Generalized Voronoi Diagrams
Theory and Related Applications

Stefan de Lorenzo

Cumulative dissertation submitted to the Faculty of Natural Sciences of the University of Salzburg in partial fulfillment of the requirements for the doctoral degree Dr. techn.

Supervisor: Martin Held
Department of Computer Science
University of Salzburg

July 2021

Department of Computer Science
University of Salzburg
Jakob-Haringer-Straße 2
5020 Salzburg
Austria

Stefan de Lorenzo
Matriculation number: 00920908

Generalized Voronoi Diagrams: Theory and Related Applications

July 2021

Abstract

The standard *Voronoi diagram* of points in the plane is one of the most prominent tools in computational geometry and has been intensively studied in the past. It can be generalized in a wide variety of ways. A natural extension is to associate multiplicative weights with the individual input points. In this scenario, the extent of a Voronoi region does not only depend on the location of the respective site, but is also influenced by the corresponding weight. In this cumulative dissertation, an efficient practice-minded strategy for computing the *multiplicatively weighted Voronoi diagram (MWVD)* of points is presented. Additionally, we will take a look at the recognition and reconstruction of *weighted bisector graphs*.

Another way to generalize the Voronoi diagram is to extend the range of permissible input sites. We will utilize the Voronoi diagram of straight-line segments and circular arcs, to generate tool paths inside planar shapes for NC machining. Of course, these generalizations can also be combined. The *generalized weighted Voronoi diagram (GWVD)* even allows the endpoints of one of its input straight-line segment to be associated with different multiplicative weights. We present an in-depth analysis of the GWVD and introduce an alternative structure that we will refer to as the *variable-radius skeleton (VRS)* inside a polygon. Both the GWVD as well as the VRS can be used to generate so-called *variable-radius offsets*.

Additionally, we take a close look at *convex decompositions* of points sets. In particular, several heuristics are presented that allow us to reduce the face count of an initial convex decomposition.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Professor Martin Held for his invaluable guidance, motivation, and patience. Furthermore, I want to thank my colleagues Günther Eder and Peter Palfrader who always had time for insightful discussions.

I am extremely grateful to my parents, Angelika and Herbert, and the rest of my family for supporting me spiritually throughout my studies. My special thanks goes to my girlfriend Sabrina for her continuous support and understanding.

This work has been supported by the Austrian Science Fund (FWF) under the grant number P31013-N31.

Contents

I

Introduction

1
Preliminaries

Geometric structures and algorithms are omnipresent in our modern world. Amongst many things, they are used in computer graphics, industrial NC machining, urban planning, and the modeling of crystal growth [ZL02; Hel91; Oka+08; SD91].

In Chapter 1 we will take a look at some of the fundamental geometric structures that this cumulative dissertation is based on. In particular, Section 1.1 gives an overview of the *Voronoi diagram* with a particular emphasis on the *medial axis* as well as weighted Voronoi diagrams. A structure that is closely related to the Voronoi diagram is the *straight skeleton*; see Section 1.2. Additionally, Section 1.3 is dedicated to *convex decompositions* and their computation.

Chapter 2 summarizes the contributions of this cumulative dissertation. Note that the main parts of this dissertation have already been published or have been submitted to peer-reviewed journals and workshops. The corresponding papers can be found in Part II.

1.1 Voronoi Diagrams

1.1.1 The Voronoi Diagram of Points

The standard *Voronoi diagram* $\mathcal{VD}(S)$ of a set S of n points in the plane that we will refer to as *sites* is a versatile tool in computational geometry; see Figure 1. It consists of several so-called *Voronoi regions*. Every Voronoi region $\mathcal{VR}(s, S)$ is associated with a site $s \in S$ and contains all points in $\mathbb{R}^2$ that are closest to s under the Euclidean distance metric. More formally,

$$\mathcal{VR}(s, S) := \{p \in \mathbb{R}^2 : d(s, p) \leq d(s, S)\},$$

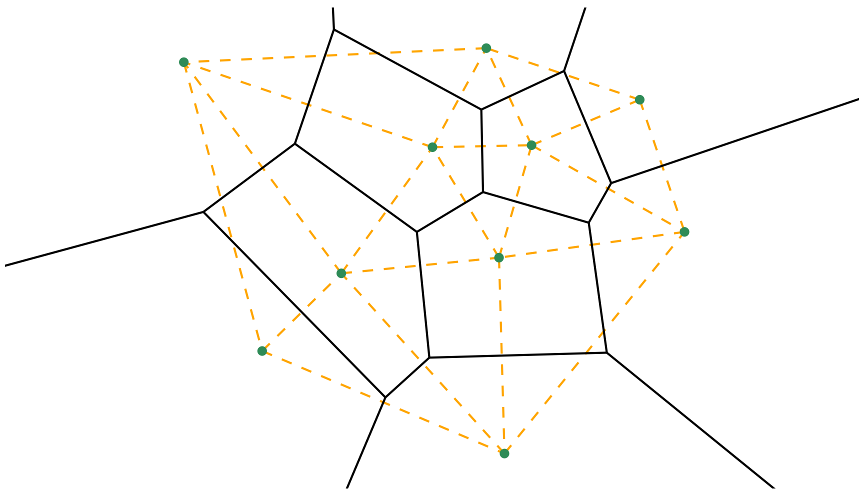


FIGURE 1: The Voronoi diagram (shown in black) of a set of input points (highlighted in green) as well as the corresponding Delaunay triangulation (shown in orange).

where $d(\cdot,\cdot)$ denotes the standard Euclidean distance. The boundary of a Voronoi region is commonly referred to as a *Voronoi cell*. The combinatorial complexity of $\mathcal{VD}(S)$, that is, the number of its vertices, edges, and faces, is bounded by $\mathcal{O}(n)$. Additionally, every Voronoi region $\mathcal{VR}(s,S)$ is convex, i.e., if the two points p and q are situated inside $\mathcal{VR}(s,S)$, then the straight-line segment $\overline{pq}$ also lies entirely within $\mathcal{VR}(s,S)$. Furthermore, a *bisector* between two sites $s_1, s_2 \in S$ is the set of all points in the plane that are equidistant to s_1 and s_2.

The earliest mention of a structure that resembles the Voronoi diagram dates back to the 17th century. In his book that is titled *Principles of Philosophy* [Des44], Descartes outlines his vortex theory of planetary motion that attempts to model the orbits of planets and other astronomical objects. These vortices form a subdivision of space in which each vortex is associated with a single star; see Figure 2.

Over two centuries passed until the Voronoi diagram was first explicitly formalized by the mathematicians Dirichlet [Dir50] and Voronoi [Vor08]. Later on, it was rediscovered in several scientific fields, e.g., by Thiessen [Thi11] to determine the mean areal precipitation of specific catchment areas. Thus, Voronoi diagrams are also sometimes referred to as *Dirichlet tessellations* or *Thiessen polygons*.

