledge of the graph, whereas agents in our model have only access to local information.

Various search problems for polygons have been analyzed in literature. These include agents having to spot an intruder while moving freely across the polygon [25] or using a ray-like flashlight from the boundary [2], or viewing/illuminating the entire polygon, often also referred to as the *art gallery problem* [37]. In the geometric model proposed by Karaivanov et al. [22], we will look at sweeping a polygon with barrier curves. They construct optimal sweeps for rather simple classes of polygons (see Section 2.1) and prove that computing sweepwidth is $\mathcal{NP}$-hard. We will develop approximation algorithms for constructing sweeps. That model is generalized by Markov et al. [30] 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.

If restricted to a single curve and simple polygons, sweepwidth is equivalent to the *elastic ring-width* [40], solved in time $O(n^2 \log n)$ [18]. Hence, ring-width constitutes an upper bound for sweepwidth. However, that bound can be arbitrarily bad, e. g., for an arbitrarily narrow T-polygon (cf. Figure 4).

1.4 OUR CONTRIBUTION

In Chapter 2, we consider the problem of decontaminating n-vertex polygons using barrier curves (see Section 1.2.2) and seek to minimize the bottleneck length of the computed sweep. We consider several classes of polygons and develop algorithms to approximate the sweepwidth and construct sweeps with that bottleneck length. Moreover, we introduce a class of polygons called square-tree polygons (see Definition 14), for which we are able to give an $O(1)$ approximation for the sweepwidth. These results are summarized in Table 1. Furthermore, if we had a polynomial time $O(1)$ approximation algorithm for node search (see Section 1.2.3), we could construct such an algorithm for sweeping a complex polygon (see Theorem 37).

POLYGON TYPE	RUNTIME	APPR. FACTOR	REFERENCE
Convex	$O(n)$	$O(1)$	Section 2.2
STP	$O(n^3 \log n)$	$O(1)$	Section 2.3
Simple	$O(n)$	$O(\log n)$	Section 2.4
Complex	$O(n^{14} \log^4 n)$ [2]	$O(\log n)$	Section 2.5
Complex	$O(n \log^4 n)$	$O(\sqrt{n})$	Section 2.5.4
Complex	in $\mathcal{NP}$	$O(1)$	Corollary 47

Table 1: Summary of algorithms for computing the sweepwidth of a polygon.

In Chapter 3, we develop algorithms for decontaminating an n-node environment in the robot model (see Section 1.2.1), using as few tiles as possible. Similarly to above, we consider several classes of environments. These results are summarized in Table 2. We discuss how to use multiple robots to speed up those algorithms. For all algorithms, we are able to achieve linear speedup under certain restrictions regarding the number of robots. We can also show that the runtime achieved for

2 Actually constructing the sweep takes time $O(n^{20} \log^4 n)$.

convex environments is asymptotically optimal. Lastly, we prove that computing the minimum number of tiles needed to decontaminate an environment is $\mathcal{NP}$-hard (see Section 3.4).

ENVIRONMENT TYPE	RUNTIME	APPR. FACTOR	REFERENCE
Parallelogram	$O(n)$	1	Section 3.1.1
Convex	$O(n)$	$O(1)$	Section 3.1
Simple	$O(n^2)$	$O(\log n)$	Section 3.2
Complex	$O(n^2 \log n)$	$O(\sqrt{n})$ [3]	Section 3.3

Table 2: Summary of algorithms for decontaminating environments with robots.

3 Provided the environment has radius $O(\sqrt{n})$. If c is the maximum number of tiles on a line in direction $d \in \{\mathrm{N}, \mathrm{NW}, \mathrm{NE}\}$, then we have an approximation factor of $O(c)$.

2 DECONTAMINATION OF POLYGONS

This chapter deals with the decontamination of polygons, as defined in Section 1.2.2. Specifically, we will introduce various algorithms to compute decontamination sweeps. We begin by looking at existing results from [22] for optimal sweeps. Afterwards, we consider a class of polygons P for which we are able to give an $O(1)$ approximation, i. e., an algorithm that computes a sweep with a bottleneck length of at most $O(1) \cdot sw(P)$. Then we give an $O(\log n)$ approximation for simple polygons and one for complex polygons, having a significantly higher runtime cost. Therefore, we also give a more efficient approximation algorithm for complex polygons at the cost of having only an approximation factor of $O(\sqrt{n})$.

2.1 OPTIMAL SWEEPS

Karaivanov et al. [22] construct optimal sweeps for some simple classes of polygons. Before describing sweeps, we state some of their results as they are necessary for reasoning about optimal decontamination sweeps.

LEMMA 1: For two path-connected planar regions A and B with $A \subseteq B$, it holds that $sw(A) \leq sw(B)$ [22].

The intuition behind the previous lemma is that we can obtain a decontamination sweep for A by restricting the barriers of a decontamination sweep of B to A.

Generally, reasoning about curves inside a polygon appears to be rather difficult. The next theorem allows us to only consider sweeps whose barriers consist of one or two line segments connecting two points on the polygon's boundary.

DEFINITION 2: A sweep is in *canonical form* if it only consists of curves made up by one or two line segments that start and end on the region border (i. e., the outline in case of polygons) [22].

If a sweep is in canonical form, all of its barriers are similar to the barriers connected to the hole of the polygon in Figure 2.

THEOREM 3: From each decontamination sweep, a decontamination sweep in canonical form can be constructed without increasing its length at any time [22].

The following corollaries are implied directly.

COROLLARY 4: A point at distance d from the region border needs curves of length at least $2d$ to be swept [22].

COROLLARY 5: The sweepwidth of a circle equals its diameter [22].

COROLLARY 6: The sweepwidth of an $a \times b$ rectangle is $\min\{a, b\}$ [22].

An *L-polygon* is an orthogonal polygon, i. e., each interior angle is either 90° or 270°, made up by the union of two axis-aligned rectangles of dimensions $a \times w$ and $w \times b$ sharing a single vertex, as depicted in Figure 3. The sweep consists of a single barrier, starting at $1-2$, then moved to $3-4$, $4-5$, $4-6$, and finally to $7-8$.

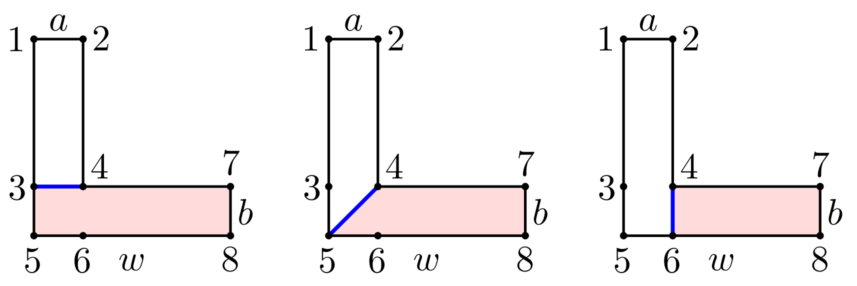


Figure 3: Decontamination sweep of an L-polygon (reproduced from [22]).

LEMMA 7: An L-polygon with rectangles of dimensions $a \times w$ and $w \times b$, $w \geq a + b$ has $sw(L) = \sqrt{a^2 + b^2}$ [22].

A *T-polygon* is the union of two axis-aligned rectangles of dimensions $w \times a$ and $b \times w$ such that the short side of the latter rectangle is attached to the middle of the former, as depicted in Figure 4. Depending on the ratio of a to b, there are two optimal ways to sweep a T-polygon. In the left image, the vertical part is cleaned first, with a barrier stopping at 1—2. Afterwards, the horizontal part is swept with a barrier starting at 4—5 and moved to 6—7. In the right image, after cleaning the vertical part, the barrier moves from 1—2 to 1—3—2 and is subsequently split, its parts moving to 4—5 and 6—7, respectively.

LEMMA 8: A T-polygon with rectangles of dimensions $w \times a$ and $b \times w$, $w \geq 2a + b$ has

$$sw(T) = \begin{cases} \sqrt{4a^2 + b^2} & \text{if } a \leq 2b/3 \\ a + b & \text{otherwise} \end{cases} \quad [22].$$

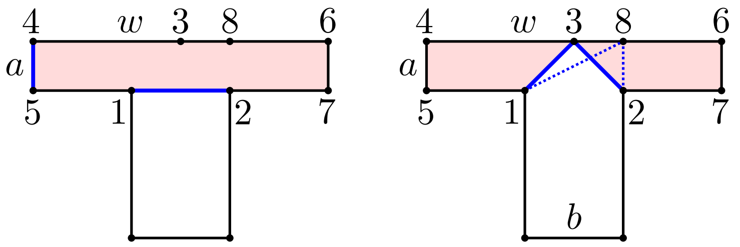


Figure 4: Decontamination sweep of a T-polygon (reproduced from [22]).

This result can be generalized to a *comb polygon* which is similar to the T-polygon but with multiple vertical rectangles.

2.2 CONVEX POLYGONS

In this section, we shall consider convex polygons, i. e., polygons P such that the line between any two points in P is also fully contained in P. We will show that the following upper bound on $sw(P)$ is within a factor of $O(1)$ of the actual sweepwidth of P if P is convex.

DEFINITION 9: Let $S \subseteq \mathbb{R}^2$ be compact and path-connected. A *supporting line* of S is a line intersecting the boundary of S but not its interior. The *width* of S, $w(S)$, is the minimal distance of two parallel supporting lines of S.

As an example, the blue lines above and below the polygon in Figure 5 are part of two parallel supporting lines.

LEMMA 10: For any compact path-connected shape S, it holds that $sw(S) \leq w(S)$ [22].

Proof. S is fully contained between two parallel supporting lines at distance $w(S)$. The area between these lines can clearly be swept with a single curve of length $w(S)$. Due to Lemma 1, we have $sw(S) \leq w(S)$. □

Karaivanov et al. [22] conjecture that $sw(P) = w(P)$ holds if P is a convex polygon. They plan on proving this in a full version of their paper, however that is not available yet, to the best of our knowledge. Hence, we will show $sw(P) = \Theta(w(P))$ next.

LEMMA 11: Given a convex polygon P, a rectangle r can be inscribed in C such that a homothetic copy R of r (i.e., a version of r upscaled by a factor λ) is circumscribed about P with positive homothety ratio $\lambda \leq 2$ [28]. See Figure 5 for a visualization.

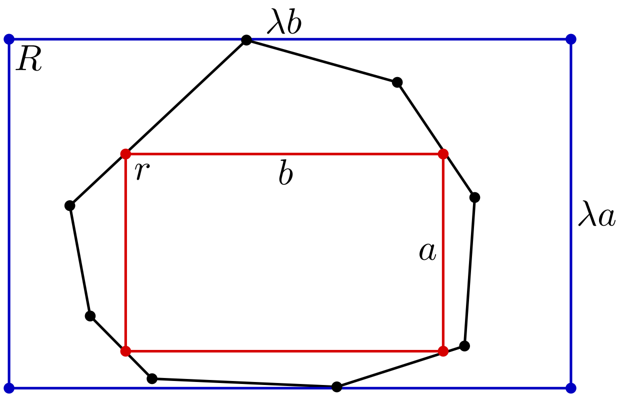


Figure 5: Inscribed and circumscribed rectangle of convex polygon as in Lemma 11.

LEMMA 12: Given a convex polygon P, it holds that $sw(P) \geq w(P)/2$.

Proof. Let r be an inscribed rectangle in P with side lengths $a \leq b$ such that a homothetic copy R of r with positive homothety ratio $\lambda \leq 2$ exists that is circumscribed about P as in Lemma 11. Clearly, $\lambda a \geq w(P) \geq sw(P)$. From Lemma 1 it follows that $sw(P) \geq a$. Therefore, $\lambda \cdot sw(P) \geq w(P)$ and finally $sw(P) \geq w(P)/2$. □

We have shown that the decontamination sweep constructed in the proof of Lemma 10 is within $O(1)$ of the optimum. We still need an algorithm to compute that sweep. Before we get to that however, we need to discuss what kind of output that algorithm shall produce. Obviously, the algorithm cannot output all points of the barrier curves at all times because they are infinite. Thus, the output needs to define the sweep using some *suitable format*. For this algorithm, we will be quite explicit and describe how to construct a function defining the barrier points' coordinates during the sweep. In later algorithms, we will describe the output in a less formal manner but shall still ensure that it is clear how the sweep is constructed.

In the following, we assume that common mathematical operations on numeric types such as the coordinates of a polygon's vertices can be performed in time $O(1)$.

THEOREM 13: Let P be a convex polygon with n vertices, given by a circular linked list as points $p_i = (x_i, y_i)$. A decontamination sweep with bottleneck length $w(P) = \Theta(sw(P))$ can be constructed in time $O(n)$.

Proof. The width of P and the corresponding supporting lines are be computed in time $O(n)$ [20]. Note that one edge of the polygon will be parallel to the supporting lines. Afterwards, P is rotated such that the supporting lines are parallel to the x-axis. This is possible in $O(n)$ by just applying a rotation matrix. We further translate and scale the polygon such that its minimal x-coordinate is 0 and the maximal 1. The barrier is then parallel to the y-axis.

Next, we sort the vertices such that $x_i \leq x_{i+1}$ for $x = 1, \dots, n$. W.l.o.g., we can assume that P has no redundant points, i. e., points whose removal would not change the outline of P. Such points could also easily be removed in time $O(n)$. The sorting can be done in $O(n)$ by splitting the list of points where x_i is maximal and minimal, respectively. Since P is convex, there are at most two points with maximal (minimal) x-coordinate and if so, they are connected. Thus, the described split yields two lists of points, sorted by their x-coordinate which can be merged to one in time $O(n)$.

On a horizontal interval from x_i to x_{i+1}, the upper and lower part of the polygon's outline are characterized by two affine linear functions $f_i(x)$ and $g_i(x)$. Given a time $t \in [0, 1]$, let i be such that $t \in [x_i, x_{i+1}]$. The barrier points at time t shall therefore be $(t, f_i(t))$ and $(t, g_i(t))$. This defines a sweep as in Lemma 10 and therefore a decontamination sweep. The output consists of the functions f_i and g_i for each interval $[x_i, x_{i+1}]$. These functions can easily be computed from the edges incident to x_i and x_{i+1} in time $O(1)$ each. Therefore, the algorithm has a runtime of $O(n)$. □

2.3 A CLASS OF POLYGONS WITH AN $O(1)$ APPROXIMATION

In this section, we look at another class of polygons for which we can give an $O(1)$ approximation algorithm for an optimal decontamination sweep.

2.3.1 *Square-tree Polygons and Weighted Node Search*

DEFINITION 14: A polygon P is a square-tree polygon (STP), if it is the union of axis-aligned squares $S_1, \dots, S_n$ with side lengths $r_i \in \mathbb{N}$ having the following properties:

(a) For $i \neq j$, let $I_{ij} = S_i \cap S_j$. W.l.o.g., let $r_i \leq r_j$. Either $I_{ij} = \emptyset$ or I_{ij} is exactly the set of points on one side of S_i.

(b) Let $T_P = (V, E)$ with $V = \{1, \dots, n\}$ and $E = \{\{i, j\} \mid I_{ij} \neq \emptyset\}$. T_P is a tree.

(c) Each side of S_i is adjacent to at most one other square.

See Figure 6 for an example. The graph G is drawn as defined above; r_i is written inside S_i.

The approach to decontaminate an STP will be to perform decontamination (or node search) on the weighted tree obtained by assigning weights $w(i) := r_i$ to the nodes of T_P. We will first formally define decontamination for weighted graphs as an extension of Section 1.2.3 and refer to an existing approximation algorithm for decontaminating weighted trees. Afterwards, we show how to transform a decontamination strategy on a weighted tree to a decontamination sweep in an STP.

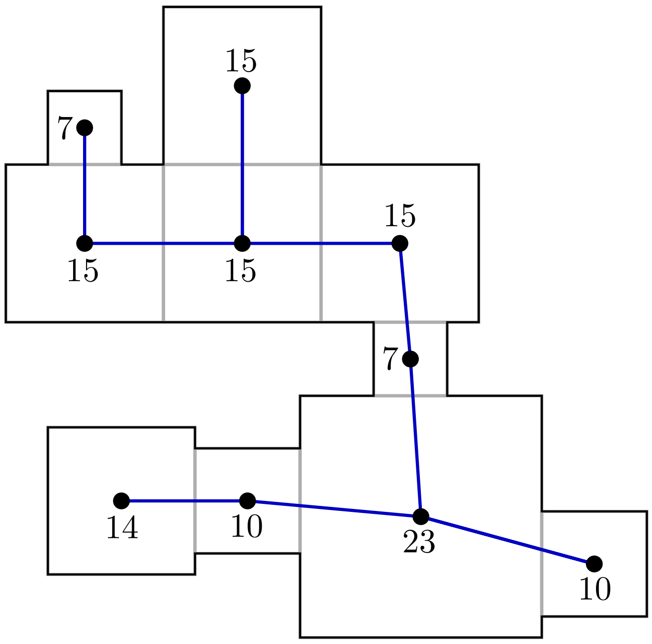


Figure 6: Square-tree polygon as defined in Definition 14.

Lastly, we show that the node-search number of a weighted tree and the sweepwidth of the corresponding STP are within a constant factor of one another.

Dereniowski [11] gives an $O(1)$ approximation algorithm for the edge search problem in weighted trees. We will, however, use the weighted node search problem in our proofs. Although they state that their methods also yield an $O(1)$ approximation for weighted node search, they do not give an algorithm for obtaining such a node-search strategy from an edge-search strategy. Therefore, we will introduce the *weighted edge search problem* next (as defined in [11]), together with the *weighted node search problem* and show how to transform strategies from one to the other.

Let $G = (V, E, w)$, $w \colon V \cup E \to \mathbb{N}^+$ be a connected undirected weighted graph. Initially, all nodes and edges are considered contaminated. A *search strategy* $\mathcal{S}$ is a sequence of the following types of moves:

(a) Place $s \geq 1$ searchers on a node.
(b) Remove $s \geq 1$ searchers from a node v, provided at least s searchers are present at v.
(c) Slide $s \geq 1$ searchers from u to v, provided $\{u, v\} \in E$ and at least s searchers are present at u.

An edge $e \in E$ is cleared (or decontaminated) if at least $w(e)$ searchers simultaneously slide across e. A node $v \in V$ is cleared if it is occupied by at least $w(v)$ searchers. Nodes or edges are recontaminated if there exists a path of unguarded nodes (i. e., nodes v containing fewer than $w(v)$ searchers) between that node or edge and a contaminated node or edge. $\mathcal{S}$ is said to clear G if all nodes and edges are cleared at the end of $\mathcal{S}$. Let the *weighted edge-search number* of $\mathcal{S}$, $wes(G, \mathcal{S})$, be the maximum number of searchers simultaneously present on G during $\mathcal{S}$. $wes(G)$ is then the minimum $wes(G, \mathcal{S})$ among all strategies clearing G.

Next, we define *weighted node search*. Let $G = (V, E, w)$, $w \colon V \to \mathbb{N}^+$ be a connected undirected graph. Weighted node search is similar to weighted edge search, with the restriction that only moves (a) and (b) are allowed and an edge $\{u, v\}$ is cleared if $w(u)$ searchers are present at u and $w(v)$ at v. Further, only edges can be considered contaminated. We define the *weighted node-search number* wns akin to wes.

LEMMA 15: Let $G = (V, E, w)$, $w \colon V \cup E \to \mathbb{N}^+$ with $w(e) = 1$ for all edges $e \in E$. It holds that $wes(G) - 1 \leq wns(G) \leq 2 \cdot wes(G)$.

Proof. We will prove this lemma constructively by transforming a strategy for edge search to a strategy for node search and vice versa, without significantly increasing the number of searchers.

Let $\mathcal{S}_e$ be an edge-search strategy clearing G. We will now construct a node-search strategy $\mathcal{S}_n$ clearing G. Moves of type (a) and (b) can simply be transferred from $\mathcal{S}_e$ to $\mathcal{S}_n$. Sliding moves (c) from u to v however, need to be dealt with. If $\{u, v\}$ is actually cleared by the sliding move, u was guarded beforehand, otherwise recontamination would have occurred immediately. In case the edge is not cleared, the move was redundant and can just be removed. Otherwise in $\mathcal{S}_n$, we place $w(v)$ searchers on v and then remove as many searchers from u. We potentially also remove searchers from v to match the number of searchers v has in $\mathcal{S}_e$.

$\mathcal{S}_n$ clears G since edges and nodes are cleared at the same time they are cleared in $\mathcal{S}_e$ and recontamination semantics are equivalent for node and edge search. It holds that $wns(G, \mathcal{S}_n) \leq wes(G, \mathcal{S}_e) + \max_{v \in V} w(v) \leq 2 \cdot wes(G, \mathcal{S}_e)$ and hence $wns(G) \leq 2 \cdot wes(G)$.

Now, we prove the other inequality. Let $\mathcal{S}_n$ be a node-search strategy clearing G. We construct an edge-search strategy $\mathcal{S}_e$ as follows. Moves of (a) and (b) are taken from $\mathcal{S}_n$ as is. Whenever an edge $\{u, v\}$ is cleared in $\mathcal{S}_n$, we place an additional searcher on u and slide it over to v to clear the edge in the edge search model.

$\mathcal{S}_e$ clears G by the same argument as before. At most one additional searcher is used, thus $wes(G, \mathcal{S}_e) - 1 \leq wns(G, \mathcal{S}_n)$ and $wes(G) - 1 \leq wns(G)$. □

Together with the previous lemma, [11] allows us to construct an $O(1)$-approximate node-search strategy for weighted trees in runtime $O(\Delta n^3 \log n)$ where Δ is the maximum node degree. Since a tree T_P obtained from an STP P has $\Delta \leq 4$, we can find an optimal node-search strategy in time $O(n^3 \log n)$.

2.3.2 *Constructing a Decontamination Sweep*

Given a node-search strategy $\mathcal{S}$ for T_P, it is easy to construct a decontamination sweep for P:

LEMMA 16: Let P be an STP. It holds that $sw(P) \leq 5 \cdot wns(T_P)$.

Proof. Let $\mathcal{S}$ be a node-search strategy for T_P. We will construct a decontamination sweep from $\mathcal{S}$. When a node i is cleared, a barrier curve is put up around the corresponding square S_i. Afterwards, the inside of that barrier is swept, leaving the entire square decontaminated. A total curve length of at most $5 \cdot w(i) = 5r_i$ is required. Clearly, a square is decontaminated and recontaminated at exactly the same time the corresponding node would be in $\mathcal{S}$. We therefore obtain a decontamination sweep with bottleneck length at most $5 \cdot wns(T_P, \mathcal{S})$. It follows that $sw(P) \leq 5 \cdot wns(T_P)$.

Clearly, these operations can be performed in time $O(n)$. □

The final step is to construct a node-search strategy from a decontamination sweep, thereby showing that $wns(T_P) = O(sw(P))$.

LEMMA 17: Let P be an STP. It holds that $wns(T_P) = O(sw(P))$.

Proof. Consider a decontamination sweep of P. Due to Theorem 3, we may assume the sweep to be in canonical form, w.l.o.g. To decontaminate a square S_i, barriers of length at least r_i need to be inside S_i at some point, by Corollary 4. Whenever all paths connecting the centers of adjacent squares S_i and S_j are blocked by a barrier, curves of length at least $\min\{r_i, r_j\}$ inside $S_i \cup S_j$ are needed due to property (a) from Definition 14. Therefore, if all paths from S_i to the centers of adjacent squares $S_{j_1} \dots S_{j_l}$, $l \leq 4$ are blocked, then there are curves of length at least $\min\{r_i, \max r_{j_k}\}$ inside $\bigcup_k S_{j_k} \cup S_i$.

Now, let us look at a decontamination sweep. During the sweep, barrier curves partition the area of P into multiple decontaminated and contaminated areas. Let T be a maximal subtree of T_P with the property that the centers of the squares corresponding to its nodes are all in the same decontaminated area. Let i be a node of T that has neighbors $j_1, \dots, j_l$, $1 \leq l \leq 4$ in T_P that are not in T. Since T is maximal, the centers of S_i and S_{j_k} are separated. Thus, curves of length at least $\min\{r_i, \max r_{j_k}\}$ are inside S_i and its neighbors. We can therefore separate the node i from its neighbors $j_1, \dots, j_l$ using a total of $\min\{w(i), \sum w(j_k)\} \leq 4 \cdot \min\{r_i, \max r_{j_k}\}$ searchers, placed either on i or $j_1, \dots, j_l$ (1).

We construct a strategy $\mathcal{S}$ such that the subtrees T as defined above are always decontaminated (2). Consider the events altering these subtrees, i. e., (a) adding or (b) removing nodes, and (c) joining or (d) splitting of subtrees. We can ensure that no two events occur at exactly the same time by altering curves by an infinitesimal amount such that events occur slightly earlier or later without changing the bottleneck length of the sweep significantly.

In the following, we describe how to handle each event, such that (2) is satisfied at all times.

In case (a), if a node v becomes a new tree of size one, we first guard v. Depending on (1), fewer searchers might be required to guard the neighbors of v rather than v itself. In that case, we guard the neighbors and remove all searchers from v.

If v is added to an existing tree T, v is either already guarded or its parent u wrt. T. In the latter case, we guard v. As above, we now either guard the children of v or leave v itself guarded. The neighborhood of u might also need to be modified in accordance with (1).

The cases (b), (c), and (d) are handled analogously in that there are at most 5 affected nodes whose neighborhood needs to be modified due to (1). All searchers on nodes without neighbors not in the same subtree are removed. After these modifications, for each node i guarded as described in (1) with s searchers, curves of length at least $s/4$ are inside S_i or one of its neighbors. Hence, each point in a curve is "counted" at most 5 times and the number of searchers used is $O(sw(P))$. The modifications require an additional $O(\max r_i)$ searchers because at most one event is handled at a time. In total, $wns(T_P) \leq wns(T_P, \mathcal{S}) = O(sw(P))$. □

Putting the previous two lemmas together, we obtain the final theorem for this section.

THEOREM 18: Given an STP P together with its tree T_P, we can compute a decontamination sweep of P in time $O(n^3 \log n)$ with bottleneck length $O(sw(P))$.

Proof. First, a node-search strategy $\mathcal{S}$ clearing T_P is computed in time $O(n^3 \log n)$. From [11] and Lemma 17, we know that $wns(T_P, \mathcal{S}) = O(sw(P))$. The proof of Lemma 16 describes how to construct a sweep from $\mathcal{S}$ in time $O(n)$. $\square$

REMARK 19: We defined STPs as having integer side lengths. However, the previous algorithm also works with arbitrary side lengths if we round the r_i up to the nearest fraction of $\max r_i/n$, as it then uses at most $sw(P)$ more searchers than a version of the same algorithm able to deal with arbitrary side lengths (cf. Section 2.5). Further, we conjecture that our results still hold if we allow multiple adjacent squares on the same side.

REMARK 20: If we are given P, but not T_P, the latter can be computed greedily by searching for three adjacent edges with two inner angles of 90°, such that the middle edge is at most as long as the others and the square defined by these edges fits inside P. Under the promise that P is indeed an STP, such edges have to exist and the shortest is the side of a square associated with a leaf in T_P. We repeat this until all squares are identified.

More interesting is perhaps the problem to construct P, given T_P, or prove that it is impossible. E. g., a node with weight 1 and four neighbors with weight 2 cannot be constructed as an STP.

2.4 SIMPLE POLYGONS

In this section, we develop a linear time algorithm to compute a decontamination sweep of a simple n-vertex polygon P (i. e., a polygon without holes) with bottleneck length $O(R_P \log n) = O(sw(P) \log n)$ where R_P is the radius of the largest inscribed circle in P. This algorithm relies on the *medial axis* of P, which is introduced in the following.

2.4.1 *Medial Axis*

The *medial axis* M of P was originally proposed by Blum [3] and is defined as the set of centers of circles inside P touching the boundary in two or more points, together with convex vertices of P. For any point $x \in M$, we denote the maximal inscribed circle centered in x as $C(x)$ with radius $r(x)$. Figure 7 depicts the medial axis of a simple polygon, including $C(x)$ for several $x \in M$.

The medial axis forms a planar graph $G = (V, E)$ (inner nodes red, leaves dark red) whose edges are curves consisting of straight-line segments (blue) and portions of parabolic curves (green) [38]. For simple polygons, this graph is a tree [7, 29]. Note that edges and nodes of G will also refer to the corresponding points and curves in M. Thus, a node $v \in V$ identifies a point inside P, and an edge $e \in E$ a set of points. There always exists a node $v \in V$ with $r(v) = R_P$ (see the large circle in Figure 7).

In the following, we subdivide P into multiple polygons P_e for each $e = \{u, v\} \in E$ such that P_e intersects $P_{e'}$ iff e and e' are incident, and $e \subset P_e$. This will allow us to transform a node-search strategy on G into a decontamination sweep of P.

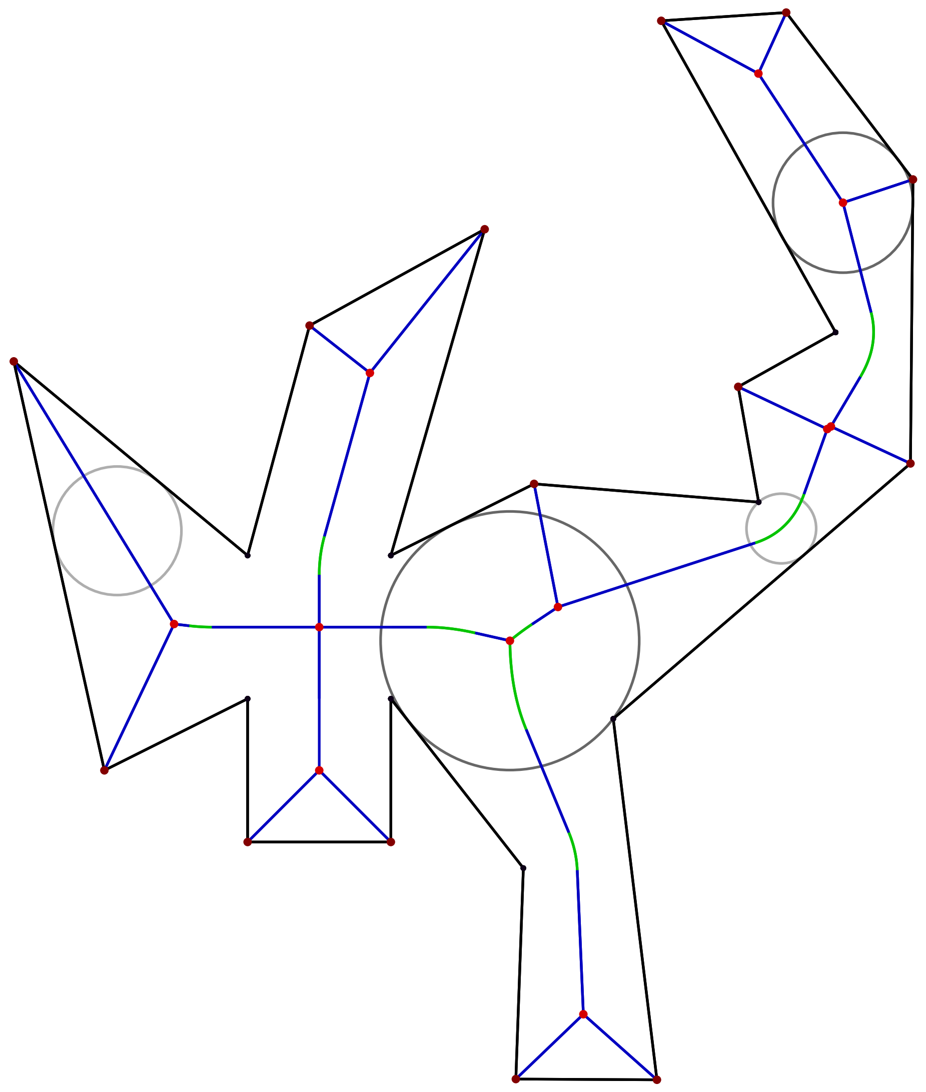


Figure 7: Medial axis of a simple polygon.

LEMMA 21: The radius function $r\colon M \to \mathbb{R}$ is (sequentially) continuous.

Proof. We show that $r_{|e}$ is continuous for each edge $e = \{u, v\} \in E$. The continuity of r follows. e can be regarded as a metric space, with the distance along e serving as a metric. It is isomorphic to $[0, 1]$ and hence compact. For metric spaces, continuous and sequentially continuous are equivalent. Assume that r is not continuous. Let $(x_n) \subseteq e$ with $\lim_{n\to\infty} x_n = x$ such that $(r(x_n))$ does not converge to $y := r(x)$. Let (x'_n) be a subsequence of (x_n) such that $(r(x'_n))$ converges to $y' \neq y$. There exists N such that $\forall n \geq N : |x_n - x| < |y' - y|/2$. But that contradicts $|r(x_n) - r(x)| \leq |x_n - x|$ because otherwise the smaller circle would be completely inside the larger one. □

LEMMA 22: If P is partitioned by the line segments from v to the contact points of $C(v)$ for all $v \in V$, then each obtained component contains one edge $e \in E$ as well as its incident nodes (see Figure 8). It shall be identified as P_e. P_e contains no other edges or nodes. Components P_e and $P_{e'}$ intersect each other iff e and e' are incident.

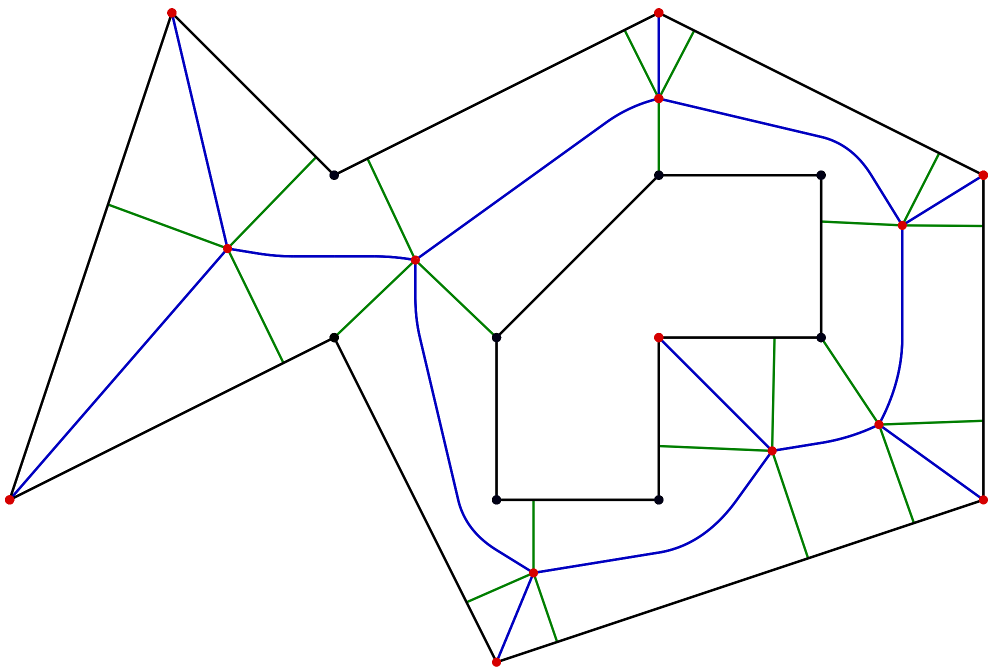


Figure 8: Partitioning a polygon based on the medial axis (blue) using line segments (green) from nodes to contact points of maximum inscribed circles centered at nodes.

Proof. For $v \in V$, $C(v)$ touches the outline of P in $\delta := \deg(v)$ points $p_1, \dots, p_\delta$ (ordered clockwise), such that exactly one edge incident to w emanates from w in between the line segments $\overline{vp_i}$ and $\overline{vp_{(i \bmod \delta)+1}}$ [29]. Similarly, if we draw such line segments from a point on an edge, they are separated by the edge as well. Edges do not intersect because G is planar. A line segment $\overline{vp_i}$ is not intersected by an edge, as visualized by Figure 9, since otherwise the circle $C(p')$ at the intersection p' would be contained in $C(v)$ and thus could only have one contact point with the outline of P. Two such line segments $\overline{vp}$ and $\overline{wq}$ can only intersect in their endpoints, since $C(p')$ on the intersection p' would otherwise have two contact points but $p' \notin M$ by the previous statement.

For an edge $e = \{v, w\} \in E$ and the corresponding line segments $\overline{vp}$ and $\overline{wq}$, such that both line segments lie on the same side of e, assume p and q to lie on different outlines (e.g. on the outer outline and on a hole). If we sweep a circle $C(x)$ along e from v to w, the contact point on that side of the edge, where p and q lie, has to jump between the outlines.

Let $e' := \{x \in e \mid x < x' \text{ for all } x' \in e \text{ where } C(x') \text{ does not touch the outline of } p\}$ and $y = \sup e'$. y must be on the interior of e, since $C(x)$ touches the same outline as p (q) on a short interval near v (w). By the continuity of r, e, and the outlines of P, $C(y)$ touches two different outlines on one side of e. Hence, x is a node, which is a contradiction. Therefore, every edge $e = \{u, v\} \in E$ is contained in only one component of the partition.

By the above reasoning, $P_e \subset P$ is the polygon constructed from the four relevant line segments and the two parts of outlines of P (see Figure 8). Assume there is some edge e' inside P_e. Since $e' = \{v', w'\}$ cannot cross the line segments or the outline of P, it must be fully contained in P_e. Hence, every path on M from v to v' is also inside P_e. But that would imply an edge branching off from e', which is impossible.

Clearly, the only points of intersection between polygons P_e and $P_{e'}$ can be the associated line segments, including the node. Hence, the intersection of P_e and $P_{e'}$ is equivalent to the incidence of e and e'. □

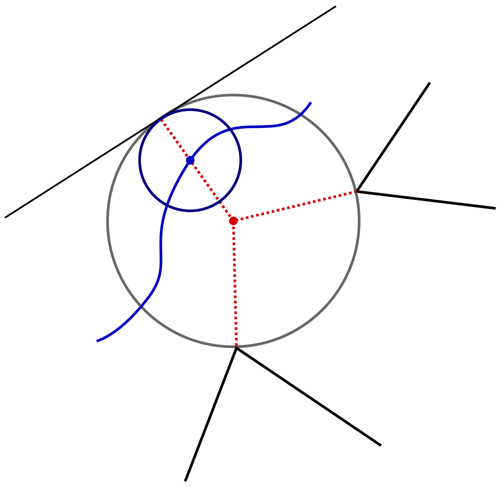


Figure 9: Visualization of the proof that no edge of the medial axis can cross the line segment connecting a node to one of its closest points on the polygon's outline.

The next lemma allows us to decontaminate the P_e polygons.

LEMMA 23: Let $e = \{u, v\} \in E$. If barriers are placed on $C(u)$ and $C(v)$ and another circle $C(x)$ is swept along the edge from u to v, the entire area $C(u) \cup C(v) \cup P_e$ is decontaminated.