Several strategies are known that allow us to compute $\mathcal{VD}(S)$ in worst-case optimal $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space. The first strategy that achieved worst-case optimality is a divide-and-conquer algorithm which is presented by Shamos and Hoey [SH75]. Fortune's famous approach [For87] uses a so-called sweepline that moves upwards across the plane to construct $\mathcal{VD}(S)$. This algorithm was later generalized to a class of *nice metrics* by Dehne and Klein [DK97]. Amongst other things, nice metrics include all convex distance functions.

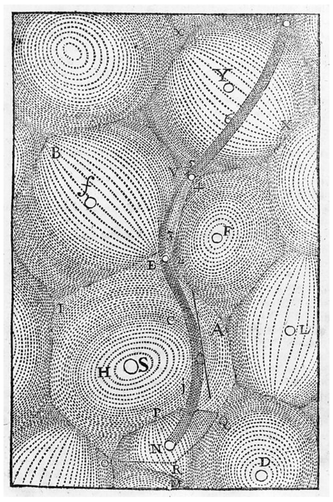


FIGURE 2: A subdivision of space into several vortices as it appears in Descartes' work [Des44].

Randomized geometric strategies are usually simpler compared to their deterministic counterparts, although their theoretical analysis tends to be more involved. In their pioneering work, Clarkson and Shor [CS89] introduce an abstract framework for establishing average-case bounds of randomized algorithms. Chew [Che90] outlines a simple strategy that allows to compute the Voronoi diagram of a set of points that are in convex position in linear expected time. Guibas et al. [GKS90] present an approach that uses incremental construction to generate $\mathcal{VD}(S)$ in expected $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.

Additionally, it is possible to derive the so-called *Delaunay triangulation* $\mathcal{DT}(S)$ of S in linear time from the corresponding Voronoi diagram $\mathcal{VD}(S)$. All sites that are neighbors in $\mathcal{VD}(S)$ are connected via a triangulation edge. Thus, $\mathcal{DT}(S)$ is called the straight-line dual of $\mathcal{VD}(S)$. An advantageous property of the Delaunay triangulation is that it maximizes the minimum angle inside a triangle over all possible triangulations.

A vast number of geometric problems can be solved efficiently if the corresponding Voronoi diagram has already been computed. Amongst other things, it is possible to find the closest pair of points in S or compute the Euclidean minimum spanning tree (EMST) of S, i.e., the planar straight-line graph (PSLG) on S that is connected and has minimum total edge length, in linear time if $\mathcal{VD}(S)$ is known. A more exhaustive list of problems is given by Aurenhammer and Klein [AK00].

1.1.2 The Medial Axis

The Voronoi diagram of points can be easily generalized to other types of inputs sites. We refer to Held and Huber [HH09] for a definition of the Voronoi diagram of straight-line segments and circular arcs. Blum [Blu67] was the first to introduce the medial axis transform inside a planar shape; see Figure 3. Since then many researchers have worked on finding efficient ways for computing the medial axis. Lee [Lee82] presents a divide-and-conquer algorithm that is able to generate the medial axis inside a simple polygon with n edges in $\mathcal{O}(n \log n)$ time and provides a corresponding practical implementation.

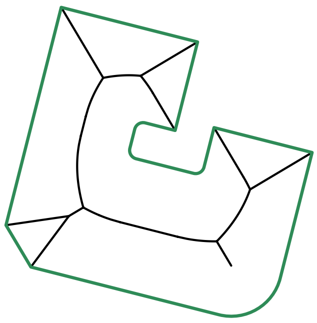


FIGURE 3: The medial axis (highlighted in black) inside a planar shape.

Inside a planar region that is bounded by a set S of straight-line segments and circular arcs, the medial axis $\mathcal{MA}(S)$ is a subset of the Voronoi diagram $\mathcal{VD}(S)$. Therefore, $\mathcal{MA}(S)$ can be derived from $\mathcal{VD}(S)$ in linear time. Yap [Yap87] presents an $\mathcal{O}(n \log n)$ time algorithm for computing the Voronoi diagram of n straight-line segments and circular arcs. Held and Huber [HH09] present an efficient algorithm that is able to compute Voronoi diagrams of points, straight-line segments, and circular arcs in expected $\mathcal{O}(n \log n)$ time. Its implementation is based on standard double-precision floating-point arithmetic.

1.1.3 Weighted Voronoi Diagrams

Another natural extension of this idea is to associate additive or multiplicative weights with the individual sites. These weighted Voronoi diagrams were first introduced by Boots [Boo80] in terms of market area analysis. The additively weighted Voronoi diagram (AWVD) has a linear combinatorial complexity overall and can be computed in worst-case optimal $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space, e.g., by using Fortune's sweepline approach [For87]. Aurenhammer [Aur87] presents the power diagram (PD) which can be seen as an AWVD in the power distance; see Figure 4. It subdivides the plane into several convex regions.

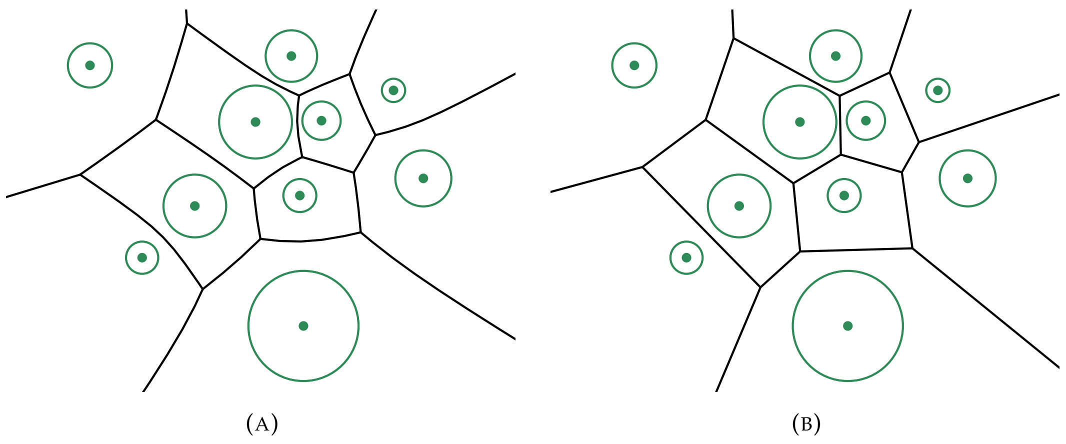


FIGURE 4: An AWVD is shown in (A). Additionally, (B) shows the corresponding PD for the same site locations and weights. Note that the magnitude of the individual weights is indicated by the radius of the disks that are centered at the respective site locations.

Things change drastically when dealing with the multiplicatively weighted Voronoi diagram (MWVD), as its Voronoi regions may be disconnected. A face that does not contain its generating site is usually referred to as an *orphan* [LS03]. Moreover, every such Voronoi region may consist of up to linearly many orphans. Thus, the combinatorial complexity of the MWVD of a set of n sites is bounded by $\mathcal{O}(n^2)$; see Figure 5.