Proof. Let X be the set of points of the edge from u to v, including u, v, and Y, Z the two parts of the outline between $C(u)$ and $C(v)$. This is well defined as outlined in Lemma 22. Let y_0 and z_0 be the intersection of Y and Z with $C(u)$, respectively. X, Y, Z with the Euclidean metric are sequentially compact, since they are isomorphic to $[0, 1]$ using the distance along the edge or outline as metric. Let $f\colon X \to Y$, $g\colon X \to Z$ be the functions mapping from the edge to the two closest points. f and g are well defined by the proof of Lemma 22.

Assume f is not continuous. Let $(x_n) \subseteq X$ with $\lim_{n\to\infty} x_n = x$ such that $(f(x_n))$ does not converge to $y := f(x)$. Let (x'_n) be a subsequence of (x_n) such that $(f(x'_n))$ converges to $y' \neq y$. We have $\lim_{n\to\infty} r(x_n) = r(x)$ since r is continuous. Hence, the circle of radius $r(x)$ centered in x touches both y' and y. But that contradicts the definition of f. Analogously, g is continuous.

After sweeping a circle $C(x)$ along X from u to x_1, the entire area bordered by Y, Z and the line segments $\overline{uy_0}$, $\overline{uz_0}$ and $\overline{x_1 f(x_1)}$, $\overline{x_1 g(x_1)}$ has been swept. This holds because the line segments $\overline{xf(x)}$, $\overline{xg(x)}$ (contained inside $C(x)$) move continuously as f, g, as well as the curve of e are continuous. Hence, an intruder on one side of these line segments would have to jump over these segments to get to the other side without touching them. Once $C(x)$ has reached v, the entire area $C(u) \cup C(v) \cup P_e$ has been decontaminated. □

An algorithm for computing an optimal node-search strategy in time $O(n \log n)$ is given by Megiddo et al. [31]. In order to comply with the targeted $O(n)$ runtime, we present an algorithm that computes a node-search strategy for a tree T with n nodes using $O(\log n)$ searchers.

2.4.2 *Constructing a Decontamination Sweep*

Algorithm 1 Algorithm to compute a node-search strategy for trees.

```
1: procedure NodeSearch($T, u, d$)    ▷ Tree $T$ rooted at $u$; $d(v)$ number of descendants of $v$
2:     $w \leftarrow \operatorname{argmax}_{\text{child } v \text{ of } u} d(v)$
3:     if $u$ is not guarded then
4:         Place searcher on $u$.
5:     for child $v \neq w$ of $u$ do
6:         NodeSearch($T_v, v, d$)    ▷ $T_v$ is the subtree rooted at $v$
7:     Place searcher on $w$.
8:     Remove searcher from $u$.    ▷ All subtrees of $v$ besides $T_w$ have been cleared
9:     NodeSearch($T_w, w, d$)
```

LEMMA 24: Algorithm 1 computes a node-search strategy for an n node tree using at most $\log n + 1$ searchers in time $O(n)$.

Proof. Correctness and runtime are obvious. We show that at most $\log n + 1$ searchers are used by induction on the number of nodes. The statement clearly holds for a single node. Now, assume NodeSearch is called with $d(u) = n$. Clearly, for each recursion from line 6, $d(v) \leq d(u)/2$. Hence, at most $\log(n/2) + 1 = \log n$ searchers are used during the recursions, for a total of $\log n + 1$ searchers in the subtree of u, including the searcher on u. Since all searchers are removed from the subtree after NodeSearch returns, no more than $\log n$ searchers are used during the loop. Between lines 6 and 9, only two searchers are on T_u. Since $d(w) < d(u)$ and no searcher is placed in line 4 in the recursion from line 9, no more than $\log n + 1$ searchers are on T_u during that recursion. □

We can now prove the final theorem of this section.

THEOREM 25: Given a simple polygon P with n vertices, we can compute a decontamination sweep in time $O(n)$ with bottleneck length $O(R_P \log n) = O(sw(P) \cdot \log n)$.

Proof. The first step is to compute the medial axis of P in time $O(n)$ using the algorithm by Chin et al. [6]. The underlying tree T can be extracted easily as the medial axis is represented using a set of nodes connected by curves. A node-search strategy $\mathcal{S}$ is obtained with Algorithm 1.

Since the decomposition given by Lemma 22 is isomorphic to the medial axis graph G, a node-search strategy on G can be transformed to a decontamination sweep. For each searcher on a node v, we add $C(v)$ as barrier. Once an edge is decontaminated in G, we sweep a circle along it as described in Lemma 23. This uses curves with a total length of at most $2\pi R_P \cdot (ns(T, \mathcal{S}) + 1) = O(R_P \log n)$.

Note that the radius function used in Lemma 23 can be computed from the curve segments making up the corresponding edge and the sites they are the bisectors of. We refer the reader to [29] for more details on bisectors in the context of medial axes. □

REMARK 26: The given bound on the approximation factor is tight.

Proof. Consider a polygon P made up from k aligned squares of length k, connected via straight lines of width 1 touching the center of sides, such that the squares form a balanced binary tree. P has $n = \Theta(k)$ vertices. We can decontaminate P by using only one barrier of length k to sweep squares in addition to $O(\log k)$ barriers of length 1 at entries of edges. Hence, $sw(P) = \Theta(n)$. The only medial axis nodes with degree greater than 1 are the centers of squares. Therefore, the algorithm uses barriers of length $\Theta(n \log n)$. □

Although the sweep constructed with the previous theorem can be much worse than the optimum, the next remark shows that there are in fact polygons for which it achieves results within $O(1)$ of the optimum. It generally performs well for families of polygons where R_P is small in comparison to the area. Note that Lemma 10 always yields a sweep with bottleneck length at most $w(P)$.

REMARK 27: There are polygons P with $sw(P) = \Theta(R_P \log n)$. For these polygons, Theorem 25 yields an $O(1)$-optimal decontamination sweep.

Proof. A complete binary tree T with n nodes has $ns(T) = \Theta(\log n)$, which follows from [33]. Replacing multiple edges in T with a path each does not increase $ns(T)$ by more than one. If we replace edges using sufficiently long paths, T can be the tree T_P of an STP P as defined in Definition 14 consisting of only unit squares. Arbitrarily long straight paths of unit squares have only $O(1)$ vertices. It holds that $sw(P) = \Theta(R_P \log n)$. □

REMARK 28: We conjecture that an $O(1)$ approximation can be obtained by adding nodes onto edges of the medial axis where the radius is minimal and assigning $w(v) := r(v)$ for all nodes v and then applying the methods from Section 2.3. However, proving that bound on the sweepwidth appears difficult because arbitrarily many nodes with a large radius can be close together such that their circles overlap.

2.5 COMPLEX POLYGONS

In this section, we give a polynomial time algorithm for computing a decontamination sweep of a complex n-vertex polygon P (i. e., a polygon with holes, but no intersecting edges) with bottleneck length $O(\log n \cdot sw(P))$. This is accomplished by rasterizing P into a hexagonal grid represented by an induced subgraph G_P of the triangular lattice graph (see also Section 1.2.1) and then computing a node-search strategy for that grid graph using existing algorithms. We chose the hexagonal grid in order to not have to deal with diagonal "connections" of squares in a Cartesian grid. First, we will show how to rasterize P and prove that $sw(P) = \Theta(ns(G_P))$. Afterwards, a method will be described that allows *compression* of polygons in a way such that G_P has at most $O(n^{14})$ nodes.

2.5.1 *Rasterization*

Before rasterization, the resolution of the grid needs to be determined, i. e., the length of edges in the triangular grid graph or the distance between the centers of adjacent *cells* (hexagons implied by the lattice). It shall be $r_P := R_P/n$ with the intuition that $O(n)$ cells can be guarded with curves of total length $O(sw(P))$. R_P can be calculated using the medial axis of P. Lee [29] notes that the medial axis can be obtained from the *Voronoi edges* of P by removing edges outside of P or incident to reflex vertices. This can be done in time $O(n)$. The Voronoi diagram can be computed in time $O(n \log n)$ for a set of point sites and line segment sites [14]. For completeness, we briefly define the Voronoi diagram.

DEFINITION 29: The *Voronoi diagram* of a set of sites in the plane partitions the plane into regions, called *Voronoi regions*, each belonging to one site. The Voronoi region of a site s is the set of points of p such that s is among the sites closest to p.

In case of point- and line segment sites, the Voronoi diagram is determined by a planar graph such that the edges (referred to as *Voronoi edges*) are made up by the intersection of two Voronoi regions and the nodes by the intersection of at least three [14].

LEMMA 30: The medial axis of a complex polygon can be computed in time $O(n \log n)$.

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.

DEFINITION 31: 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 (i) is adjacent to two cells on opposite sides that are both intersected by at least one edge (dark gray in Figure 10).

(b) *Full cell*: cell inside P without (i) (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 Figure 11).

G_P is then the graph induced by full cells.

The general idea behind blocked cells is that we want to be able to ignore *small features* of the polygon that cannot be reasonably approximated by our cells. Therefore, those regions that can be reasonably approximated by the grid will be decontaminated one after the other, separated by blocked cells. We will argue that there are only $O(n)$ blocked cells. The next lemma allows us to decontaminate connected components of full cells sequentially without having to leave barriers inside them.

LEMMA 32: Let C_1 and C_2 be two connected components of full cells. No simple path W from C_1 to C_2 of empty cells has a distance greater than two to its closest blocked cell if there exists a curve inside P from C_1 through W to C_2 (cf. Figure 14).

Proof. The curve condition is needed since empty cells may be intersected by arbitrarily many edges. Assume such a path exists. First, we will argue that all cells of W would have to lie on the same edge. To that end, we prove that W contains no

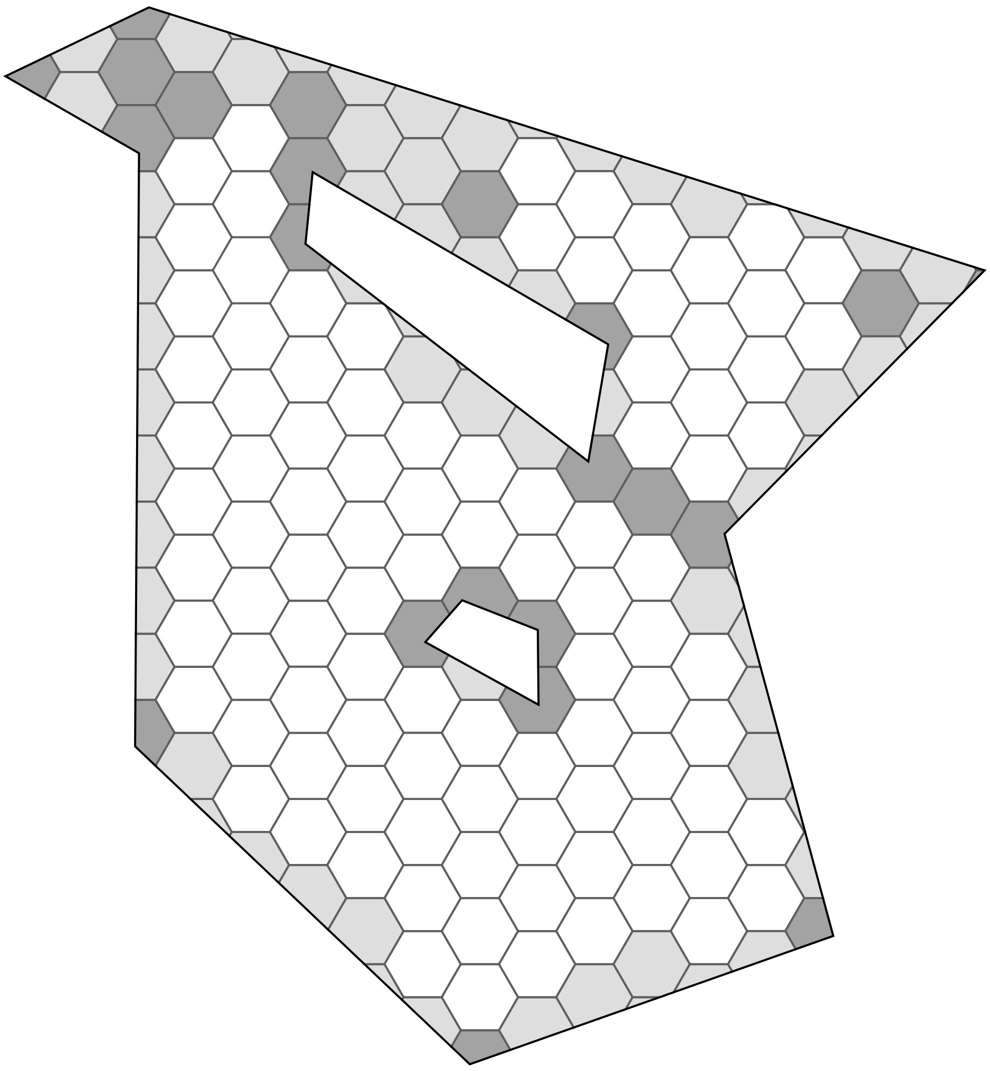


Figure 10: Polygon rasterized into a hexagonal grid (blocked cells dark gray, empty cells light gray, full cells white).

empty cell completely inside P and then show that W does not contain cells lying on different edges of the polygon.

Note that we assume the cells to lie on an edge reachable from the curve via a line segment without intersecting edges of P, e. g., if there was another edge e' directly below the lower edge e of Figure 12 with the curve above e, then the empty cells would not be considered to lie on e'.

W begins in an empty cell u adjacent to a full cell of C_1. Assume that W contains an empty cell v completely inside P. This will be a redefined cell from Definition 31. To reach v from u, W has to pass the first blocked cell between the relevant edges (see fourth example in Figure 13), which would bring W too close to a blocked cell.

Assume that W contains an empty cell v that does not share an edge with u. Let v be the first such cell. Thus, W must either contain adjacent cells not sharing an edge (first example in Figure 13), or a cell containing both edges (second example). If the angle between the edges is small, then W would have passed a blocked node before reaching v (left cells in Figure 12). Otherwise, the vertex of one of those edges would

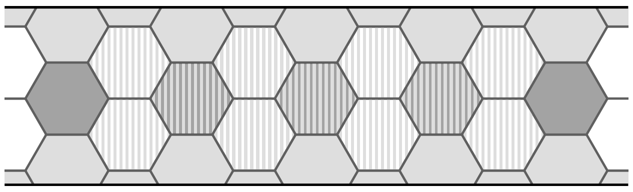


Figure 11: Redefining cells between outermost blocked cells of an edge pair as empty (striped cells).

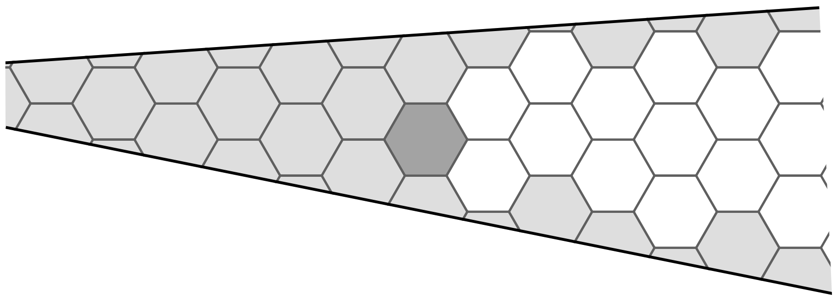


Figure 12: Blocked cells between two polygon edges when rasterizing the polygon into a hexagonal grid.

be too close to W (first two examples). If the angle is too large, then the edges have no adjacent empty cells (third example).

We can now assume that all cells of W lie on one edge, connecting two maximal connected components of full cells. Therefore, the area needs to be constricted near W in a way that not an entire two layers of cells fit on top of W (see right side of Figure 12 where two layers of full cells can be considered to lie above the empty cells on the lower edge), otherwise the components would be connected. Also, that constriction needs to be caused by a vertex at a distance of at most two from W, otherwise we would have a situation similar to Figure 11 or Figure 12 again where W passes through the redefined area between two edges. See Figure 14 for such a situation. Hence, W cannot exist.

□

LEMMA 33: There are $O(n)$ blocked cells.

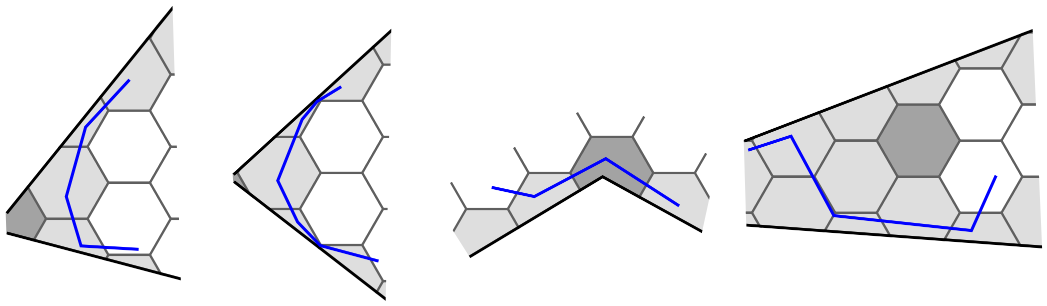


Figure 13: Visualization of the fact that paths of empty cells starting in a full cell and connecting different edges are close to a blocked node.

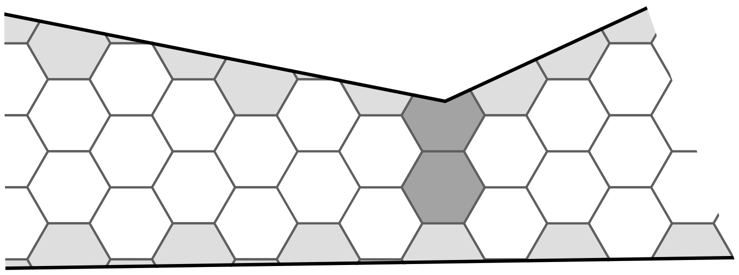


Figure 14: Connected components of full cells separated by blocked cells.

Proof. There are $O(n)$ blocked cells at or near vertices. Now consider a blocked cell v with a distance of at least six to all vertices. It will be between two edges e_1, e_2, as depicted in Figure 12. There are no other edges at a distance of at most three to v, since they would either intersect e_1/e_2 or have a vertex too close to v. Draw a line segment in one of the cardinal directions through the center of v, connecting both edges. These line segments have a length of at most three and therefore do not intersect other such line segments or polygonal edges. Two edges are connected by at most two line segments. Hence, we can contract polygonal edges into nodes of a planar graph by stretching the line segments into curves arbitrarily close to the original edge. By Euler's formula, that graph has at most $O(n)$ edges, each representing a blocked cell. □

The next step is to perform a reduction from node search to sweeping and vice versa, similar to Lemmas 16 and 17. Following the arguments from the proof of Lemma 32, we obtain the following lemma.

LEMMA 34: A curve in P connecting an empty cell u to a full cell v, such that u has a distance of at least three to all full cells, crosses a cell within distance at most two of a blocked cell.

We therefore partition the polygon by cutting away all cells within distance at most two of a blocked node. Barriers will be put up around those cells at the beginning of a decontamination sweep and they will subsequently be cleared. These barriers have total length $O(n \cdot r_P) = O(R_P) = O(sw(P))$. This subdivision can be performed in time $O(n^2)$ by calculating the intersection of the barriers around blocked cells with the polygon.

Components after the subdivision outside those barriers that do not contain full cells are pieces of polygonal edges (see Figure 12) with distance $O(r_P)$ of each other. Hence, those components can be decontaminated with incurring only $O(r_P)$ additional bottleneck length.

For those components containing full cells, the following lemmas perform the reduction to node search.

LEMMA 35: Consider a component P' of the polygon obtained by removing all cells within distance at most two of a blocked node. Let $G = (V, E)$ be the graph induced by the full cells inside that component and $\mathcal{S}$ a node-search strategy clearing G. A decontaminating sweep of P' with bottleneck length $O(r_P \cdot ns(G, \mathcal{S}))$ can be constructed from $\mathcal{S}$ in time $O(|V|)$.

Proof. Whenever a node v is cleared, barriers are put up around all cells within distance at most two of the corresponding cell. Afterwards, the area inside the barriers is swept. The barriers are kept up until the searcher is removed from v. Due to Lemma 34, all empty cells in P' are within distance two of a full cell. Therefore, each curve $f\colon [0,1] \to P'$ passing through a path W containing empty cells can be mapped to a curve $f'\colon [0,1] \to P'$ passing through a path of only full cells W', such that $\max_{t\in[0,1]} d(v(f(t)), v(f'(t))) \leq 2$ where $v\colon P' \to V \cup \{\text{empty cells}\}$ maps points on P' to their cells. In case f connects a contaminated point with a decontaminated point, W' is clearly guarded in G. As barriers are put up at distance 2 of guarded cells, f' passes through a barrier. $\square$

LEMMA 36: Let G, P' be defined as in Lemma 35. Given a decontamination sweep of P' with bottleneck length L, we can construct a node-search strategy $\mathcal{S}$ clearing G with $ns(G, \mathcal{S}) = O(L/r_P)$.

Proof. We follow the idea of Lemma 17. Hence, at any point in time during $\mathcal{S}$, all those nodes shall be cleared whose center is decontaminated (1). To that end, let v be a cell whose center is decontaminated. If v has a neighbor w whose center does not reside in the same decontaminated area as that of v, there is a barrier of length at least $r_P/2$ in v and w. In that case, searchers are put on v and all its neighbors. Hence, $O(L/r_P)$ searchers are used to guard all nodes from (1). The events from Lemma 17 changing the set of nodes that need to be guarded can be handled analogously. $\square$

THEOREM 37: Let $G_1, \ldots, G_k$ be the connected components of the graph induced by full cells. It holds that $sw(P) = \Theta(r_P \max_i ns(G_i))$.

Proof. First, we show $sw(P) = O(r_P \max_i ns(G_i))$. For subdividing P at blocked cells, barriers of length $O(R_P)$ are needed. By definition of R_P, $\max_i ns(G_i) = \Omega(R_P/r_P) = \Omega(n)$. Each part P_i of P corresponding to G_i can be decontaminated using curves of length $O(r_P \cdot ns(G_i))$ due to Lemma 35, given an optimal node-search strategy clearing G_i.

Next, we prove $\max_i ns(G_i) = O(sw(P)/r_P)$. Suppose we have an optimal decontamination sweep. Clearly, each P_i is swept as well. Therefore, Lemma 36 allows for the construction of a node-search strategy using $O(sw(P)/r_P)$ searchers for each G_i. $\square$

The next lemma describes how to compute the categorization of cells in accordance with Definition 31.

LEMMA 38: If P intersects m cells, then all cells intersecting P can be computed and categorized in time $O((m + n) \log n)$.

Proof. The first step is to compute cells intersected by the outline. For each edge e from cell u to v, start in cell u. Then check for each cell in the neighborhood of u if it intersects e. Continue iteratively with those neighbors until reaching v. The number of cells visited per edge is linear in the number of cells intersected by that edge. Hence, that process takes time $O(m + n)$. If a cell is intersected by multiple edges but does not lie on a vertex of the polygon, we save only the first edges wrt. a ray from the center of each neighbor. These are the only edges needed for assigning a blocked cell to a pair of edges.

Next, we compute those cells completely inside P. For each cell adjacent to a cell intersecting an edge, we check if it lies in P. This can be done in $O(\log n)$ for each cell [35], with preprocessing time $O(n \log n)$, the total runtime for that step then being $O((m+n)\log n)$. For each of those cells, we check if it is a full-, blocked- or empty cell in time $O(1)$.

The remaining full cells inside P are computed using a flood fill approach in time $O(m)$. Finally, we need to redefine cells. To that end, we determine the first and last blocked cell for each relevant pair of edges in time $O(m)$ because we can find the associated edge pair in time $O(1)$ for each blocked cell by looking at edges of neighboring empty cells. Redefinition is then performed in time $O(m)$ using a flood fill approach. □

2.5.2 *Computing Node-search Strategies*

After rasterizing, we need to compute node-search strategies for each connected component of full cells. Algorithms from literature do not give a node search strategy directly. Instead we need to take a "detour" via tree decomposition, path decomposition and interval graphs. We define these terms, citing Bodlaender [4].

DEFINITION 39: A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a family of subsets of V, one for each node of T and T a tree such that

- $\bigcup_{i \in I} X_i = V$
- For all edges $\{v, w\} \in E$, there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$.
- For all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The width of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The *treewidth* of G, $tw(G)$ is the minimum width over all possible tree decompositions of G.

DEFINITION 40: A *path decomposition* of a graph $G = (V, E)$ is a sequence of subsets of vertices $(X_1, \dots, X_r)$, such that

- $\bigcup_{1 \le i \le r} X_i = V$
- For all edges $\{v, w\} \in E$, there exists an i, $1 \le i \le r$ with $v \in X_i$ and $w \in X_i$.
- For all $1 \le i \le j \le k \le r$: $X_i \cap X_k \subseteq X_j$

The width of a path decomposition is $\max_{1 \le i \le r} |X_i| - 1$. The *pathwidth* of G, $pw(G)$ is the minimum width over all possible path decompositions of G.

DEFINITION 41: A graph $G = (V, E)$ is an *interval graph*, iff one can associate with each vertex $v \in V$ an interval $I_v = [l_v, r_v] \subset \mathbb{R}$, such that for all $v, w \in V$, $v \neq w$: $\{v, w\} \in E \Leftrightarrow I_v \cap I_w \neq \emptyset$.

The *interval thickness* of a graph G, $\theta(G)$ is the smallest maximum clique size of an interval graph G' that contains G as a subgraph.

With these definitions, we can now show how to compute a node-search strategy for a planar graph.

LEMMA 42: Given a planar graph G with n nodes, a node-search strategy $\mathcal{S}$ with $ns(G, \mathcal{S}) = O(\log n \cdot ns(G))$ clearing G can be computed in time $O(n \log^4 n)$.

Proof. Gu and Xu [19] describe how to compute a tree composition of G with width $\Theta(tw(G))$ in time $O(n \log^4 n)$. Korach and Solel [26] show that $tw(G) \le pw(G) = O(\log n \cdot tw(G))$. Their proof also allows for a linear time construction of a path decomposition, given a tree decomposition. It further holds that $\theta(G) = pw(G) + 1$ [4]. An interval supergraph graph of G' with smallest maximum clique size $pw(G) + 1$ can be constructed in linear time, given a path decomposition. Finally, Kirousis and Papadimitriou [23] show that $ns(G) = \theta(G)$. Their proof directly implies a linear time algorithm to compute a node-search strategy given G'. □

2.5.3 *Polygon Compression*

It remains the problem that the number of cells intersected by the polygon can be arbitrarily large. We will describe an algorithm that allows us to *compress* P such that it fits into a rectangle of size $O(R_P \cdot n^2) \times O(R_P \cdot n^4)$.

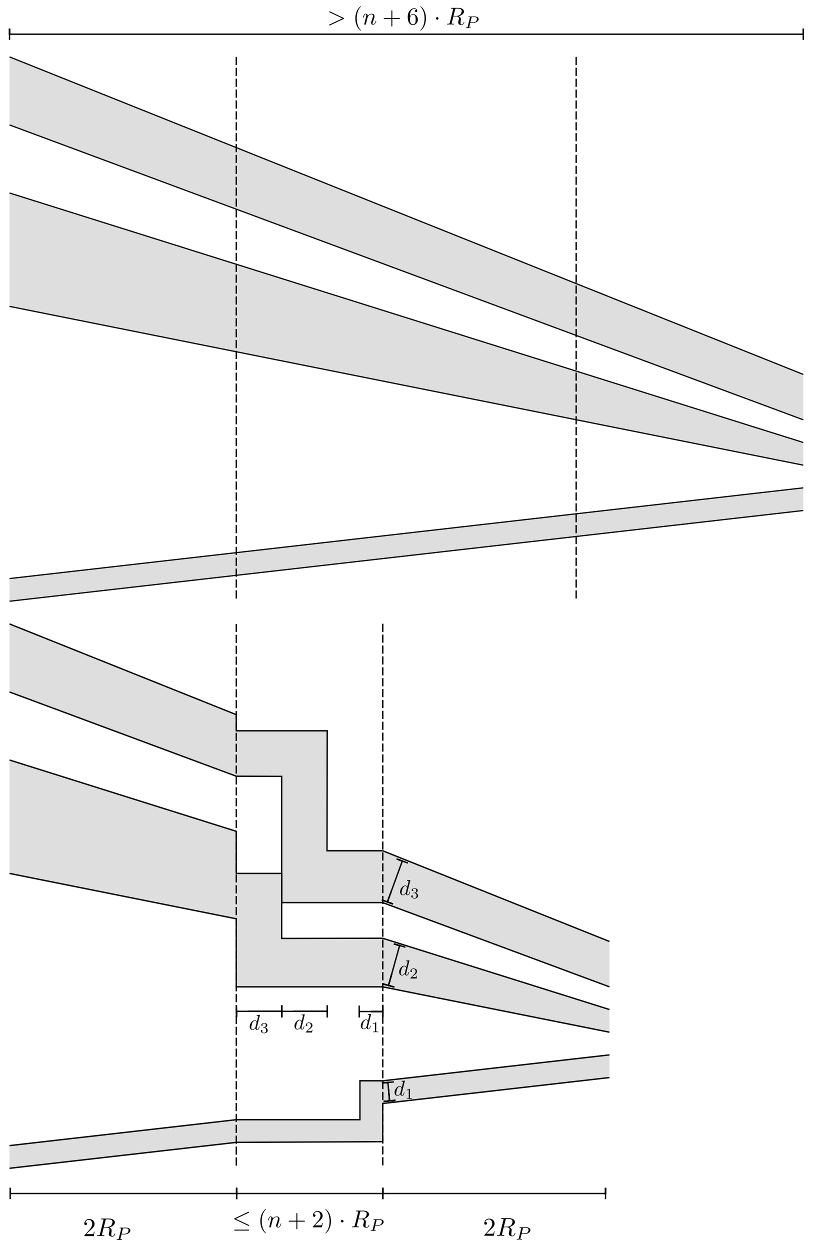


Figure 15: Illustration of polygon compression (d_1, d_2, d_3 are the minimum distance between their respective lines; area inside the polygon gray).

Let $I' := [x'_0, x'_1] \subset \mathbb{R}$, $x'_1 - x'_0 > (n+6)R_P$ such that no vertices of P have their x-coordinate inside I. That interval will look like the top of Figure 15, i. e., there are k pairs of subsegments of edges, bordering part of the polygon. For each pair, we compute their minimum distances $d_1, \ldots, d_k$ 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 Figure 15. 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 $y_i \leq y'_i$. We then replace the lower edge by a path constructed as follows. Begin in (x_0, y_i) and move as far right until reaching either x_1 (see d_1 in Figure 15) or an edge of $i-1$ (see d_3). We can clearly move right until at least $x_1 - \sum_{j=1}^{i-1} d_i \geq x_1 - n \cdot R_P$. Then we move straight up to y'_i and continue to (x_1, y'_i). Since $x_1 - x_2 \geq R_P + 2$, 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 $x_1 - x_0 + (n+1) \cdot R_p \geq R_P$ inside I where all edges have straight segments. We compress I'' to a length of R_P.

We obtain a polygon P' by performing the above steps on P both in x- and y-direction.

LEMMA 43: P' can be computed in time $O(n^4)$ and has $O(n^4)$ vertices.

Proof. The compression steps can be performed in time $O(n)$ for every maximal I', of which there are at most n. In total, $O(n^2)$ time is required for compression in x-direction. The obtained polygon has size $O(n^2)$ since each step introduces at most n new vertices. Similarly, compression in y-direction on that polygon takes time $O(n^4)$ and the resulting polygon P' has $O(n^4)$ vertices. □

The bound on the size after the first compression is tight. Consider a polygon P with $R_P = 1/3$ and n line segments of length n^2 parallel to the x-axis stacked on top of each other and n such line segments of length n next to each other. All long line segments are split once for each small line segment. Afterwards, the polygon has $\Omega(n^2)$ vertices. We conjecture that the polygon has no more than $O(n^3)$ vertices after the second compression.

The next lemma bounds the number of cells the polygon might intersect in the context of Section 2.5.1.

LEMMA 44: P' intersects $O(n^{14})$ cells of radius $r_P = R_P/n^4$.

Proof. After the first compression, the polygon has width at most $O((R_P \cdot n)n)$, since the maximal difference of the x-coordinates of two vertices is $O(R_P \cdot n)$. After the second compression, its height is analogously bounded by $O((R_P \cdot n^2)n^2)$. Therefore, the polygon fits into a rectangle of size $n^6 r_P \times n^8 r_P$ and intersects $O(n^{14})$ cells. □

Since we want to compute sweeps on P', we need to be able to transfer them to P and argue that compressions do not change the sweepwidth by more than a constant factor.

LEMMA 45: Let P' be the result of compressing P once. Then $sw(P') = \Theta(sw(P))$.

Proof. We prove this statement by modifying a sweep while only increasing bottleneck length by a constant factor such that barriers are never kept inside the compressed region; they are only swept through it when both ends are already blocked. This allows us to map sweeps from P to P' and implies $sw(P') = \Theta(sw(P))$.

Consider a single pair of edges inside an interval I' as defined above. For the purposes of this proof, we can treat the area between those edges as two rectangles with dimensions $2R_P \times d_0$ and $2R_P \times d_1$ joined together, where d_0 is the minimal and d_1 the maximal distance between the edges inside I'. This is justified by the fact that the angle between edges becomes arbitrarily small as $n \to \infty$. The rectangles are jointed together directly because we remove all barriers from the compressed interval I.

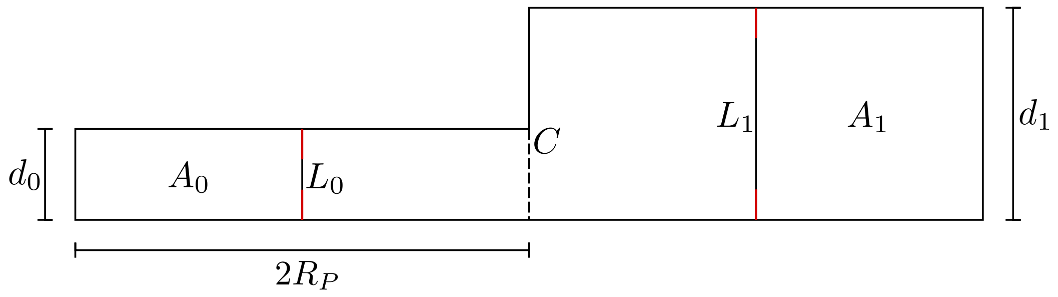


Figure 16: Modeling the area between two edges in a compressed interval as two conjoined rectangles (A_0 (A_1) is the outer half of the smaller (larger) rectangle, $L_{0/1}$ refer to the shown lines, C is the central area, and red lines show to newly placed barriers).

This situation is depicted in Figure 16. We now describe how to transform an optimal sweep in canonical form to remove all barriers from C. Barriers in A_0 and A_1 are kept unchanged. At any point in time, let ℓ be the total length of barriers inside $A := A_0 \cup C \cup A_1$. We place barriers of length 2ℓ on L_0 and L_1 as depicted in Figure 16. This increases the total length of barriers inside A by a factor of at most 8. If $\ell \geq d_1/4$, all paths from A_0 and A_1 to C are blocked. In that case, we sweep C which requires barriers of length at most $2d_1$. We will show that the altered sweep still decontaminates the polygon.

Assume that at some point there is a point z outside C that decontaminated in the original sweep but contaminated in the modified one. Hence, z must been recontaminated at some point. Let p be the recontamination path.

□

With this we have the final theorem.

THEOREM 46: The sweepwidth of a complex n-vertex polygon P can be approximated with a factor of $O(\log n)$ in time $O(n^{14} \log^4 n)$. A decontamination sweep with that bottleneck length can be computed in $O(n^{20} \log^4 n)$.

Proof. R_P is obtained from the medial axis in time $O(n \log n)$ We apply the results from Section 2.5.1 on P'. Using Lemma 38, all cells are categorized in time $O(n^{14} \log n)$. Note that the original value of R_P and not $R_{P'}$ will be used there. A decontamination sweep is calculated for each connected component of full cells with Lemma 42 in time $O(n^{14} \log^4 n)$. By Theorem 37, this gives us a sweep with bottleneck length $O(\log n \cdot sw(P))$ of P'.

To construct a sweep of P, we need to undo compressions. The proof of Lemma 45 describes how to handle barriers inside compressed regions. The constructed sweep uses only barriers of the following types: constructing barriers around a node, sweeping barriers around a node, or sweeping an area of empty cells between two edges. For each pair of barrier and compressed edge, we can easily adapt the barrier to the decompressed polygon in time $O(1)$. We need to do this for $O(n^4)$ edges and then $O(n^2)$ edges, taking time $O(n^{14} \log^4 n \cdot n^4 \cdot n^2)$ (the number of barriers might increase after the first decompression step). □

We conclude this section with an interesting corollary. It is known that recontamination does not help for node search [27], which implies that node search is in $\mathcal{NP}$. This is currently an open problem for decontamination sweeps [22]. Our results imply that recontamination can only improve bottleneck length by a factor of $O(1)$ and that $O(1)$ approximation of sweepwidth is in $\mathcal{NP}$.

COROLLARY 47: $O(1)$ approximation of the sweepwidth of a complex polygon is in $\mathcal{NP}$.

2.5.4 *Faster Algorithm*

The above algorithm can be considered too slow for practical purposes. Therefore, we give an algorithm for computing a sweep in time $O(n \log^4 n)$. We follow the idea from Section 2.4 in that we first compute the medial axis and then perform node search on the graph induced by the medial axis.

THEOREM 48: A decontamination sweep of a complex n-vertex polygon P with bottleneck length $O(\sqrt{n} \cdot R_P) = O(\sqrt{n} \cdot sw(P))$ can be computed in time $O(n \log^4 n)$.

Proof. The planar graph G_P has pathwidth $pw(G_P) = O(\sqrt{n})$ [4]. The rest follows from above and Lemma 42. □

REMARK 49: Similar to Section 2.4, a polygon P with $sw(P) = \theta(\sqrt{n} \cdot R_P)$ can be constructed. Let P be an $(2k+1) \times (2k+1)$ square containing $k \times k$ unit squares spaced equidistantly. It has $n = \Theta(k^2)$ vertices. The medial axis graph is a $(k+1) \times (k+1)$ grid with one additional node at the four corner vertices. The grid has pathwidth $\Theta(\sqrt{n})$ [19].

3 DECONTAMINATION IN THE ROBOT MODEL

In this chapter, we develop algorithms to decontaminate environments with robots as defined in Section 1.2.1. At first, we consider convex environments. An environment is convex if its border nodes define a convex polygon when embedded into the Euclidean plane. There, we will describe our algorithms in more detail introducing the reader to key methods while allowing for a more abstract reasoning in later sections. Afterwards, we present algorithms for environments with and without holes. Our algorithms work with just a single robot and multiple robots can improve their runtime. Finally, we show that deciding whether an environment can be decontaminated using a certain number of tiles is $\mathcal{NP}$-hard.

3.1 CONVEX ENVIRONMENTS

Convex environments always have one of the following shapes:

DEFINITION 50: Consider a set of $k \in \{3,4,5,6\}$ lines through nodes of the triangular grid graph, each in direction N, NE, or NW. The nodes inside their intersection form a shape, as shown in Figure 17.