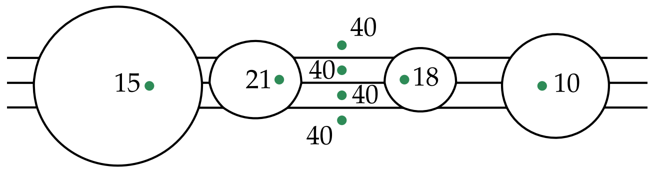


FIGURE 5: A worst-case example of a MWVD with 8 sites [AE84]. Note that the respective weights are written next to the site locations.

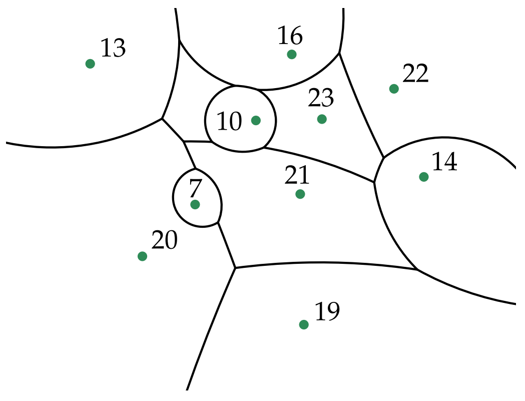
The bisector between two (multiplicatively weighted) sites s_1 and s_2, with $w(s_1) \neq w(s_2)$, is a circle. Recall that Apollonius of Perga defined a circle as a set of points that have a fixed distance ratio to two foci. Table 1 summarizes the underlying distance functions for different weighted Voronoi diagrams.

	AWVD	MWVD	PD
Distance function	$d(p,s) - w(s)$	$\dfrac{d(p,s)}{w(s)}$	$d(p,s)^2 - w(s)^2$

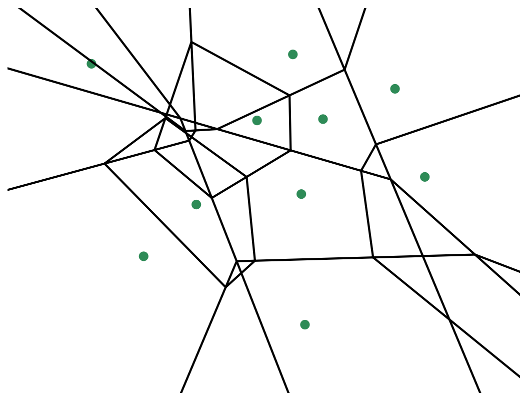
TABLE 1: The distance functions that are associated with AWVDs, MWVDs, and PDs, where p is a point in $\mathbb{R}^2$ and $s \in S$ is associated with the real-valued weight $w(s)$ in which $w(s) > 0$ for MWVDs and $w(s) \geq 0$ for AWVDs as well as PDs.

Aurenhammer and Edelsbrunner [AE84] present an algorithm for constructing the MWVD of a set S of n input points in $\mathbb{R}^2$ in optimal $\mathcal{O}(n^2)$ time and space. They transform the construction of the MWVD $\mathcal{VD}_w(S)$ in the plane into a three-dimensional problem in which all sites in S are situated in the xy-plane. Every bisector between two sites is associated with a sphere in $\mathbb{R}^3$. All of these spheres share a common point. The closed interior of every sphere is then mapped to a three-dimensional closed half-space using a geometric technique called *inversion*; see [Bro79]. Inversion is involutory, that is, applying inversion twice yields the original point. Every Voronoi region is mapped to a polyhedron in $\mathbb{R}^3$. Therefore, $\mathcal{VD}_w(S)$ is mapped to the intersection of the partition, that is induced by the polyhedra that are associated with the sites in S, with the sphere that corresponds to the xy-plane. Finally, $\mathcal{VD}_w(S)$ is generated by inverting this intersection.

Har-Peled and Raichel [HR15] show that the expected complexity of the MWVD equals $\mathcal{O}(n \log^2 n)$ if all weights are chosen randomly. They use a so-called *overlay arrangement* $\mathcal{OA}(S)$ of S that is generated by incrementally constructing the standard (unweighted) Voronoi diagram, in which the sites are inserted ordered by their decreasing weight, without throwing away edges that are situated inside newly inserted Voronoi regions; see Figure 6.



(A)



(B)

FIGURE 6: In (A) a MWVD is illustrated in which the corresponding weights are written next to the site locations. Furthermore, (B) shows the respective overlay arrangement.

Furthermore, every point $p \in \mathbb{R}^2$ is associated with a *candidate set* that includes all sites of S whose Voronoi region may potentially include p. Kaplan et al. [KRS11] show that the expected complexity of $\mathcal{OA}(S)$ is bounded by $\mathcal{O}(n \log n)$. Additionally, the candidate set inside every face of $\mathcal{OA}(S)$ is fixed and its size is $\mathcal{O}(\log n)$, with high probability. They also sketch a corresponding construction strategy that runs in expected $\mathcal{O}(n \log^3 n)$ time and uses the algorithm of Aurenhammer and Edelsbrunner [AE84] as a subroutine.

1.1.4 ANISOTROPIC VORONOI DIAGRAMS

Labelle and Shewchuk [LS03] introduce the *anisotropic Voronoi diagram* to generate anisotropic triangular meshes. Additionally, they sketch an incremental construction strategy that allows computing loose anisotropic Voronoi diagrams that potentially include fewer orphans than the corresponding anisotropic Voronoi diagram. The *multiplicatively weighted crystal-growth Voronoi diagram (CGVD)*, whose Voronoi regions are orphan-free, is introduced by Schaudt and Drysdale [SD91]. In this setup, the distance between a site and a point in its Voronoi region equals the length of the shortest path that is completely situated in the respective Voronoi region. They also present a corresponding strategy that computes an approximation of the CGVD in $\mathcal{O}(n^3)$ time.

1.1.5 GENERALIZED WEIGHTED VORONOI DIAGRAMS

Held et al. [HHP16] use the *generalized weighted Voronoi diagram (GWVD)* of a set S of sites, that consists of points and straight-line segments, to generate *variable-radius offsets*; see Figure 7. These variable-radius offsets are exclusively formed by straight-line segments and circular arcs.

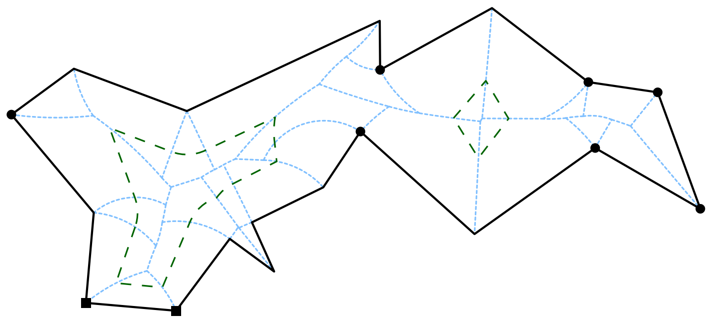


FIGURE 7: The GWVD inside a polygon that was generated by Held et al. [HHP16].