– Three lines in directions N, NE, and NW form an equilateral *triangle*.
– Two sets of parallel lines form a *parallelogram*.
– Two parallel lines, plus one line in each remaining direction, form a *trapezoid*.
– Five lines, at most two in each direction, form a *pentagon*.
– Six lines, two in each direction, form a *hexagon*.

Usually, we will assume parallelograms to be aligned with directions N and NE, consisting of ℓ columns (in N direction) and h rows (in NE direction). For trapezoids, we define the height h as number of nodes in a shortest path between the parallel lines (3 in the example).

Before describing the algorithms, we need to state some assumptions. Initially, all tiles are arranged in a parallelogram that is placed on a line of the shape (see Figure 20 (1)). We will also refer to them as *stash*. Their number shall be at least the minimal number of tiles needed for the algorithm while not exceeding it by more than a constant factor. The robots are placed on that parallelogram, filling columns N to S, W to E. The northernmost robot of the first column shall be *leader* and all robots share a common north and chirality. We consider this assumption to be reasonable as leader election and shape formation are outside the scope of this thesis. One can also imagine that most practical contexts would allow for a certain amount of control on how robots and tiles are inserted (e. g., when injecting nanomatter for surgery).

For reference, shape formation has been studied by Gmyr et al. [15]. They describe how to construct intermediate shapes without holes and from them simple convex shapes. Note that their methods would have to be altered slightly in order to deal with finite environments as they operate on an infinite grid graph. Leader election in a connected configuration of m robots that do not share a common chirality or north

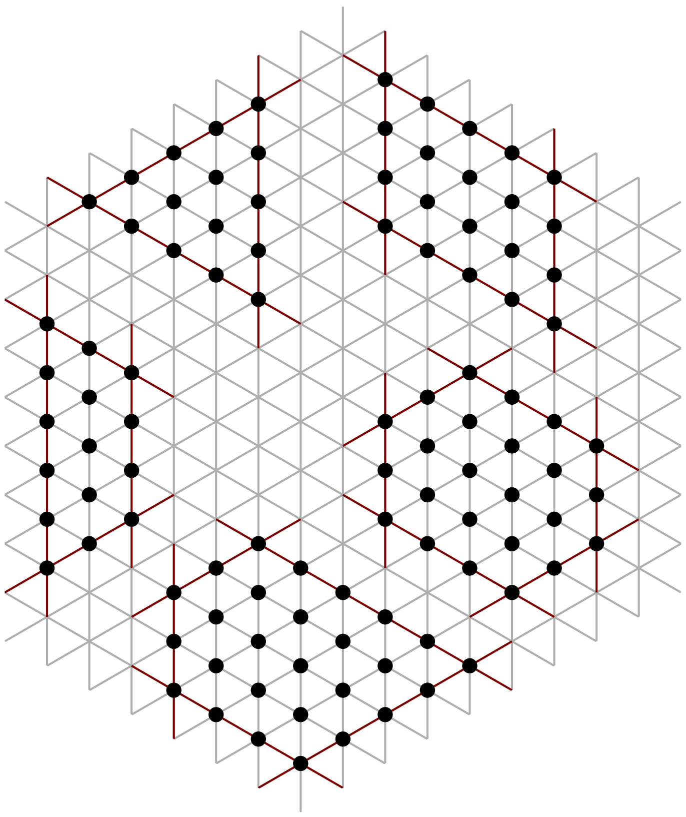


Figure 17: Convex shapes in the triangular grid graph defined as the inside of an intersection of lines.

can be performed in $O(m)$ rounds [10]. Therefor robots need to generate random bits which is not unrealistic in practice. These results were obtained for the amoebot model but are directly transferrable to our model since movement of robots is not required.

Furthermore, the environment is guaranteed to have the shape for which we describe the algorithm. Even if that were not the case, it would be easy to verify the shape of the outline beforehand, using the stash as a marker when traversing the outline to know when a traversal is complete. Holes inside the environment can be recognized during the execution of the algorithm. Lastly, we assume the environment to be sufficiently large, since the algorithms do not always take trivial edge cases such as $h = 1$ into account.

3.1.1 *Parallelograms*

The parallelogram will be decontaminated by sweeping a line of tiles across its longer side as depicted in Figure 18. That line consists of h tiles. The next lemma shows that this optimal.

LEMMA 51: A parallelogram $\mathcal{E}$ consisting of ℓ rows and h columns, such that $\ell \geq h$, has $td(\mathcal{E}) \geq h$.

Proof. We look at a sequence of tile movements decontaminating $\mathcal{E}$ and argue that it uses at least h tiles. Consider the first tile movement after which every row and column contains at least one decontaminated node. W.l.o.g., a tile is moved into a column C of only contaminated nodes. Afterwards, every row contains a tile: Only one node of C is decontaminated. It contains a tile. Since no row is fully contaminated, all other rows also contain a tile that separates the contaminated node of C in that row from the decontaminated node. □

The above lemma allows us to give a lower bound on the number of rounds needed for the decontamination of simple environments by a single robot.

COROLLARY 52: There exists an n-node environment $\mathcal{E}$ that cannot be decontaminated in less than $\Omega(n^{3/2})$ rounds by a single robot.

Proof. Consider an environment $\mathcal{E}$, consisting of two $k \times k$ parallelograms, joined together by a line of length k^2. $\mathcal{E}$ has $n = \Theta(k^2)$ nodes. If the stash is inside one of the parallelograms, then k tiles need to be transported to the other one, which takes $\Omega(k^3) = \Omega(n^{3/2})$ rounds. □

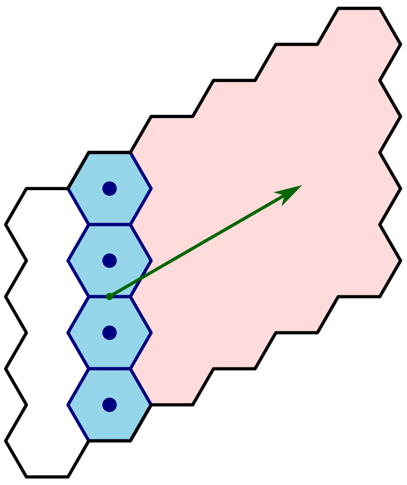


Figure 18: Decontaminating a parallelogram by sweeping it with a line of tiles.

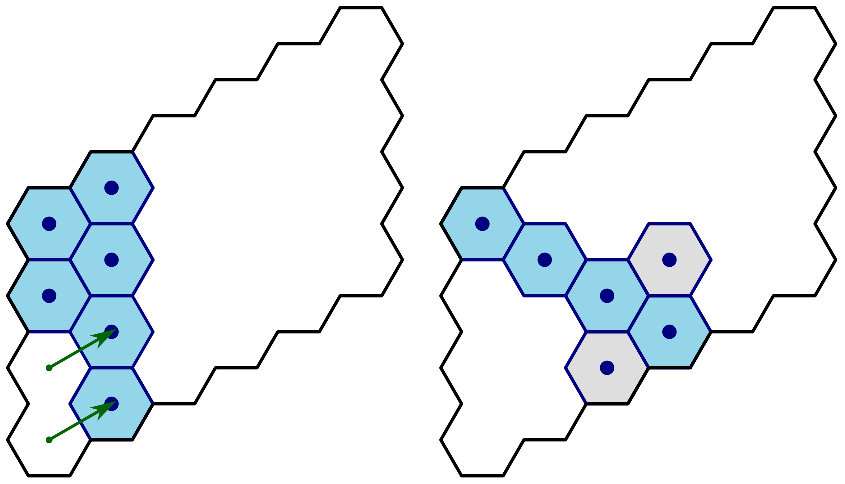
The algorithm will consist of three fundamental steps:

1. Determine the shorter side of the parallelogram.
2. Construct a line along the shorter side.
3. Move the line across the parallelogram.

First, we describe the algorithm for a single robot. To determine the shorter side, the robot builds a line as depicted in Figure 19a. If the number of tiles is insufficient to build a full line, we attempted to build the line along the longer side and need to rebuild it on the other side. We continue by sweeping the line across the parallelogram. Note that the line might still be parallel to the longer side, if excess tiles are present, which does not affect the results. Alternatively, the shorter side can be determined exactly by building a diagonal as in Figure 19b. It always ends on the longer side.

LEMMA 53: A single robot can construct a line as shown in Figure 19b in $O(h \cdot \ell)$ rounds.

Proof. This line is built by repeatedly taking a tile from the stash and moving along the outline of the parallelogram to the most NW node. From there, the robot moves



(a) Building a line with excess tiles to sweep the parallelogram.
(b) Detecting the shorter side of a parallelogram by building a diagonal.

Figure 19: Preliminary steps before sweeping the parallelogram.

SE as far as possible, over the line that was already built, and drops the tile or halts if the line has reached another side of the parallelogram.

Afterwards, a new tile needs to be picked up. If the line is connected to the stash, the robot can take any northernmost tile in a column of the stash intersecting the diagonal (see gray tiles in the figure). This guarantees connectivity, since the tiles south of that tile are either connected to a border node or the line. Otherwise, the robot moves back over the line to the stash and takes any northernmost tile from there.

To bring a tile from the stash to the end of the line, $O(\ell)$ rounds are needed, for a total of $O(h \cdot \ell)$ rounds. □

Consider the task of building the line for sweeping. W.l.o.g., the longer side is parallel to the axis from SW to NE. If there are excess tiles, we build $O(1)$ additional lines, the first of which possibly being partial. The empty space in the partial line is decontaminated as indicated by the green arrows.

LEMMA 54: A single robot can construct a line with excess tiles as shown in Figure 19a in $O(h \cdot \ell)$ rounds.

Proof. This can be done similarly to Lemma 53. □

Finally, the parallelogram is decontaminated by moving the line NE until it reaches the outline.

LEMMA 55: A single robot can decontaminate the parallelogram in $O(h \cdot \ell)$ rounds using h tiles, which is asymptotically optimal.

Proof. We construct the line using Lemma 54 and move it to the eastern border of the parallelogram. To move the line east by one step, the robot moves each tile NE, starting with the southernmost tile. This preserves connectivity and leaves the column formerly occupied by the line decontaminated. Each movement of the line takes $O(h)$ rounds, for a total of $O(h \cdot \ell)$ rounds.

To decontaminate the environment, at some point a tile needs to be placed onto each of the $h \cdot \ell$ nodes, requiring at least one round per node. Therefore, $\Omega(h \cdot \ell)$ rounds are required for decontamination. □

In the following, we will discuss how to use multiple robots to speed up the process of decontaminating a parallelogram. Recall that the only assumption we can make about the activation order of robots is that any robot will be activated after a finite amount of time. However, a round is only over after each robot has been activated at least once. There shall be $m \leq h$ robots.

LEMMA 56: *m robots can construct a line as shown in Figure 19a in $O(h \cdot \ell / m)$ rounds.*

Proof. The steps of the algorithm are illustrated in Figure 20. Initially, the robots are located on the stash (1). If the stash already borders the western border of the parallelogram, the robots shall start on the second column. The leader (red) begins by picking up its tile. Afterwards, the robot to its south may pick up its tile, continuing until all robots in the column have picked up their tile (2). The last robot in the column then signals to the leader that they may start moving. Signal passing can easily be implemented by reading and modifying states of adjacent robots. Before the last robot leaves the column, it signals to the southernmost robot of the next column that its column may start moving as well, if the next column exists and is already occupied. The last column needs to be handled differently to ensure connectivity; the northernmost robot starts and moves along the column's side, as indicated in (5). The robot signals its former southern neighbor to follow upon reaching its SW.

The robots with a tile travel to the westernmost empty column and fill it with tiles from N to S (3). Robots from different column may travel concurrently. Once the last robot has transported its tile, it notifies the leader (4). If the robots are spread across two different columns, the last robot will need to move to the leader (6).

Afterwards, all robots travel back onto the stash (5) and continue as described previously. In case not all robots fit onto the stash, they wait next to the stash. Once the stash has been cleared (6), the leader establishes the final line (7). Robots waiting where the stash once was are notified and all robots position themselves next to the line as in (8) to prepare for moving the line.

Each movement phase, of which there are $O(h/m)$, can be carried out in $O(\ell)$ rounds due to the concurrent movement of robots: We show that m robots can travel ℓ steps in time $O(\ell + m)$ inductively, by proving that robot k ($k = 0$ being the first robot) takes at least t steps in $t + k$ rounds, for any $t \geq 0$. For $k = 0$, the first robot clearly takes at least t steps in t rounds. If robot $k - 1$ takes $t + 1$ steps in $t + k$ rounds, then robot k can take t steps in $t + k$ rounds.

The centralized nature of the algorithm makes the absence of locking conditions obvious. Note that the same construction can be performed largely without synchronization by the leader. However, there the number of rounds required is less obvious. □

The final step of the algorithm is to sweep the line across the parallelogram. We will describe next how to do that with multiple robots and linear speedup.

LEMMA 57: *m robots can move a line across the parallelogram in $O(h \cdot \ell / m)$ rounds.*

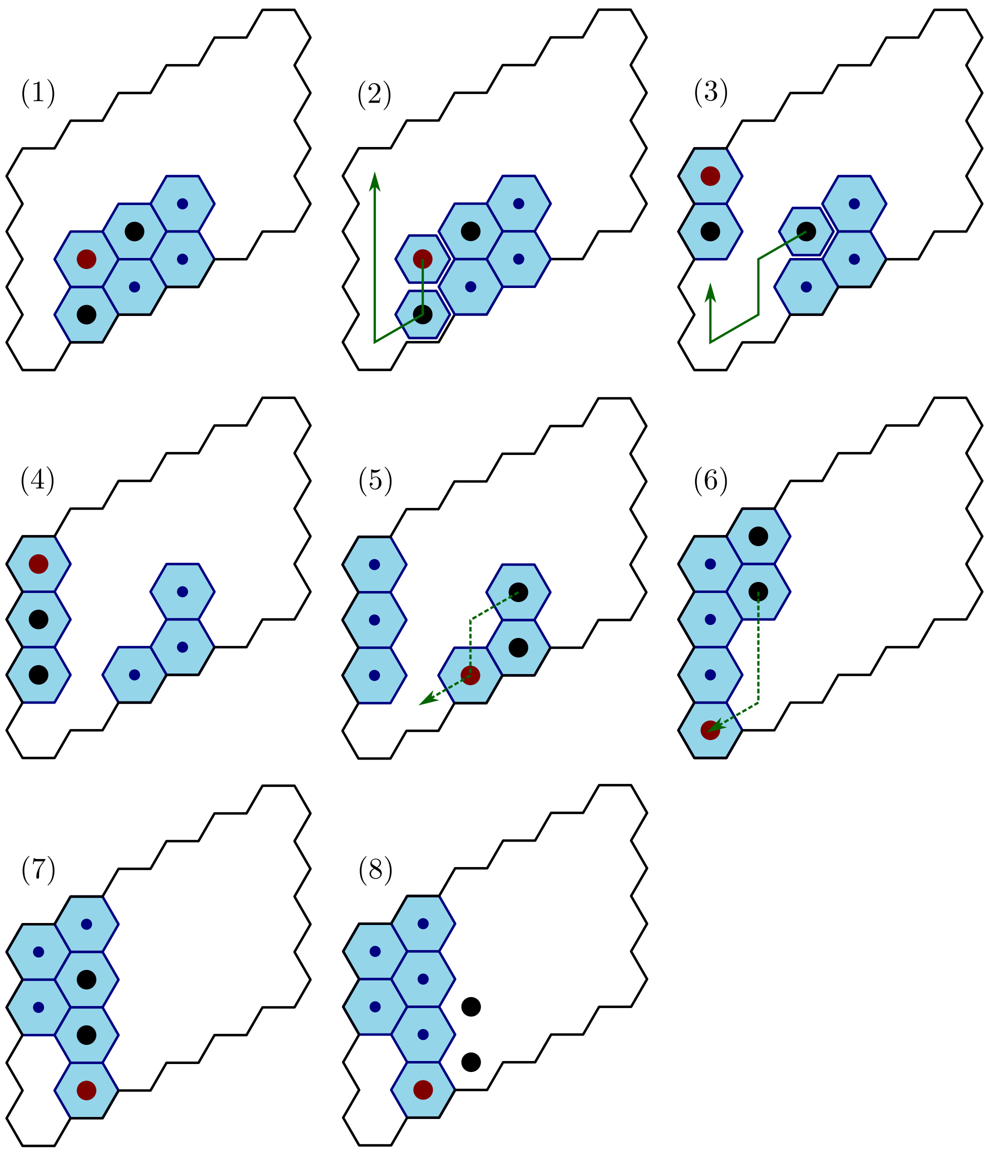


Figure 20: Constructing a line with multiple robots.

Proof. The idea is to execute the line movement from Lemma 55 in a pipelined manner. Initially, the robots shall be arranged as in Figure 20 (8). Each robot operates in two phases. First, it traverses the line from N to S and moves each tile NE. Afterwards, it moves S along the line's eastern border. Consequently, the robots essentially cycle around the line, as indicated by the red arrows in Figure 21.

A robot r in the first phase checks if there is a tile at NW or N. It waits until there is an empty tile at N (see robot a). Then r picks up its tile. If the node at NE is unoccupied, r moves and places its tile there (robot b has just done that). Otherwise, it passes its tile to the robot r' at NE and then moves N (see robot c). It further modifies the state of r' to indicate to a robot N of r' (see robot d) that it may not move to the NE of r'. There, it waits until r' has placed the tile. r' places the tile and moves NE, if it is not in the easternmost column, and continues in the second phase.

If there is no tile at NW or N, r has reached the line's northernmost tile. Then it waits until the node at NE is empty and picks its tile up and places it there. If there

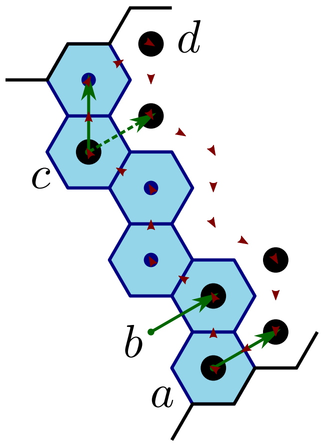


Figure 21: Moving a line across the parallelogram with multiple robots.

is still an empty node at NE, i. e., the sweep is not yet finished, r moves there and goes into the second phase. Otherwise r halts.

Robots in the second phase just follow the line S until they either reach the line's southernmost tile and switch to the first phase, or until they are north of a tile on a border node and move S waiting for the completion of the sweep which will be accomplished by the robots remaining on the line.

The line and its eastern neighborhood (see red path in Figure 21) is never fully occupied by robots because $m \leq h$. Hence, there is always progress. Note that robots only ever have to wait more than $O(1)$ rounds for the robot directly in front of them along the path around the line. Thus, the arguments from the proof of Lemma 56 can be applied to show that all robots can traverse the line once as described above in $O(h)$ rounds, halting in their starting position relative to the line (i. e., arranged as in Figure 20 (8)). Afterwards, the line was moved by m steps. Clearly, this also works if the robots do not halt after one traversal but continue until the sweep is finished. Hence, $O(h \cdot \ell / m)$ rounds are needed in total. □

THEOREM 58: *m robots can decontaminate a parallelogram in $O(h \cdot \ell / m)$ time using h tiles, which is asymptotically optimal.*

Proof. First, a line is built using Lemma 56. If the number of tiles is insufficient, i. e., the line was built on the wrong side, it is rebuilt at an adjacent side of the parallelogram. Afterwards, the line is swept across the parallelogram using Lemma 57.

Optimality follows from the fact that $h \cdot m$ nodes need to be decontaminated which corresponds to that number of activations of a robot. With m robots, $\Omega(h \cdot \ell / m)$ rounds are needed. Note that having more than h robots could not improve the runtime since $\Omega(\ell)$ rounds are always needed for transporting a tile from the stash to the furthest node. □

3.1.2 *Triangles and Trapezoids*

Triangles and trapezoids can be decontaminated similarly to parallelograms, with the key difference that the line is shortened by one after each step of the sweep, as depicted in Figure 22.

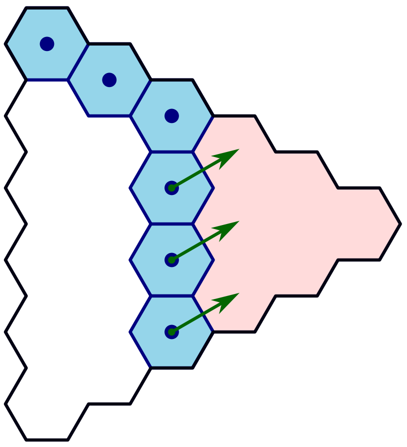


Figure 22: Decontaminating a triangle by sweeping it with a line of tiles.

LEMMA 59: A single robot can decontaminate a triangle with side length h in $O(h^2)$ rounds using h tiles. $\Omega(h)$ tiles are always required.

Proof. First, the line is built as in Lemma 54. The line is then moved across the shape, similar to Lemma 55, with the difference that the northernmost tile of the line is not moved NE (see Figure 22).

A triangle with side length h contains an $\lfloor h/2 \rfloor \times \lfloor h/2 \rfloor$ parallelogram. Hence, $\Omega(h)$ tiles are required by Lemma 51. □

We conjecture that h tiles are needed to decontaminate a triangle of side length h.

To decontaminate triangles with multiple robots, we need to ensure that robots leave the line after traversing it, once the line's length has reached m because our pipelined sweeping algorithm only achieves the desired speedup if the number of robots does not exceed the length.

THEOREM 60: m robots can decontaminate a triangle with side length h in $O(h^2/m)$ rounds using h tiles, which is asymptotically optimal.

Proof. Assume the triangle to be oriented as in Figure 22. After building the line as in Theorem 58, a robot is placed on the triangle's NE border, at distance m from the easternmost node to mark the point where robots have to leave the line after traversal instead of continuing to move around it. This can be done in $O(h)$ rounds by moving the robots along the outline in counterclockwise direction until the leader has reached the easternmost node. The other robots will build up along the NE side, forming a line. The last robot in that line shall stay there.

We can now perform the sweep as in Lemma 57. When the line reaches the marker, robots leaving the line move onto the tile that is left behind and signal to the next robot finishing its traversal to do the same. The marker moves out of the way once it encounters the first robot. Thus, the sweep is completed in $O(h^2/m)$ rounds, which is asymptotically optimal as previously. □

COROLLARY 61: m robots can decontaminate a trapezoid with height h and longest side length ℓ using h tiles in $O(h \cdot \ell/m)$ rounds, which is asymptotically optimal.

Proof. A parallelogram can be obtained from the trapezoid by cutting off a triangle of height h at one end. Therefore, a line is moved across the trapezoid like on a parallelogram until it reaches the triangle where it continues as described in Theorem 60. □

3.1.3 *Pentagons and Hexagons*

Consider a pentagon or hexagon. Let h be the minimal distance between parallel lines and ℓ the maximal distance between any two nodes on the shape. It contains $\Omega(h \cdot \ell)$ nodes. For decontamination, we subdivide the environment into simpler shapes using diagonals from corners, and then decontaminate these shapes separately.

THEOREM 62: m robots can decontaminate a pentagon or hexagon in $O(h \cdot \ell / m)$ rounds using $O(h)$ tiles, which is asymptotically optimal.

Proof. The first step is to build a line along one of the sides with a length of at most h using Lemma 56. From then on, we can view the line as a side and never move its tiles. Only the tiles behind the line will be used. This makes efficient construction of lines with multiple robots trivial since there is no stash inside the shape.

We subdivide a pentagon using a single line, and a hexagon with potentially two lines, as depicted in Figure 23. The construction of these lines is attempted without regard to their length. If the number of tiles is insufficient, the lines are removed again and reconstructed in a different direction (see the red dotted line). The resulting shapes between the lines are triangles, parallelograms, and trapezoids. They are decontaminated separately using $O(h)$ tiles. □

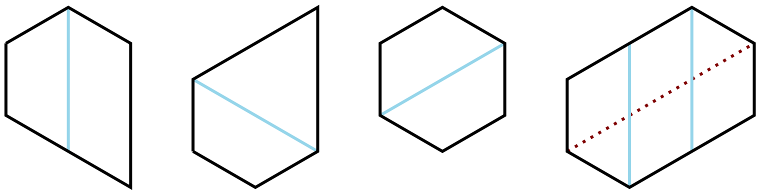


Figure 23: Subdividing pentagons and hexagons into triangles, parallelograms, and trapezoids.

3.2 SIMPLE ENVIRONMENTS

For the decontamination of simple environments, we will use a strategy similar to that proposed in Section 2.4 for simple polygons. Let s be some border node. The remaining nodes are partitioned into *layers* by their distance from the *source* s, as depicted in Figure 24.

LEMMA 63: The connected components (or *barriers*) of a layer are lines starting and ending on border nodes with only $120°$ angles in the direction of the previous barrier.

Proof. This is clearly true for the first layer. Now consider some barrier U and a walk W around U in the underlying lattice graph. If we exclude the previous barrier from

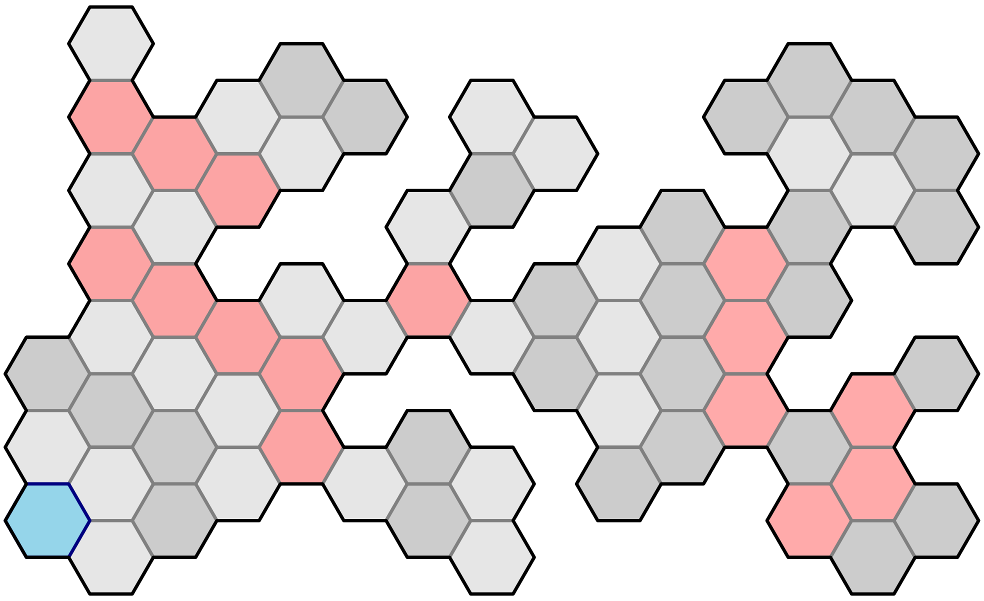


Figure 24: Layers in a simple environment such that the nodes of the same layer have equal distance to the source node (blue). Connected components of layers such that the neighboring nodes of the next layer are not connected are colored red.

W, then W is a line with only 120° angles in the direction of U. The barriers $V_1, \dots, V_k$ in the next layer adjacent to U are the connected components of the intersection of W and $\mathcal{E}$. Hence, they start and end on a border node and have only 120° angles. Since the environment is simple, the neighborhoods of two barriers in the next layer are disjoint. □

The decontamination starts by placing a tile on the source node s. Afterwards, tiles are placed on the next layer to form a new barrier and subsequently removed from the previous layer. Repeating this procedure, the next layer might be disconnected at some point. Let U be the set of nodes of a barrier, such that the barriers $V_1, \dots, V_k$ are the connected components of the next layer. In the example, U would be the first red barrier and V_1, V_2, V_3 the gray barriers adjacent to it. The removal of U would disconnect $\mathcal{E}$ such that the V_i are part of different connected components because $\mathcal{E}$ is simple. These components are decontaminated recursively while the barrier U remains. Similar to Algorithm 1, the V_j with the longest outline will be decontaminated last. U is removed after the construction of barrier V_j.

Before describing these steps in more detail, we need to discuss how to handle the stash. It is assumed to be connected and placed somewhere on the border, such that the nodes not containing tiles of the stash are connected. The barrier formed by those nodes containing stash tiles that also have a neighboring empty node (i. e., the nodes separating the stash from the rest of the $\mathcal{E}$) will be treated just like the outline. Thus, we assume that the number of tiles behind that barrier is sufficient to perform decontamination. The barrier is *marked* using local tile patterns on its inside, near the start and end. We assume the stash to be of sufficient dimensions for this. We will also use tiles to mark certain barriers. These patterns are defined by their shape and number of tiles, e. g., a single tile on a barrier or two tiles. We will not always define them formally, but it will be obvious that they can be constructed.

As a further simplification, we assume that environments are free of *bottlenecks*, i. e., nodes whose removal would disconnect the neighborhood locally (e. g., the single red barrier in Figure 24). This is equivalent to 2-connectivity for simple environments. Thus, nodes are only visited once when traversing the outline, which makes traversal of the outline by multiple robots trivial[1] and allows us to mark positions on the outline with tiles easily. We further require that the stash is placed in a way such that treating it as part of the outline does not lead to bottlenecks.

The next lemmas show that the length of a barrier is bounded by $O(td(\mathcal{E})) = O(\sqrt{n})$.

LEMMA 64: It holds that $O(td(\mathcal{E})) = O(\sqrt{n})$.

Proof. It is trivial to transform a node-search strategy $\mathcal{S}$ on $\mathcal{E}$ into a sequence of tile movements, such that $ns(\mathcal{E}, \mathcal{S})$ tiles decontaminate $\mathcal{E}$. Since $\mathcal{E}$ is planar, $ns(\mathcal{E}) = O(\sqrt{n})$ [4]. □

LEMMA 65: If the algorithm outlined above constructs a barrier of length ℓ, then $td(\mathcal{E}) = \Omega(\ell)$.

Proof. The idea is that barriers only increase in length by $O(1)$ in each layer. Therefore, a hexagon with radius $\Omega(\ell)$ fits into the area that is taken up by the barriers growing to length ℓ.

For a barrier of length ℓ to be constructed, the barrier on the previous layer had to have length at least $\ell - \delta$, for some sufficiently large $\delta = O(1)$. Therefore, there are connected barriers $V_1, \ldots, V_k$, $k = \Theta(\ell)$ on subsequent layers such that $|V_{k-i}| \geq \ell - i\delta$ and $|V_1| \geq \ell/2$ (these barriers are usually not present at the same time during decontamination). Note that barriers may become shorter or separated in the next layer. By removing nodes from $\mathcal{E}$, we can ensure that $|V_i| \leq |V_{i+1}|$ and that nodes in V_i for $2 < i < k$ are only adjacent to nodes in V_{i+j}, $j \in \{-1, 0, 1\}$. Consequently, only the nodes in V_i, $i < k$ at distance at most δ from the end of their barrier are on a border node. Also, if $v \in V_i$, $w \in V_{i\pm 1}$ and v has minimal distance d_v to a border node in V_i, then $d_w \geq d_v - \delta$.

Let $u \in V_{\lfloor k/2 \rfloor}$ be a node in the middle of the barrier. u has a distance of at least $\ell - k\delta/2 - \delta$ to the border nodes in $V_{\lfloor k/2 \rfloor}$. Consider a path from u to a border node. It has length $\Omega(\ell)$ because the distance to a border node in the current layer can only decrease by at most δ with each step. Thus, a regular hexagon with radius $\Omega(\ell)$ fits into $\mathcal{E}$. □

In the following, we describe how to compare the lengths of outlines with a single robot. Since robots have only constant memory size, we cannot just count the number of nodes. Let $ol(\mathcal{E})$ be the length of the environment's outline, i. e., the number of border nodes in $\mathcal{E}$. Note that for a partial environment $\mathcal{E}'$ in $\mathcal{E}$, we count only those border nodes of $\mathcal{E}'$ towards $ol(\mathcal{E}')$ that are also border nodes in $\mathcal{E}$.

LEMMA 66: Consider a barrier U with disconnected next barriers $V_1, \ldots, V_k$, as described above. Let $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be the connected components obtained from $\mathcal{E}$ after removing U such that V_i is in $\mathcal{E}_i$, and $\ell_1 \geq \cdots \geq \ell_k$, $\ell_i = ol(\mathcal{E}_i)$. A single robot can determine $\mathcal{E}_1$ in $O(\ell_2 \cdot (|U| + \hat{\ell}))$ rounds, where $\hat{\ell} := \sum_{i=2}^{k} \ell_i$.

1 Protocols for handling bottlenecks when traversing the outline with multiple robots can be devised but we were unable to construct one with acceptable slowdown.

Proof. We place a tile on each outline, next to the first node of the corresponding barrier V_i. Then, the robot repeatedly moves each tile forward once until all but one tile have reached their barrier V_i again. Forward movement of tiles is well defined due to 2-connectivity of $\mathcal{E}$. Each iteration can be accomplished in $O(|U| + \hat{\ell})$ rounds by moving along U and that part of the outline of $\mathcal{E}$ containing the outlines of the $\mathcal{E}_i$. If $\ell_1 = \ell_2$, then the tile reaching its barrier last is declared to be on $\mathcal{E}_1$. Otherwise, only the tile on $\mathcal{E}_1$ has not reached its barrier again after ℓ_2 iterations. Hence, $\mathcal{E}_1$ is determined after $O(\ell_2(|U| + \hat{\ell}))$ rounds. □

Using this algorithm, we can describe how to decontaminate a simple environment.

THEOREM 67: A single robot can decontaminate a simple n-node environment $\mathcal{E}$ in $O(n^2)$ rounds using $O(\log n \cdot td(\mathcal{E}))$ tiles.

Proof. In the following, we first describe the algorithm and then analyze the number of rounds and tiles required.

We use the algorithm outlined at the beginning of this section. This is trivial until a barrier U has multiple next barriers. In that case, we use Lemma 66 to determine and mark the barrier V_j associated with the longest outline. Note that the robot only constructs the next barriers V_i once it starts decontaminating the associated outline. Since barriers contain at least two nodes, there is sufficient space to place a tile as marker that can be distinguished from a barrier. We distinguish barriers under construction and markers by placing markers on the right and constructing barriers from the left.

Now, we want to apply the algorithm recursively on all V_i, $i \neq j$. For this, the robot needs to access the stash when the length of subsequent barriers changes. The robot can find the stash by simply traversing the outline of $\mathcal{E}$. It returns by following the path of barriers, traversing the outline of $\mathcal{E}$ between barriers. We allow the robot to detect the path locally by keeping the first barrier after a split during recursion on the associated outline. This takes $O(n)$ rounds.

After recursion on V_i is finished, we remove that barrier, which can be done in a way similar to building a barrier. The previous barrier when returning from recursion is always the first one that can be reached by following the outline. We remove barrier U once only V_j remains, which is then moved forward until the next barrier has multiple components or $\mathcal{E}$ is fully decontaminated.

For building barriers, $O(n^2)$ rounds are sufficient since each node is part of one barrier and bringing a tile there from the stash takes $O(n)$ rounds. Tiles used for markers can also be transported in $O(n^2)$ total rounds since $O(n)$ markers are used throughout the execution.

For comparing outline lengths, let us split the number of rounds from Lemma 66 into two parts, namely traversals of the barrier U and traversals of $\mathcal{E}_i$'s outlines. Each node is only part of a single barrier, which is clearly traversed at most $O(n)$ times for comparing outlines. Hence, traversals of barriers account for $O(n^2)$ rounds.

For traversals of the outlines, consider the *main path*, i. e., the path of barriers with longest associated outlines, starting in the source node, and disregard all other recursions for now. This corresponds to only considering the recursive calls from line 9 in Algorithm 1. From each barrier U of the main path, only one comparison is run. Also, the outlines that are split off by U and not part of the main path are not used for further comparisons from the main path. Hence, the total number of nodes traversed by the tiles from Lemma 66 for barriers U of the main path is at most $\ell = ol(\mathcal{E})$. Therefore, the robot uses $O(\ell^2)$ rounds moving along the outlines of $\mathcal{E}_1, \dots, \mathcal{E}_k$. Thus, $O(\ell^2)$ rounds are used for the main path.

Let $T(\ell)$ be the maximal number of rounds used for traversing the outline when comparing outlines. For the main path, at most $\alpha\ell^2$ rounds are needed for some constant $\alpha \geq 1$. For environments $\mathcal{E}_1, \dots, \mathcal{E}_k$ split off from the main path, we have $\ell_i \leq \ell/2$ for $\ell_i = ol(\mathcal{E}_i)$ and $\sum_{i=1}^k \ell_i \leq \ell$. Hence, $\sum_{i=1}^k T(\ell_i)$ rounds are used for comparing outlines in $\mathcal{E}_1, \dots, \mathcal{E}_k$. Therefore, $T(\ell) \leq \max_{\text{valid } k, \ell_1, \dots, \ell_k} \sum_{i=1}^k T(\ell_i) + \alpha\ell^2$. We will show by induction that $T(\ell) \leq 2\alpha\ell^2$. For $\ell = 1$ this is obvious, given $T(1) = 1$. For $\ell > 1$, we have $\sum_{i=1}^k T(\ell_i) \leq \sum_{i=1}^k 2\alpha\ell_i^2 \leq \alpha\ell^2$, since $\ell_i \leq \ell/2$ and $\sum_{i=1}^k \ell_i \leq \ell$. Thus, $T(\ell) \leq 2\alpha\ell^2$, which means that $O(n^2)$ rounds are used for comparing outlines.

Regarding the number of tiles needed, notice that there is never more than one barrier on the main path. Also, the length of outlines is at least halved in recursions. Therefore, at most $\log n$ barriers are present at the same time. By Lemma 65, we need $O(\log n \cdot td(\mathcal{E}))$ tiles in total. □

REMARK 68: Similar to Remark 26, we can argue that the approximation factor $O(\log n)$ is tight and that there are environments where an approximation factor of $O(1)$ is achieved.

Proof. The example from Remark 26 can be embedded as an environment $\mathcal{E}$ with $n = \Theta(k^c)$ nodes for some constant c. Note that we need lines of width 2 for 2-connectivity. It holds that $td(\mathcal{E}) = \Theta(k)$, but our algorithm needs $\Theta(k \log k) = \Theta(k \log n)$ tiles.

Similarly, we can use lines to build a tree-like environment $\mathcal{E}$ with n nodes such that the graph $\mathcal{E}$ has $pw(\mathcal{E}) = \Theta(\log n)$. Our algorithm will only use $O(\log n)$ tiles. □

We conjecture that the time bound of $O(n^2)$ can be improved by using *temporary stashes* instead of bringing tiles all the way back to the stash. However, it appears to be difficult to decide where and when to erect temporary stashes of which size.

Now, we consider how to use multiple robots to decrease the number of rounds needed by the algorithm. The general idea remains unchanged, but we will use a different algorithm for comparing outlines. Moreover, the additional robots shall follow the current barrier, taking up tiles when the barrier decreases in size and providing tiles when it increases. Hence, multiple tiles can be brought from and to the stash at once.