Every point site $s \in S$ is associated with a weight $w(s)$. If $\overline{pq}$ is an input straight-line segment, then the weight changes linearly from $w(p)$ to $w(q)$ along $\overline{pq}$. The Voronoi regions of the GWVD are not necessarily connected. Its edges are either formed by conic sections or semi-algebraic sets. Additionally, the set of permissible input sites can be extended to circular arcs.

1.1.6 APPLICATIONS

The Voronoi diagram is widely used in practice. Meguerdichian et al. [Meg+01] employ Voronoi diagrams to solve the coverage problem for sensor networks. In particular, they use it to identify the maximum breach and maximum support paths, that is, the paths inside the underlying field such that for any point along the respective path the distance to the closest sensor is maximized or minimized, respectively. Held and Palfrader [HP21] model coverage areas under the assumption that the individual sites emit signals at distinct strengths. Additionally, these signals are allowed to vary in different directions.

Okabe et al. [Oka+08] present *generalized network Voronoi diagrams* of points, straight-line segments, and polygons. This structure is used to model the shortest distances between specific places in an urban environment. By adding weights, the network Voronoi diagram can be generalized even further. From a practical point of view, this is important for applications, where the distance to a certain location is not the only relevant factor.

An interesting application of the medial axis is to generate tool paths for high-speed machining (HSM). In this context a *pocket* is the area along a workpiece in which material should be removed. In contrast to conventional pocket machining[1], optimizing the length of the tool path is no longer as important when it comes to generating HSM tool paths, other factors such as the maximum cutter engagement angle or the smoothness of the tool path are becoming more focused.

Elber et at. [ECD05] use the medial axis inside a pocket to generate $\mathcal{C}^1$-continuous HSM tool paths. Additionally, Held and Spielberger [HS09] derive a series of wavefronts from the medial axis. Afterwards, they interpolate between neighboring wavefronts and insert smoothing arcs to generate a $\mathcal{G}^1$-continuous tool path. They are even able to achieve $\mathcal{C}^2$ continuity by approximating their tool paths by curves that consist of uniform cubic B-splines [HH08]. Later on, they extended their strategy to be able to handle pockets with islands, i.e., parts inside the pocket that should not be machined [HS14]. Common to these tool path generation strategies is the requirement that a user-supplied *step-over* value, i.e., a maximum distance between two successive cutter passes, must be respected.

Additionally, the Voronoi diagram finds applications in city planning [Huf73; Kaz+17]. It is also frequently employed for automated land partitioning [DSS13]. In this scenario, it is desirable to reduce the fragmentation of the individual plots of land as much as possible.

[1]Note that Held [Hel91] provides an in-depth discussion of this topic.

1.2 Straight Skeletons

Peschka [Pes77] was the first to mention straight skeletons; see Figure 8. Over a century passed until they were rediscovered and formalized by Aichholzer et al. [Aic+95]. The straight skeleton has a linear combinatorial complexity. Unlike the Voronoi diagram, which can be defined based on an explicit distance function, the straight skeleton can only be defined procedurally.

The arcs of the straight skeleton inside a polygon P are traced out by the vertices of a shrinking wavefront that covers an increasing portion of the interior of P as time progresses. Initially, the wavefront coincides with P. Every edge of P generates one wavefront edge. All of these wavefront edges move inwards at unit speed. Two different event types indicate topological changes of the wavefront. An *edge event* occurs whenever a wavefront edge disappears. Additionally, a *split event* takes place if a reflex wavefront vertex crashes into a wavefront edge. The straight skeleton has some obvious similarities to the medial axis inside a polygon. The major difference between the two structures is that the straight skeleton consists exclusively of straight-line segments, whereas the medial axis may incorporate parabolic arcs. Note that this concept can also be extended to PSLGs.

Aichholzer and Aurenhammer [AA96] present a triangulation-based algorithm to compute the straight skeleton of a PSLG in $\mathcal{O}(n^3 \log n)$ time and $\mathcal{O}(n)$ space, although for most practical inputs this bound turns out to be far too pessimistic. Additionally, they show that every face of the straight skeleton of a PSLG is a monotone polygon.

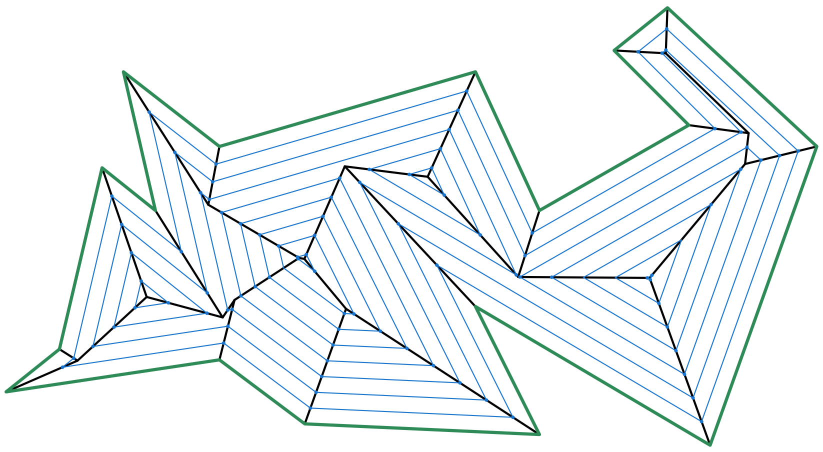


Figure 8: The straight skeleton inside a polygon as well as a series of wavefronts that are highlighted in blue. This straight skeleton was generated using the Surfer2 package [EHP20].

1.2.1 WEIGHTED STRAIGHT SKELETONS

The straight skeleton can be generalized by assigning additive or multiplicative weights with the input edges. In case of additive weights, the individual wavefront edges may start at different points in time. If multiplicative weights are associated with the input edges, then the wavefront edges are allowed to move at different speeds.

Biedl et al. [Bie+15a] show that the additively weighted straight skeleton can be computed in $\mathcal{O}(n \log n)$ time inside a monotone polygon. Additionally, Eder et al. [EHP20] provide an implementation of this strategy as well as the algorithm by Aichholzer and Aurenhammer [AA96] that is based on exact arithmetic using the Computational Geometry Algorithms Library (CGAL) [CGA21]. They also offer an in-depth discussion of the challenges that they faced as well as a thorough experimental evaluation.

Eppstein and Erickson [EE99] were the first to present an algorithm for computing straight skeletons inside a polygon that achieves a sub-quadratic runtime. In particular, their strategy takes $\mathcal{O}(n^{17/11+\varepsilon})$ time and space, for any fixed $\varepsilon > 0$. Additionally, their approach can also be used for generating multiplicatively weighted straight skeletons in $\mathcal{O}(n^{8/5+\varepsilon})$ time and space. They introduce the so-called *motorcycle graph* to speed up the computation of split events. A detailed analysis of the properties of weighted straight skeletons is given by Bield et al. [Bie+15b].

1.2.2 APPLICATIONS

Straight skeletons are employed for generating three-dimensional roof and terrain models. Held and Palfrader [HP17] derive different types of roofs from weighted straight skeletons inside polygons. Every point p inside the respective polygon P is lifted to $\mathbb{R}^3$ by assigning an additional z-coordinate to it that equals the time at which the wavefront reaches p. These roofs have the property that they do not include local minima. Hence, rain that hits the surface of such a roof is guaranteed to drain towards the boundary of P.

Interestingly enough, straight skeletons can also be used in the field of mathematical origami [DDL98]. In particular, they are used to find solutions of the *fold-and-cut problem* that can be summarized as follows: Imagine a polygon P that is drawn on a sheet of paper. We are now allowed to perform a series of foldings. Finally, we want to be able to take a pair of scissors and perform a single straight cut to produce the shape that is outlined by P. The respective crease patterns can be derived from the straight skeleton.

So-called *mitered offsets* find applications in NC machining. Park and Chung [PC03] argue that *rounded offset curves*, that are derived from Voronoi diagrams, are not well-suited for profile machining. In this specific scenario, it is especially important to avoid erosion at the reflex nodes of the respective pocket. Mitered offsets reduce the time that the tool is in contact with these reflex nodes significantly compared to rounded offsets. Palfrader and Held [PH15] describe how mitered offsets can be generated efficiently using straight skeletons.

Haunert and Sester [HS08] utilize straight skeletons in terms of cartography. When scaling down topographic databases, it is usually necessary to eliminate certain features and divide up their area among their neighbors. Straight skeletons are well-suited to implement such a *collapse operation* as they preserve topological relationships.

1.3 CONVEX PARTITIONS

Oftentimes, a problem that seems simple at first glance turns out to computationally costly in the end. But what does it even mean for a problem to be *hard*? A way to determine the hardness of a decision problem[2], i.e., a problem in which the output for all of its instances is either *yes* or *no*, is to associate it with a so-called *problem class*. Two of the most prominent examples of such problem classes are $\mathcal{P}$ and $\mathcal{NP}$. The problem class $\mathcal{P}$ contains all decision problems that are solvable in polynomial time by a deterministic algorithm. If a decision problem is solvable by a non-deterministic algorithm in polynomial time, then it is in $\mathcal{NP}$. Naively spoken, $\mathcal{NP}$ includes all decision problems for which it is easy to verify a proposed solution, but potentially hard to find a valid solution. One of the most famous problems in modern mathematics is to prove or disprove whether $\mathcal{P} = \mathcal{NP}$. In fact, it has even earned a spot among the *Millennium Prize Problems* [CJW06]. Additionally, a problem is said to be $\mathcal{NP}$-hard if it is at least as hard as any problem in $\mathcal{NP}$. Unless $\mathcal{P} = \mathcal{NP}$, our best chance for dealing with an optimization problem that is $\mathcal{NP}$-hard sufficiently fast is to find a suitable approximation scheme.

This section focuses on the MINIMUM CONVEX PARTITION (MCP) problem in which we are given a set S of n points and want to compute a plane graph with vertex set S, with each point in S having positive degree, that partitions the convex hull of S into the smallest possible number of convex faces; see Figure 9. Although the hardness of MCP is unknown for points in general position, i.e., where no three points are collinear, Grelier [Gre20] announced that if we allow three points to lie on the same line, then MCP is $\mathcal{NP}$-hard. If the points in S are assumed to be in general position, then Knauer and Spillner [KS06] provide a 3-approximation that runs in $\mathcal{O}(n \log n)$ time. They also propose an algorithm that improves the approximation factor to $^{30}/_{11}$

[2]Note that we can transform any optimization problem into a decision problem.

and takes $\mathcal{O}(n^2)$ time. Whenever the points of S are situated on h nested convex hulls, MCP can be solved exactly in $\mathcal{O}(n^{3h+3})$ time [FMR01].

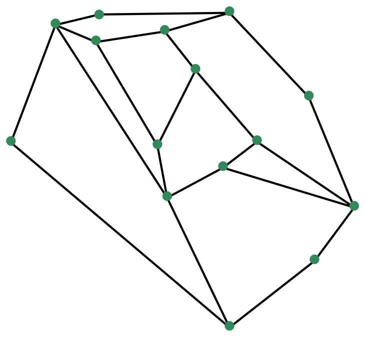


FIGURE 9: A solution of the MCP problem for an input set with 15 points.

A problem that is related to MCP is the decomposition of a simple polygon P, with n nodes, into convex pieces. If P includes holes, then Lingas [Lin82] shows that this problem is $\mathcal{NP}$-hard.

Otherwise, polynomial-time algorithms exist. Chazelle and Dobkin [CD79] present an algorithm that runs in $\mathcal{O}(n^6)$ time if Steiner points are allowed to be added. In his PhD thesis, Chazelle [Cha80] introduces a strategy that is based on dynamic programming and solves the problem in $\mathcal{O}(n + r^3)$ time, where r is the number of reflex nodes of P. If no Steiner points are allowed to be inserted, then Keil and Snoeyink [KS02] present an algorithm that generates minimum convex decompositions of simple polygons in $\mathcal{O}(n + \min\{nr^2, r^4\})$ time.

2 CONTRIBUTION

This chapter provides an overview of the main contributions of this work that are cumulated in Part II. In Section 2.1 we take a look at a tool path generation algorithm that utilizes the medial axis. Section 2.2 summarizes the strategy for generating convex decompositions and reducing their face count that is based on several heuristics. An efficient algorithm for computing the MWVD of points in $\mathbb{R}^2$ is discussed in Section 2.3. Held et al. [HHP16] further generalize the MWVD of points, straight-line segments, and circular arcs to generate variable-radius offsets. We build on their work and introduce the *variable-radius skeleton (VRS)* inside a polygon; see Section 2.4. Finally, in Section 2.5 a rundown of our results in respect to the recognition and reconstruction of weighted bisector graphs and MWVDs of points is given.

2.1 GENERATION OF SPIRAL-LIKE PATHS WITHIN PLANAR SHAPES

In *On the Generation of Spiral-Like Paths Within Planar Shapes* [HL18] (see Page 31), we are given a planar shape P in $\mathbb{R}^2$ that is bounded by a set of straight-line segments and circular arcs and want to compute a tool path that has the following properties:

(1) It starts inside of P, ends on the boundary of P, and is completely situated within P.

(2) A user-specified step-over value $\Delta > 0$ is respected along the entire tool path.

(3) The tool path is free of self-intersections and consists exclusively of straight-line segments.