LEMMA 69: Consider a barrier U with disconnected next barriers $V_1, \dots, V_k$, as in Lemma 66. Let $\mathcal{E}_1, \dots, \mathcal{E}_k$ be the connected components obtained from $\mathcal{E}$ after removing U such that V_i is in $\mathcal{E}_i$, and $\ell_1 \geq \dots \geq \ell_k$, $\ell_i = ol(\mathcal{E}_i)$. $m \geq \log \ell_2$ robots can determine $\mathcal{E}_1$ in $O(\log^2 \ell_2 \cdot (|U| + \hat{\ell}))$ rounds, where $\hat{\ell} := \sum_{i=2}^k \ell_i$.

Proof. The idea is to use the algorithm from Lemma 66, but move the tiles by 2^t steps along the outline in iteration t. This is easy to do for $t = 0$ using only the leader. For each iteration $t > 0$, the leader maintains a connected *tail* of t robots, which follow when traversing outlines or U. This works since $\mathcal{E}$ is 2-connected, which implies that the leader never needs to move onto a node occupied by the tail when traversing an outline. The remaining robots are assumed to be parked along the outline of $\mathcal{E}$, behind U. After every iteration, one robot is added to the tail. The tail can now implement a shared counter, each robot storing one bit, starting with the least significant bit in the leader up to the most significant bit in the tail's last robot. The counter allows us to move tiles by 2^t steps.

Moving one step together with the tail and incrementing the counter once can be done in $O(t) = O(\log \ell)$ rounds. After iteration t, every tile has either reached its barrier or was moved $2^{t+1} - 1$ steps in total. Hence, $O(\log \ell_2)$ iterations are sufficient. Each iteration takes $O(\log \ell_2 \cdot (|U| + \hat{\ell}))$ rounds for a total of $O(\log^2 \ell_2 \cdot (|U| + \hat{\ell}))$.

If there is an iteration after which only one tile has not reached its barrier yet, we have identified $\mathcal{E}_1$. Otherwise, $\ell_1 \leq 2\ell_2$. In that case, we find $\mathcal{E}_1$ by measuring each outline using the counter. A second counter maintains the maximum length measured so far. Comparing and updating counters takes $O(\log \ell_2)$ rounds. After determining ℓ_1, the robots traverse the outlines once again to find an outline of that length, belonging to $\mathcal{E}_1$. This takes $O(\log \ell_2 \cdot (|U| + \hat{\ell}))$ rounds. $\square$

Note that the above technique can in principle be performed by a single robot and a tail of tiles representing a binary counter. However, distinguishing tiles part of the counter from other markers and barriers introduces some technical challenges.

THEOREM 70: If $\log n \leq m \leq td(\mathcal{E})$, m robots can decontaminate a simple n-node environment $\mathcal{E}$ in $O(n^2/m)$ rounds using $O(\log n \cdot td(\mathcal{E}))$ tiles.

Proof. We use the approach from Theorem 67. First, we describe how to use multiple robots to speed up constructing barriers and then analyze the runtime for comparing outlines with Lemma 69.

Initially, $\lfloor m/2 \rfloor$ of the robots shall pick up tiles from the stash. Trivially, it can be done in $O(m^2) = O(n)$ rounds. The robots form a *tail*, following their leader along the outline. Therefore, $O(m)$ rounds are needed to advance all robots by one step along the outline. The leader constructs barriers as previously. Now it can get tiles from the tail, in $O(td(\mathcal{E}))$ rounds each. When the tail does not have tiles anymore, it moves to the stash to pick up $\lfloor m/2 \rfloor$ tiles again, in $O(n)$ rounds. Barriers are removed in the same way.

Hence, the construction or removal of a barrier requires $O(td(\mathcal{E}))$ rounds plus the number of rounds used for carrying tiles from and to the stash. $O(n)$ tiles are carried in total, but always at least $m/2$ together. Thus, $O(n^2/m)$ rounds are used for carrying tiles and $O(n \cdot td(\mathcal{E})) = O(n^2/m)$ for constructing barriers. The runtime for moving to the next barrier after recursion can be amortized via the costs of constructing the barriers in between.

The runtime for comparing outlines is analyzed as in Theorem 67, using the same terminology. Each barrier U is traversed $O(\log^2 n)$ times by the algorithm from Lemma 69, using $O(n \log^3 n)$ rounds in total.

We have $T(\ell) \le \max_{\text{valid } k,l_1,\dots,l_k} \sum_{i=1}^{k} T(\ell_i) + \alpha\ell \log^2 \ell$ for some constant $\alpha > 1$, where $k, \ell_1, \dots \ell_k$ is valid if $\ell_i \le \ell/2$ for all i and $\sum_{i=1}^{k} \ell_i \le \ell$. By induction, we prove $T(\ell) \le 2\alpha\ell \log^3 \ell$. This clearly holds for $\ell = 1$. For $\ell > 1$, we obtain

$$\begin{aligned}
T(\ell) &\le \max_{\text{valid } k,\ell_1,\dots,\ell_k} \sum_{i=1}^{k} T(\ell_i) + \alpha\ell \log^2 \ell \\
&\le \max_{\text{valid } k,\ell_1,\dots,\ell_k} \sum_{i=1}^{k} 2\alpha\ell_i \log^3 \ell_i + \alpha\ell \log^2 \ell \\
&\le \alpha(2\ell \log^3(\ell/2) + \ell \log^2 \ell) \\
&= \alpha\ell(2(\log \ell - 1)^3 + \log^2 \ell) \\
&= \alpha\ell(2(\log^3 \ell - 3\log^2 \ell + 3\log \ell - 1) + \log^2 \ell) \\
&= \alpha\ell(2\log^3 \ell - 5\log^2 \ell + 6\log \ell - 2) \\
&\le 2\alpha\ell \log^3 \ell\,.
\end{aligned}$$

Hence, all outline comparisons are done in $O(n \log^3 n) = O(n^2/m)$ rounds. □

COROLLARY 71: If $\log n \le m \le \sqrt{n}$, m robots can decontaminate a simple n-node environment $\mathcal{E}$ in $O(n^2/m)$ rounds, using $O(\log n \cdot td(\mathcal{E}) + m)$ tiles.

Proof. Analogous to Theorem 70, with the difference that we might need more tiles to ensure that at least half of the robots can carry a tile. Also, the construction of each barrier takes $O(m + td(\mathcal{E}))$ rounds. Hence, $m \le \sqrt{n}$ is necessary to achieve the speedup. □

3.3 COMPLEX ENVIRONMENTS

To decontaminate complex environments $\mathcal{E}$, we use a simple algorithm that moves a straight line barrier across the environment, similar to the algorithm for parallelograms. This line might be split into separate connected components (see Figure 25).

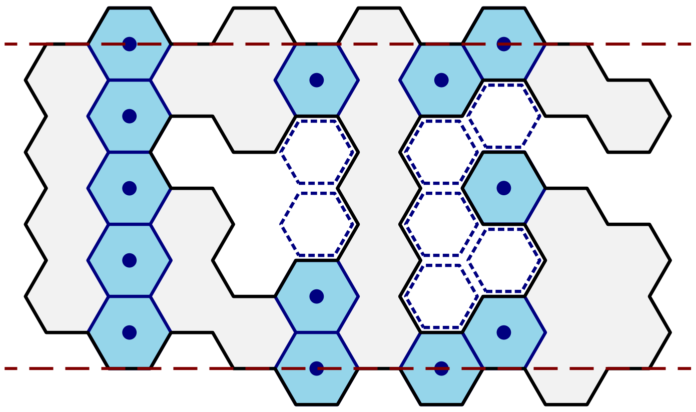


Figure 25: Barriers in a complex environment.

The assumptions from the previous section shall still hold, namely the absence of bottlenecks and the structure of the stash. Moreover, we assume no holes to be in contact with the stash.

The number of tiles required is determined by the maximum number of nodes in the same column of $\mathcal{E}$, i. e., nodes having the same x-coordinate, assuming a line in N direction (5 in the example). The difficult aspect of this algorithm is that the barrier has components not connected to the previous barrier (see the right barrier in the example).

Let us first describe the algorithm in terms of a single robot with a memory of $\Theta(\log n)$ bits. We further allow the robot to mark tiles, i. e., store $O(1)$ bits of information in tiles. These simplifications shall be removed afterwards. The key component is building a barrier at some x-coordinate, which is accomplished using Algorithm 2.

Algorithm 2 Algorithm to place tiles on all nodes with x-coordinate x_0.

```
1: procedure BuildLines($x_0$)            ▷ Robot is initially on the outer outline.
2:     for empty node $u$ on current outline at $x_0$ do
3:         Build maximal line from $u$ in N/S direction using GetTile.
4:         if $u$ at N of line and line has length $\geq 2$ then
5:                         ▷ Lines of length 1 do not connect different outlines due to the
           absence of bottlenecks.
6:             Mark S of line as *potential edge*.
7:     for node $u$ on current outline at $x_0$ do
8:         if tile on $u$ is marked as *potential edge* then
9:             Remove *potential edge* mark from current tile.
10:    for node $u$ on current outline at $x_0$ do
11:        if tile $v$ on the other side of $u$ is marked as *potential edge* then
12:            Mark $u$ and $v$ as *edge* and remove *potential edge* mark.
13:            Move to S of line.
14:            BuildBarrier($x_0$)
15:            Move to N of line.
16:    if exists node $u$ on current outline S of a line and marked *edge* then
17:        Move to N of line.
18:    else
19:        Done.
20: procedure GetTile                          ▷ Robot currently on a line.
21:    Move to N of current line.
22:    Mark current tile as *start*.
23:    repeat
24:        Traverse current outline until finding the stash or an incoming *edge* line.
25:        if *edge* line found then
26:            Move N to start of line.
27:            Mark current tile as *path*.
28:    until stash found
29:    repeat
30:        Traverse current outline until finding a *path* or *start* tile.
31:        Remove *path* or *start* marker.
32:        if *path* tile found then
33:            Move S to end of line.
34:    until *start* found
```

LEMMA 72: If $\mathcal{E}$ contains h nodes at x-coordinate x_0, then a single robot with a memory of $\Theta(\log n)$ bits, containing x_0, and the ability to mark tiles, can place tiles on all these nodes in $O(h \cdot n)$ rounds.

Proof. Algorithm 2 is executed, which is illustrated abstractly in Figure 26.

Possibly multiple lines need to be build, connecting different outlines. The idea is to build these lines in a DFS order (depth-first search), starting with lines from the outer outline. Certain lines shall be marked to form a DFS tree of all relevant

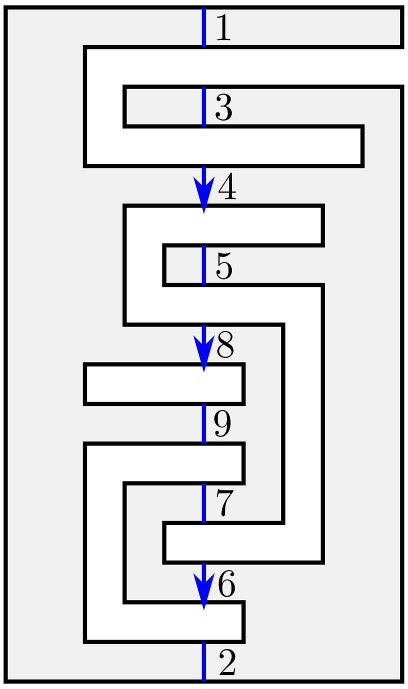


Figure 26: Building a barrier at a specific x-coordinate in a complex environment.

outlines. All tree edges shall be directed from N to S. This is essentially done with the BuildLines procedure, whose correctness we argue next.

The *potential edge* marks are used to detect which lines connect different outlines. They are removed in line 9 when a line is encountered twice from the same outline. Afterwards, only *potential edge* marks on lines to a different outline remain. Note that multiple potential edges to the same outline may still exist. They will be removed in the recursion from line 14, after one of the potential edges has been converted to an edge. After visiting an outline, no new potential edges towards it can be declared because all lines at that outline have already been built. Hence, the edges form a tree.

Now, assume that the robot does not visit an outline containing an x_0 node and among those, consider the outline with the northernmost x_0 node v. Therefore, the outline north of v has been visited and a line was built from a node u to v. Then v was marked *potential edge*. The *potential edge* mark on v could not have been removed since u and v are on different outlines and v's outline was never visited. But then u and v would have been marked *edge* in line 12. Afterwards, the outline of v would be traversed from line 14. Hence, a DFS tree is constructed as intended and all lines are built.

The GetTile procedure simply uses the edges to the current outline backwards to reach the outer outline where the stash is located. Used edges are marked so that the robot can return to where the tile is needed. Note that the robot shall always carry a tile such that it can at least place the first tile when building a new line which is then used to mark where to return.

Regarding runtime, each outline is entered at most once and traversed four times in BuildLines, which takes $O(n)$ rounds in total. Each call to GetTile takes $O(n)$ rounds as well. The algorithm terminates after $O(h \cdot n)$ rounds, since it places h tiles.

For additional clarity, we describe the algorithm based on the example from Figure 26. Outlines are traversed using the *right-hand rule*, i. e., as if a person was walking around the outline while touching it with their right hand. The robot starts on the outer outline and builds the lines 1, 2, 3, and 4. 1, 3, and 4 are marked as *potential edges*. After another traversal, these marks are removed from 1 and 3. 4 is marked *edge* and a recursion is performed. Lines 5, 6, 7, and 8 are created. Potential edges are initially 5, 6, and 8 and after another traversal only 6 and 8. 6 is marked edge and a recursion occurs. Line 9 is created and the robot returns using 6. It continues to 8 and marks 8 as an edge. Then the robot follows 8 and, after one traversal, returns along 8, then 4, and finally terminates. □

LEMMA 73: Let h be the maximum number of nodes within the same column of $\mathcal{E}$ and ℓ the number of columns. A single robot with a memory of $\Theta(\log n)$ bits and the ability to mark tiles can decontaminate an environment $\mathcal{E}$ in $O(n^2)$ rounds using $O(h)$ tiles.

Proof. First, the robot walks along the outer outline to determine the minimum x-coordinate, using the initial position as reference. It declares the minimum to be at $x = 0$ and builds barriers there using Algorithm 2. Then, for $k = 1, \dots, \ell - 1$, a barrier is built at $x = k$ and the barrier from $x = k - 1$ is subsequently removed. Removing barriers is similar to building barriers; for each outline, all lines adjacent to descendants are removed recursively. The incoming edge is removed last from an outline.

Correctness is obvious. Placing and removing a tile from each node uses $O(n^2)$ rounds. BUILDLINES is run ℓ times, using $O(\ell \cdot n)$ rounds in total to traverse outlines.

These bounds are tight for the given algorithm. It is easy to construct an environment where the distance between nearly subsequent barriers is $\Omega(n)$. □

The next step is to adapt the previous algorithms to work with our usual model, i. e., without counters and marking tiles. Therefor we again use a tail of robots or tiles as counter and tiles as markers. However, whereas only one tile on each outline as well as one tile next to each barrier was sufficient in the previous section, this is clearly not the case here. We impose further restrictions on $\mathcal{E}$ to make enough room for the different markers and a tail. Specifically, we assume that $\mathcal{E}$ was constructed from an environment $\mathcal{E}'$ without bottlenecks by replacing each node in $\mathcal{E}'$ with a hexagon of a sufficiently large radius $r = O(1)$. The algorithm on $\mathcal{E}'$ is simulated by building partially hollow hexagons of radius r instead of placing tiles. We construct these hexagons in a way such that they can store $O(1)$ bits of information as well as contain a tail of robots or tiles.

THEOREM 74: Let h be the maximum number of nodes within the same column of an environment $\mathcal{E}$ constructed from $\mathcal{E}'$ as described above and ℓ the number of columns. A single robot can decontaminate an environment $\mathcal{E}$ in $O(n^2 + \ell \cdot n \log n)$ rounds using $O(h)$ tiles.

Proof. The algorithm from Lemma 73 on $\mathcal{E}'$ is simulated on $\mathcal{E}$. The robot maintains a tail of $\Theta(\log n)$ tiles to serve as memory. It does not need to know that value. Instead, it extends the tail when the need arises. Hence, each round from Lemma 73 when traversing an outline requires $\Theta(\log n)$ rounds to simulate. Tiles can still be transported in $O(n)$ rounds each because GetTile does not use the counter. □

REMARK 75: Theorem 74 can also detect holes by checking whether Algorithm 2 marks an edge at some point.

THEOREM 76: *m* robots can decontaminate an environment $\mathcal{E}$ in $O(n^2/m + \ell \cdot n \cdot (\log n + m))$ rounds using $O(h + m)$ tiles.

Proof. The m robots form a tail, similar to the previous section. GetTile is now run by all robots together. Note that only the first robot needs to mark tiles. Hence, m tiles can be transported in $O(n)$ rounds. □

Note that a speedup is only achieved if $\ell \cdot (\log n + m) = o(n)$. In a practical context, one might assume the environment to fit inside a hexagon of radius $\Theta(\sqrt{n})$. In that case, a runtime of $O(n^{7/4})$ is achievable with $m = n^{1/4}$. More robots would make the algorithm slower.

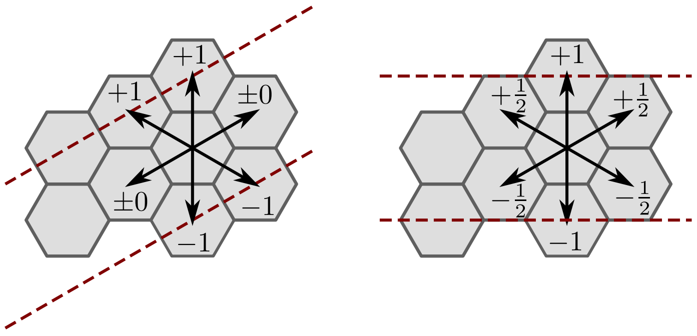


Figure 27: Measuring the width of an environment using lines in different directions. Arrows indicate how to count when traversing the outline.

Before executing the algorithm, h can be upper bounded by measuring the *width* $w(\mathcal{E})$, which is defined similarly to the width of a polygon. $w(\mathcal{E})$ shall be the maximum distance of two parallel lines containing nodes of $\mathcal{E}$, which have either direction N, NW, NE, or are orthogonal to such a direction. This is visualized in Figure 27 with lines in NW direction (left) and lines orthogonal to N (right). Width can now be measured by traversing the outline and maintaining a counter as indicated by the arrows. The difference between the observed minimum and maximum is the width.

Barriers can then be built in a direction to minimize width. Alternatively, we can also employ the trick from Section 3.1 where the algorithm is restarted if the number of tiles is insufficient, guaranteeing decontamination when it is known that the number of tiles is sufficient for barriers in one of the three directions.

3.4 COMPLEXITY

In this section, we will show that determining whether a given environment can be decontaminated with s tiles is $\mathcal{NP}$-hard. This is done using a relatively straightforward reduction from edge search on planar graphs with maximum node degree three.

Given an n-node planar graph $G = (V, E)$ with maximum node degree 3 and an integer s, deciding whether $es(G) \leq s$ is $\mathcal{NP}$-complete [32]. We shall construct an environment $\mathcal{E}$ with $td(\mathcal{E}) = es(G)$.

There exists a planar orthogonal drawing of G with area $O(n^2)$, i. e., a drawing without edge crossings in which all nodes have integer coordinates and all edges consist of line segments parallel to the x- or y-axis with integer endpoints. Such a drawing can be constructed in polynomial time [36]. With slight modifications, we obtain a drawing such that all inner angles are 120°. This is to ensure that the drawing can be embedded into the triangular lattice graph. From there, $\mathcal{E}$ can be constructed as depicted in Figure 28, ensuring that edges have at least length three. $\mathcal{E}$ has $O(n^2)$ nodes because the orthogonal drawing fits inside an area of size $O(n^2)$. Let $v \in V$ and $e \in E$. We shall refer to the set of nodes representing v and e in $\mathcal{E}$ as S_v and S_e, respectively.