This paper is based on prior work by Held and Spielberger [HS09; HS14]. The medial axis $\mathcal{MA}(P)$ inside P forms the basis of our approach. By sampling $\mathcal{MA}(P)$ such that Δ is respected, a discretized version $\mathcal{MA}''(P)$ of the medial axis is created that is enhanced by a series of so-called *clearance lines*. Note that the medial axis forms a tree inside P. Therefore, by choosing a vertex r of $\mathcal{MA}''(P)$ as the root, $\mathcal{MA}''(P)$

can be transformed into a rooted tree $\mathcal{T}_r$ which is called the *discrete medial axis tree*. Afterwards, a series of uniformly distributed wavefronts is generated by sending an impulse through $\mathcal{T}_r$ in which the Hausdorff distance between two successive wavefronts is bounded by Δ. Finally, we interpolate between consecutive wavefronts to create a closed spiral path.

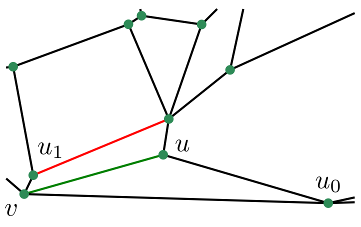
This strategy can be modified to generate double spirals, i.e., tool paths that start and end along the boundary of P. Sometimes it is advantageous to subdivide P into several regions and create a double spiral inside each of them. These double spirals can then be linked such that a single composite spiral path is created. Additionally, the POWERAPX package [HH08; HK14] can be utilized to derive $\mathcal{C}^2$-continuous tool paths from our polygonal ones. These enhanced tool paths are well-suited for HSM.

2.2 COMPUTING LOW-COST CONVEX PARTITIONS

The goal of the *2020 Computational Geometry Challenge* was to find solutions to the MCP problem. Thus, several problem instances, i.e., sets of points in $\mathbb{R}^2$, were given. In our corresponding publication *Computing Low-Cost Convex Partitions for Planar Point Sets Based on Tailored Decompositions* [Ede+20] (see Page 43), we present different strategies for computing initial decompositions as well as several heuristics that improve upon them. At first, we worked on an implementation of the following simple idea: The input point set S is triangulated and edges are removed as long as the respective faces stay convex. Various heuristics are used to reduce the face count of these initial decompositions which can be summarized as follows:

- A face of the decomposition as well as a random number of its neighbors is selected. This set of faces is called a *hole*. Afterwards, the initial triangulation edges inside this hole are restored and, again, dropped randomly. If the newly generated decomposition has fewer faces than the old one, then we keep it.
- Random edge flips are performed before we start to remove edges.
- We load previously computed decompositions to work on them again.
- The individual decompositions are subdivided into several non-overlapping regions that can then be improved in parallel.
- We select high-degree vertices of decompositions and rotate edges away from them. As a consequence, certain edges may become unnecessary; see Figure 10.

Additionally, a significant portion of the underlying problem instances contained a high number of collinear points that are situated on vertical or horizontal lines. Thus, for these specific inputs, we generate custom-made initial decompositions. More precisely, we connect points that share a common x-coordinate ordered by increasing y-coordinate. Subsequently, the top and bottom bounding chains of S are created and



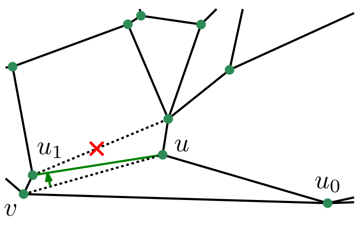


FIGURE 10: The green edge is flipped away from v. Therefore, the red edge becomes unnecessary.

the area between these chains as well as the convex hull of S is triangulated. We proceed similarly for points that share a common y-coordinate.

2.3 EFFICIENT MULTIPLICATIVELY WEIGHTED VORONOI DIAGRAMS

In *An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams* [HL20] (see Page 57), we use a practice-minded wavefront-based approach to generate the MWVD of a set S of n weighted points. The wavefront $\mathcal{WF}(S,t)$ of S at time $t \geq 0$ includes all points in $\mathbb{R}^2$ that are in minimal weighted distance t to S. It consists of so-called *wavefront arcs* as well as the common endpoints of two wavefront arcs that we will refer to as a *wavefront vertices*. Additionally, each input site $s_i \in S$ is associated with an *offset circle* $c_i(t)$ at time $t \geq 0$ that is defined as

$$c_i(t) := \{p \in \mathbb{R}^2 : d_w(p,s_i) = t\},$$

with

$$d_w(p,s_i) := \frac{d(p,s_i)}{w(s_i)},$$

where $w(s_i) > 0$ is the weight that is associated with s_i. Throughout the wavefront propagation, we identify and eliminate arcs which are situated along these offset circles that are inactive, i.e., arcs that will not contribute to the wavefront at any future point in time. Thus, it is only necessary to keep track of the *active arcs*; see Figure 11. Note that an active arc does not necessarily coincide with a wavefront arc. Collision, domination, and arc events mark topological changes of the arrangement of active arcs. One may prove that at most $\mathcal{O}(n^2)$ many events may take place in the worst case. Additionally, each event can be handled in $\mathcal{O}(\log n)$ time. Therefore, our basic event-based strategy is able to generate the MWVD of S in worst-case $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space.

Our experiments quickly indicated that the quadratic bound on the number of collision and domination events is far too pessimistic in practice. Thus, we make use of

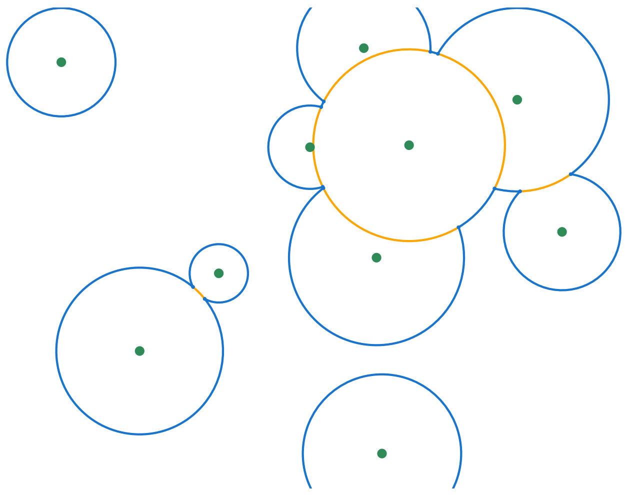


FIGURE 11: A snapshot of the wavefront propagation at time t in which the wavefront $\mathcal{WF}(S,t)$ is highlighted in blue and active arcs that are not part of $\mathcal{WF}(S,t)$ are shown in orange.

an overlay arrangement to improve our basic algorithm. This enhanced strategy computes the MWVD in expected $\mathcal{O}(n \log^4 n)$ time and $\mathcal{O}(n \log^3 n)$ space. We also have implemented this algorithm based on exact arithmetic using CGAL. Our approach can be generalized to a set of disjoint straight-line segments in the plane. Additionally, the basic strategy can be extended to handle additive and multiplicative weights simultaneously by giving sites that are associated with an additive weight a head start.

2.4 WEIGHTED SKELETAL STRUCTURES FOR VARIABLE-RADIUS OFFSETS

Held et al. [HHP16] present the *generalized weighted Voronoi diagram (GWVD)* to generate variable-radius offsets. In contrast to conventional constant-radius offsetting, variable-radius offsetting allows for different parts of the input to shrink (or expand) non-uniformly at different speeds. In our work *Weighted Skeletal Structures for Computing Variable-Radius Offsets* [HL21] (see Page 75), we present a wavefront-based algorithm for computing the GWVD of a set of points and straight-line segments in $\mathcal{O}(n^3 \log n)$ time and $\mathcal{O}(n^3)$ space. A possible drawback of the GWVD is that its regions are potentially disconnected even within a polygon. Thus, we present an alternative structure inside a polygon whose regions stay connected in any case. We call it the *variable-radius skeleton (VRS)*; see Figure 12.

Two slightly different kinds of of variable-radius offsets can be derived from the GWVD as well as the VRS. Like the straight skeleton, the VRS is defined procedurally based on several event types. We provide a wavefront-based strategy for computing the VRS that runs in cubic time. There are several possible applications of the VRS such

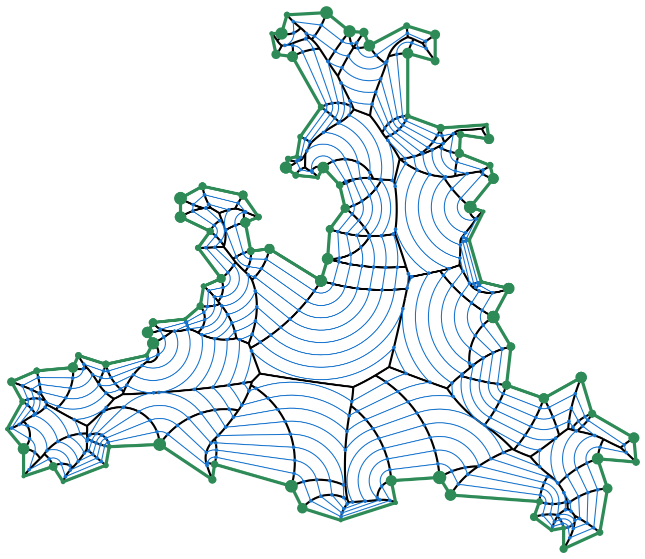


FIGURE 12: A VRS inside a polygon as well as a series of uniformly distributed variable-radius offsets.

as brush-stroke modeling or the creation of ornamental seams. Furthermore, it is possible to derive so-called *variable-radius roofs* from the VRS that are guaranteed to drain water, that is, they do no include local minima; see Figure 13.

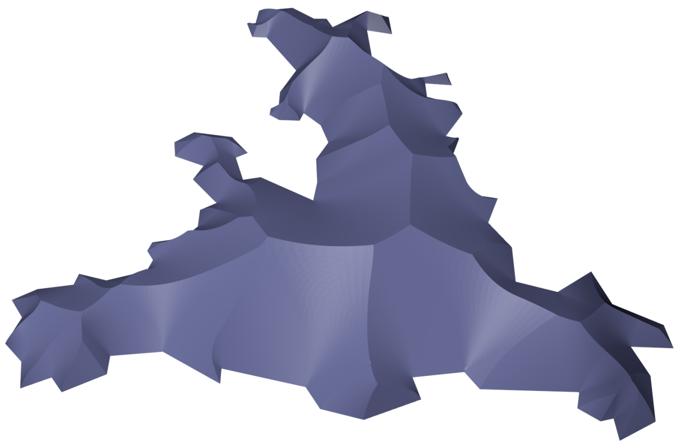


FIGURE 13: A variable-radius roof.

2.5 Recognition and Reconstruction of Weighted Voronoi Diagrams

We refer to a geometric graph as a *weighted bisector graph* if all of its faces are bounded by arcs that are formed by parts of multiplicatively weighted bisectors between pairs of sites. A vast amount of literature deals with the computation of various types of weighted bisector graphs; see Sections 1.1.3 and 1.2.1. The reverse problem is studied in *On the Recognition and Reconstruction of Weighted Voronoi Diagrams and Bisector Graphs* [Ede+21] (see Page 93). More precisely, we are given a *planar circular-arc graph* $\mathcal{G}$, that is, a planar geometric graph in which each edge is formed by a circular arc which either forms a full circle or ends in a node that has a degree of at least three. Now, we want to identify a solution set (S, σ), i.e., a set of points S in the plane together with their corresponding weight function σ, such that $\mathcal{G}$ is a weighted bisector graph induced by (S, σ) whenever such a solution exists.

Assume that m denotes the number of faces of $\mathcal{G}$. If $\mathcal{G}$ consists exclusively of disjoint nested circles and lines, then, by recursively applying circular inversion, it is possible to determine (S, σ) in $\mathcal{O}(m \log m)$ time such that $\mathcal{G}$ is the MWVD defined by (S, σ).

Whenever $\mathcal{G}$ contains nodes, we use the conjugated Möbius transform to derive several *solution circles* from a node v. Every solution circle contains all possible locations for a specific defining site of v. If the number of nodes along the boundary of a face of $\mathcal{G}$ is greater than or equal to three, then we are able to obtain the respective sites by intersecting the individual solution circles. Otherwise, families of local solutions are generated that are combined using the procedure that we apply when dealing exclusively with nested circles and lines.

We show that it can be determined in $\mathcal{O}(m)$ time whether a circular-arc graph $\mathcal{G}$, that is of regular degree three, is a bisector graph and, if so, find a corresponding solution (S, σ). Clearly, if $\mathcal{G}$ is indeed a bisector graph, then the strategy by Aurenhammer and Edelsbrunner [AE84] can be utilized to decide whether $\mathcal{G}$ is also a MWVD.

Bibliography

[AA96] Oswin Aichholzer and Franz Aurenhammer. “Straight Skeletons for General Polygonal Figures in the Plane”. In: *International Computing and Combinatorics Conference*. Springer. 1996, pp. 117–126.
DOI: 10.1007/3-540-61332-3_144.

[AE84] Franz Aurenhammer and Herbert Edelsbrunner. “An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane”. In: *Pattern Recognition* 17.2 (1984), pp. 251–257.
DOI: 10.1016/0031-3203(84)90064-5.

[Aic+95] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. “A Novel Type of Skeleton for Polygons”. In: *Journal of Universal Computer Science*. Springer, 1995, pp. 752–761.
DOI: 10.1007/978-3-642-80350-5_65.

[AK00] Franz Aurenhammer and Rolf Klein. *Voronoi Diagrams*. Vol. 5. Elsevier, 2000.

[Aur87] Franz Aurenhammer. “Power Diagrams: Properties, Algorithms and Applications”. In: *SIAM Journal on Computing* 16.1 (1987), pp. 78–96.
DOI: 10.1137/0216006.

[Bie+15a] Therese Biedl, Martin Held, Stefan Huber, Dominik Kaaser, and Peter Palfrader. “A Simple Algorithm for Computing Positively Weighted Straight Skeletons of Monotone Polygons”. In: *Information Processing Letters* 115.2 (Feb. 2015), pp. 243–247.
DOI: 10.1016/j.ipl.2014.09.021.