The top left shows a planar graph, the middle an orthogonal drawing and the right a modified version of that drawing with only 120° angles. The environment is depicted at the bottom with nodes in dark gray and edges in light gray.

Inside the environment, the decontamination of the left edge is shown. The contaminated area is tinted red. Initially, the left tile was placed at the arrow's origin. Hence, two tiles representing searchers in G guard the area of $\mathcal{E}$ corresponding to an edge of G. An edge-search strategy would decontaminate that edge by sliding one of the searchers across it. In $\mathcal{E}$, one tile is moved step by step to the other, effectively mimicking sliding a searcher.

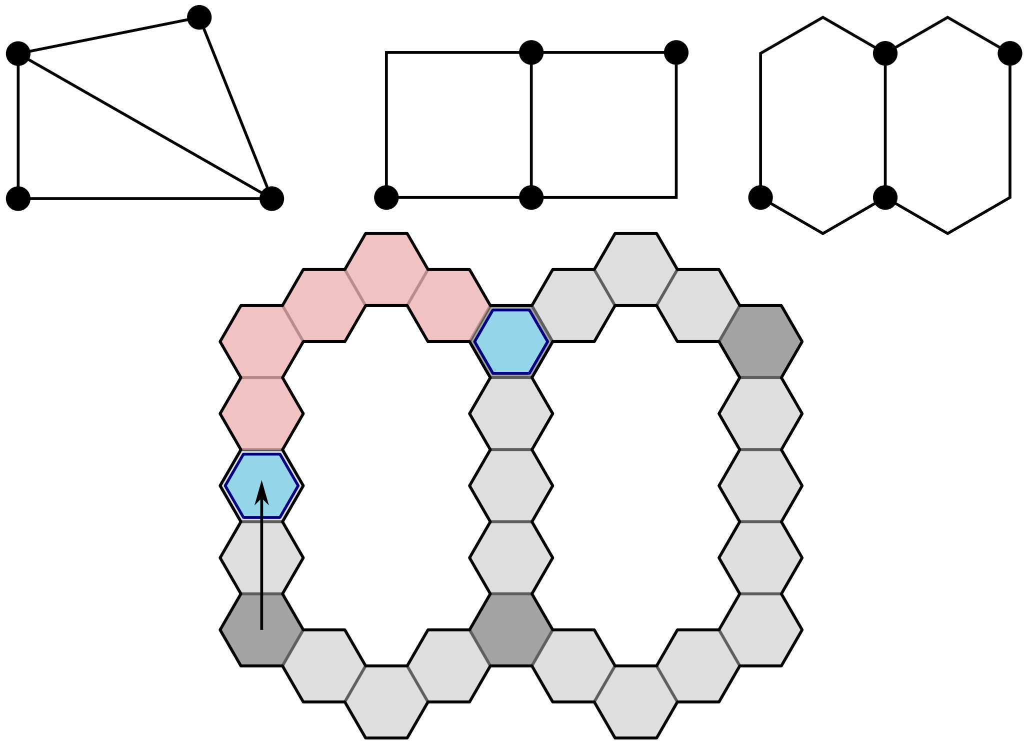


Figure 28: Constructing an environment from a planar graph via an orthogonal drawing.

THEOREM 77: Given an n-node planar graph $G = (V, E)$ with maximum node degree 3, the environment $\mathcal{E}$ as constructed above has $td(\mathcal{E}) = es(G)$.

Proof. First, we transform an edge-search strategy $\mathcal{S}$, $es(G, \mathcal{S}) = s$ into a sequence of tile movements in $\mathcal{E}$. Initially, all s tiles are located somewhere in $\mathcal{E}$ and shall be regarded as *unused* until they represent some searcher on G. We now need to transfer each move of $\mathcal{S}$ to a tile movement on $\mathcal{E}$. W.l.o.g., we may assume that at most two searchers are at the same node, because the only case where more than one searcher on the same node is necessary is immediately before or after sliding a searcher. Hence, more than two searchers on a node are unnecessary and can be removed.

If a searcher is added to a node, an unused tile is placed there or at an adjacent node inside $\mathcal{E}$, i. e., part of an edge in G. If a searcher is slid along an edge e and actually clears e, the corresponding tile is moved along S_e as depicted in Figure 28. If a searcher is removed, nothing is done; the tile is simply regarded as unused.

Clearly, the tile movement corresponding to a slide along e decontaminates all nodes in S_e and all paths in $\mathcal{E}$ connecting decontaminated and contaminated edges contain nodes with tiles. Hence, the constructed sequence of tile movements decontaminates $\mathcal{E}$.

Next, we transform a sequence of tile movements on $\mathcal{E}$ with s tiles into an edge-search strategy decontaminating G. Consider an edge $e \in E$ incident to $e' \in E$ via $v \in V$ such that S_e is fully decontaminated while $S_{e'}$ is not. There has to be a tile in either $S_{e'}$ or S_v, separating S_e from the contamination in $S_{e'}$. That tile is only counted from node v. Thus, there are at most s such tiles. A searcher shall guard v until the condition from above does not hold anymore (1).

Since S_e might initially be full of tiles and therefore fully decontaminated, that initial state needs to be replicated in G. There are at most $s/3$ such edges. We place two searchers on an incident node and slide one of them along the edge. Thereafter, all edges e with decontaminated S_e are decontaminated as well. We remove searchers according to (1), having replicated the initial state of $\mathcal{E}$ on G.

We now describe how to handle changes in the set of fully decontaminated edges. If an edge is recontaminated, an S_v might now only be incident to contaminated S_e. In that case, the searcher is removed from v, such that (1) still holds.

Consider the last move of a tile decontaminating S_e, $e = \{v, w\}$. If the last node of S_e to be decontaminated was not adjacent to S_v or S_w, we add searchers to v such that it contains two and slide one across to w. This action cannot cause us to use more than s searchers since there are at least two tiles in S_e.

Otherwise, the last node of S_e to be decontaminated is adjacent to v, w.l.o.g. There are now two cases.

(i) w is adjacent to another contaminated edge e'. Hence, there has to be a tile on $S_{e'}$ or S_w. We add a searcher to w and slide it to v. At most s searchers are used since one tile separates e from e' and another finalizes the decontamination of e.

(ii) e is the sole edge adjacent to w such that S_e is not fully decontaminated. In that case, we add a searcher to w if there is none (this only occurs if $deg(w) = 1$) and slide the searcher from w to v. Clearly, the number of searchers does not exceed s for (ii) as well. □

COROLLARY 78: Deciding whether $td(\mathcal{E}) \leq s$, given s, is $\mathcal{NP}$-hard.

We conjecture this problem to be $\mathcal{NP}$-complete. The result that recontamination does help for edge search [27] is not directly applicable here, since the tile structure needs to remain connected. Hence, the question whether the problem is in $\mathcal{NP}$ is non-trivial.

4 CONCLUSION

In this chapter, we give an evaluation of our work and propose future research questions.

4.1 EVALUATION

In this thesis, we have introduced a novel model for performing decontamination of nanoscale environments using robots and tiles, based on an existing model from literature. To the best of our knowledge, this is the first time decontamination in grids with finite-automaton robots and an instantly spreading contaminant has been studied. We were able to develop asymptotically optimal algorithms for convex environments, both in terms of runtime and the number of tiles. In more general environments, we can only guarantee runtime $O(n^2)$ with a single robot and up to $O(n^{3/2})$ using multiple robots. We believe that faster algorithms should be possible, considering that $O(\sqrt{n})$ constitutes an upper bound on the number of tiles needed to decontaminate a simple environment by Lemma 64 and $\Omega(n^{3/2})$ is a lower bound on the time a single robot needs to decontaminate any environment. For non-convex environments, we only use multiple robots to speed up the transportation of tiles from and to the stash, while essentially executing the single robot algorithm with the leader. Furthermore, we restricted ourselves to bottleneck-free environments and environments made up from smaller hexagons in the case of complex environments. We conjecture that these restrictions can be lifted without having to use fundamentally different algorithms, but that would introduce many technical difficulties.

For the geometric problem of decontaminating polygons using barrier curves, we developed several approximation algorithms. As far as we know, these are the first algorithms in literature to decontaminate all simple and complex polygons with bounds on the bottleneck length in terms of the optimum. We believe that the techniques from Section 2.5.1 to compress a polygon in such a way that it can be rasterized in a grid of polynomial size are rather interesting and might have applications in developing polynomial time approximation algorithms for other problems on polygons. Although polynomial, the fact that the resulting grids have $O(n^{14})$ nodes makes this approach not very practical. However, we conjecture this bound to not be tight. Improving the bound on the number of vertices in the polygon after compression to $O(n^3)$, as conjectured, would improve the bound on the number of nodes by a factor of $\Theta(n^3)$.

When we proposed this research, we expected to be able to apply algorithms from the geometric model to the robot model. At least on a superficial level, the techniques used for related classes of environments in both models are similar and have similar approximation factors (cf. Table 1 and Table 2). Convex environments are decontaminated by sweeping them with a line. Simple environments are decomposed into a tree in a way such that a node can be blocked using a number of nodes or barriers of length not much higher than what is a lower bound for full decontamination, hence achieving an approximation factor of $O(\log n)$. The $O(\log n)$ approximation algorithm for complex polygons as well as the algorithm for STPs do not seem to be applicable to the robot model due to memory requirements. The simpler algorithm for complex polygons has a similar approximation factor to the algorithm for complex environments, if the environment is sufficiently dense. However, the approach of these algorithms is rather different, in that we use a simple sweepline technique for robots, while we partition the polygon into a graph and compute a node-search strategy.

4.2 FUTURE WORK

Besides what we mentioned above, there are many potential avenues to extend the results from this thesis. Of course, the natural next step would be to improve the approximation factors and runtime of our algorithms. We are most interested in proving the conjecture from Remark 28. Generally, approximation algorithms for the sweepwidth of polygons translate to approximation algorithms for the pathwidth of partial grids. Therefore, we believe that improving our approximation factors for complex environments would be very difficult because the best known approximation algorithm for pathwidth of planar graphs has the approximation factor $O(\log n)$. The problem does not get easier when restricting it to partial grids (cf. Section 3.4). However, partial grids without holes, corresponding to simple polygons, might be easier.

One can also consider variants of the geometric problem where the boundary of environments may be curved. Also, the number of barriers could be restricted. For a single barrier, this has been studied as ring-width [18]. Our algorithm for simple polygons uses $O(\log n)$ barriers. Another interesting problem is whether recontamination is needed to achieve optimal sweepwidth (cf. Corollary 47). A further curious problem is computing a square-tree polygon from a given weighted tree (see Remark 20).

We have only considered approximation up to a constant factor. Therefore, PTAS (polynomial-time approximation schemes) could be considered, i. e., achieving approximation factor ϵ in polynomial time for any fixed $\epsilon > 0$. Additive approximation of pathwidth is known to be $\mathcal{NP}$-hard [5]. These results do not easily translate to our models since the proof makes use of non-planar graphs. Further open hardness questions for sweepwidth and the minimum number of tiles might be interesting to investigate, including simple environments and polygons with polynomially bounded integer coordinates.

Regarding the robot model, we conjecture that a triangle with side length h cannot be decontaminated with fewer than h tiles (cf. Lemma 59). In our algorithm for simple environments, the time to transport tiles from and to the stash is the bottleneck in terms of the runtime. We believe that it could be improved by not always bringing tiles back to the stash but instead storing them in some kind of temporary stash closer to the barriers.

Fault tolerance might also be interesting to consider in the robot model when dealing with multiple robots. Our current algorithms rely heavily on a leader and therefore are not directly applicable under the presence of faults. Finally, both models could be extended to three dimensions.

BIBLIOGRAPHY

[1] Y. Altshuler, A. Pentland, and A. M. Bruckstein. *Swarms and Network Intelligence in Search*. Springer International Publishing, 2018.

[2] B. Bhattacharya, T. Kameda, and J. 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*. 2009, pp. 2461–2466.

[3] H. Blum. "A transformation for extracting new descriptors of shape." In: *Models for the Perception of Speech and Visual Form*. Ed. by W. Wathen-Dunn. MIT Press, 1967, pp. 362–381.

[4] H. L. Bodlaender. "A partial k-arboretum of graphs with bounded treewidth." In: *Theoretical Computer Science* 209.1 (1998), pp. 1 –45. ISSN: 0304-3975.

[5] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. "Approximating treewidth, pathwidth, and minimum elimination tree height." In: *Graph-Theoretic Concepts in Computer Science*. Ed. by G. Schmidt and R. Berghammer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 1–12. ISBN: 978-3-540-46735-9.

[6] F. Y. Chin, J. Snoeyink, and C. A. Wang. "Finding the Medial Axis of a Simple Polygon in Linear Time." In: *Discrete & Computational Geometry* 21.3 (1999), pp. 405–420. ISSN: 1432-0444.

[7] H. I. Choi, S. W. Choi, and H. P. Moon. "Mathematical theory of medial axis transform." In: *Pacific Journal of Mathematics* 181.1 (1997), pp. 57–88.

[8] Y. Daadaa, P. Flocchini, and N. Zaguia. "Network Decontamination with Temporal Immunity by Cellular Automata." In: *Cellular Automata*. Ed. by S. Bandini, S. Manzoni, H. Umeo, and G. Vizzari. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 287–299. ISBN: 978-3-642-15979-4.

[9] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. "Universal Shape Formation for Programmable Matter." In: *28th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2016, pp. 289–299.

[10] Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. Bazzi, A. W. Richa, and C. Scheideler. "Leader Election and Shape Formation with Self-organizing Programmable Matter." In: *DNA Computing and Molecular Programming*. Ed. by A. Phillips and P. Yin. Cham: Springer International Publishing, 2015, pp. 117–132. ISBN: 978-3-319-21999-8.

[11] D. Dereniowski. "Approximate search strategies for weighted trees." In: *Theoretical Computer Science* 463 (2012). Special Issue on Theory and Applications of Graph Searching Problems, pp. 96 –113. ISSN: 0304-3975.

[12] D. Dereniowski and D. Urbańska. "On-line Search in Two-Dimensional Environment." In: *Approximation and Online Algorithms*. Ed. by R. Solis-Oba and R. Fleischer. Cham: Springer International Publishing, 2018, pp. 223–237. ISBN: 978-3-319-89441-6.

[13] F. V. Fomin and D. M. Thilikos. "An annotated bibliography on guaranteed graph searching." In: *Theoretical Computer Science* 399.3 (2008). Graph Searching, pp. 236 –245. ISSN: 0304-3975.

[14] S. Fortune. "A sweepline algorithm for Voronoi diagrams." In: *Algorithmica* 2.1 (1987), p. 153. ISSN: 1432-0541.

[15] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann. "Forming Tile Shapes with Simple Robots." In: *DNA Computing and Molecular Programming 24th International Conference, DNA 24, Proceedings*. 2018 (forthcoming).

[16] R. Gmyr, I. Kostitsyna, F. Kuhn, C. Scheideler, and T. Strothmann. "Forming tile shapes with a single robot." In: *Abstr. 33rd European Workshop on Computational Geometry (EuroCG)*. 2017, pp. 9–12.

[17] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, and C. Scheideler. "Shape Recognition by a Finite Automaton Robot." In: *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Ed. by I. Potapov, P. Spirakis, and J. Worrell. Vol. 117. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 52:1–52:15. ISBN: 978-3-95977-086-6.

[18] J. E. Goodman, J. Pach, and C. K. Yap. "Mountain Climbing, Ladder Moving, and the Ring-Width of a Polygon." In: *The American Mathematical Monthly* 96.6 (1989), pp. 494–510.

[19] Q.-P. Gu and G. Xu. "Near-Linear Time Constant-Factor Approximation Algorithm for Branch-Decomposition of Planar Graphs." In: *Graph-Theoretic Concepts in Computer Science*. Ed. by D. Kratsch and I. Todinca. Cham: Springer International Publishing, 2014, pp. 238–249. ISBN: 978-3-319-12340-0.

[20] M. E. Houle and G. T. Toussaint. "Computing the width of a set." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.5 (1988), pp. 761–765. ISSN: 0162-8828.

[21] F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. "Distributed reconfiguraiton of 2D lattice-based modular robotic systems." In: *Autonomous Robots* 38.4 (2015), pp. 383–413.

[22] B. Karaivanov, M. Markov, J. Snoeyink, and T. S. Vassilev. "Decontaminating Planar Regions by Sweeping with Barrier Curves." In: *CCCG*. 2014.

[23] L. M. Kirousis and C. H. Papadimitriou. "Interval graphs and searching." In: *Discrete Mathematics* 55.2 (1985), pp. 181–184. ISSN: 0012-365X.

[24] L. M. Kirousis and C. H. Papadimitriou. "Searching and pebbling." In: *Theoretical Computer Science* 47 (1986), pp. 205–218.

[25] K. Klein and S. 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. Toronto, Ontario, Canada: AAAI Press, 2012, pp. 2010–2016.

[26] E. Korach and N. Solel. "Tree-width, path-width, and cutwidth." In: *Discrete Applied Mathematics* 43.1 (1993), pp. 97 –101. ISSN: 0166-218X.

[27] A. S. LaPaugh. "Recontamination Does Not Help to Search a Graph." In: *J. ACM* 40 (1993), pp. 224–245.

[28] M. Lassak. "Approximation of convex bodies by rectangles." In: *Geometriae Dedicata* 47.1 (1993), pp. 111–117. ISSN: 1572-9168.

[29] D. T. Lee. "Medial Axis Transformation of a Planar Shape." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4.4 (Apr. 1982), pp. 363–369. ISSN: 0162-8828.

[30] M. Markov, V. Haralampiev, and G. Georgiev. "Lower bounds on the directed sweepwidth of planar shapes." In: (2015).

[31] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. "The Complexity of Searching a Graph." In: *J. ACM* 35.1 (Jan. 1988), pp. 18–44. ISSN: 0004-5411.

[32] B. Monien and I. Sudborough. "Min cut is NP-complete for edge weighted trees." In: *Theoretical Computer Science* 58.1 (1988), pp. 209 –229. ISSN: 0304-3975.

[33] T. D. Parsons. "Pursuit-evasion in a graph." In: *Theory and applications of graphs*. Springer, 1978, pp. 426–441.

[34] M. J. Patitz. "An introduction to tile-based self-assembly and a survey of recent results." In: *Natural Computing* 13.2 (2014), pp. 195–224.

[35] N. Sarnak and R. E. Tarjan. "Planar point location using persistent search trees." In: *Communications of the ACM* 29.7 (1986), pp. 669–679.

[36] R. Tamassia. "Planar orthogonal drawings of graphs." In: *IEEE International Symposium on Circuits and Systems*. 1990, 319–322 vol.1.

[37] J. Urrutia. "Chapter 22 - Art Gallery and Illumination Problems." In: *Handbook of Computational Geometry*. Ed. by J.-R. Sack and J. Urrutia. Amsterdam: North-Holland, 2000, pp. 973 –1027. ISBN: 978-0-444-82537-7.

[38] F.-E. Wolter. "Cut locus and medial axis in global shape interrogation and representation." In: *MIT Design Laboratory Memorandum 92-2 and MIT Sea Grant Report*. 1992.

[39] D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. "Active Self-assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time." In: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ITCS '13. Berkeley, California, USA: ACM, 2013, pp. 353–354. ISBN: 978-1-4503-1859-4.

[40] C.-K. Yap. "How to move a chair through a door." In: *IEEE Journal on Robotics and Automation* 3.3 (1987), pp. 172–181. ISSN: 0882-4967.