[Bie+15b] Therese Biedl, Martin Held, Stefan Huber, Dominik Kaaser, and Peter Palfrader. “Weighted Straight Skeletons in the Plane”. In: *Computational Geometry: Theory and Applications* 48.2 (Feb. 2015), pp. 120–133.
DOI: 10.1016/j.comgeo.2014.08.006.

[Blu67] Harry Blum. “A Transformation for Extracting New Descriptors of Shape”. In: *Models for the Perception of Speech and Visual Form* 4 (1967), pp. 362–380.

[Boo80] Barry N Boots. “Weighting Thiessen Polygons”. In: *Economic Geography* 56.3 (1980), pp. 248–259.
DOI: 10.2307/142716.

[Bro79] Kevin Q Brown. “Geometric Transforms for Fast Geometric Algorithms”. PhD thesis. Carnegie Mellon University, 1979.

[CD79] Bernard Chazelle and David Dobkin. “Decomposing a Polygon into its Convex Parts”. In: *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC 1979)*. Apr. 1979, pp. 38–48.
DOI: 10.1145/800135.804396.

[CGA21] CGAL Project. *CGAL User and Reference Manual*. 5.2.1. CGAL Editorial Board, 2021.
URL: https://doc.cgal.org/5.2.1/Manual/packages.html.

[Cha80] Bernard M Chazelle. “Computational Geometry and Convexity”. PhD thesis. Yale University, 1980.

[Che90] L Paul Chew. *Building Voronoi Diagrams for Convex Polygons in Linear Expected Time*. Tech. rep. Dartmouth College Hanover, NH, USA, 1990.

[CJW06] James A Carlson, Arthur Jaffe, and Andrew Wiles. *The Millennium Prize Problems*. American Mathematical Society, 2006. ISBN: 978-0821836798.

[CS89] Kenneth L Clarkson and Peter W Shor. “Applications of Random Sampling in Computational Geometry, II”. In: *Discrete & Computational Geometry* 4.5 (1989), pp. 387–421.
DOI: 10.1007/BF02187740.

[DDL98] Erik D Demaine, Martin L Demaine, and Anna Lubiw. “Folding and Cutting Paper”. In: *Proceedings of the Japanese Conference on Discrete and Computational Geometry (JCDCG 1998)*. Springer. 1998, pp. 104–118.
DOI: 10.1007/978-3-540-46515-7_9.

[Des44] René Descartes. *Principia Philosophiae*. Ludovicus Elzevirius, 1644.

[Dir50] G Lejeune Dirichlet. “Über die Reduction der Positiven Quadratischen Formen mit Drei Unbestimmten Ganzen Zahlen.” In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 40 (1850), pp. 209–227.
DOI: 10.1515/crll.1850.40.209.

[DK97] Frank Dehne and Rolf Klein. ““The Big Sweep” : On the Power of the Wavefront Approach to Voronoi Diagrams”. In: *Algorithmica* 17.1 (1997), pp. 19–32.
DOI: 10.1007/BF02523236.

[DSS13] Demetris Demetriou, Linda See, and John Stillwell. “A Spatial Genetic Algorithm for Automating Land Partitioning”. In: *International Journal of Geographical Information Science* 27.12 (2013), pp. 2391–2409. DOI: 10.1080/13658816.2013.819977.

[ECD05] Gershon Elber, Elaine Cohen, and Sam Drake. “MATHSM: Medial Axis Transform Toward High Speed Machining of Pockets”. In: *Computer-Aided Design* 37.2 (2005), pp. 241–250. DOI: 10.1016/j.cad.2004.05.008.

[Ede+20] Günther Eder, Martin Held, Stefan de Lorenzo, and Peter Palfrader. “Computing Low-Cost Convex Partitions for Planar Point Sets Based on Tailored Decompositions”. In: *Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020)*. Vol. 164. Leibniz International Proceedings in Informatics (LIPIcs). Zürich, Switzerland: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, June 2020, 85:1–85:10. ISBN: 978-3-95977-143-6. DOI: 10.4230/LIPIcs.SoCG.2020.85.

[Ede+21] Günther Eder, Martin Held, Stefan de Lorenzo, and Peter Palfrader. “On the Recognition and Reconstruction of Weighted Voronoi Diagrams and Bisector Graphs”. Submitted to *Computational Geometry: Theory and Applications*. Feb. 2021.

[EE99] David Eppstein and Jeff Erickson. “Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions”. In: *Discrete & Computational Geometry* 22.4 (1999), pp. 569–592. DOI: 10.1007/PL00009479.

[EHP20] Günther Eder, Martin Held, and Peter Palfrader. “On Implementing Straight Skeletons: Challenges and Experiences”. In: *Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020)*. Vol. 164. LIPIcs. Zürich, Switzerland: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, June 2020, 38:1–38:16. ISBN: 978-3-95977-143-6. DOI: 10.4230/LIPIcs.SoCG.2020.38.

[FMR01] Thomas Fevens, Henk Meijer, and David Rappaport. “Minimum Convex Partition of a Constrained Point Set”. In: *Discrete Applied Mathematics* 109.1-2 (2001), pp. 95–107. DOI: 10.1016/S0166-218X(00)00237-7.

[For87] Steven Fortune. “A Sweepline Algorithm for Voronoi Diagrams”. In: *Algorithmica* 2.1-4 (1987), p. 153. DOI: 10.1007/BF01840357.

[GKS90] Leonidas J Guibas, Donald E Knuth, and Micha Sharir. "Randomized Incremental Construction of Delaunay and Voronoi Diagrams". In: *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming (ICALP 1990)*. Springer. 1990, pp. 414–431.
DOI: 10.1007/BF01758770.

[Gre20] Nicolas Grelier. *Hardness and Approximation of Minimum Convex Partition*. 2020.
arXiv: 1911.07697 [cs.CG].

[Hel91] Martin Held. *On the Computational Geometry of Pocket Machining*. Vol. 500. Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 1991.
DOI: 10.1007/3-540-54103-9.

[HH08] Martin Heimlich and Martin Held. "Biarc Approximation, Simplification and Smoothing of Polygonal Curves by Means of Voronoi-based Tolerance Bands". In: *International Journal of Computational Geometry & Applications* 18.03 (2008), pp. 221–250.
DOI: 10.1142/S0218195908002593.

[HH09] Martin Held and Stefan Huber. "Topology-Oriented Incremental Computation of Voronoi Diagrams of Circular Arcs and Straight-Line Segments". In: *Computer-Aided Design* 41.5 (2009), pp. 327–338.
DOI: 10.1016/j.cad.2008.08.004.

[HHP16] Martin Held, Stefan Huber, and Peter Palfrader. "Generalized Offsetting of Planar Structures using Skeletons". In: *Computer-Aided Design and Applications* 13.5 (2016), pp. 712–721.
DOI: 10.1080/16864360.2016.1150718.

[HK14] Martin Held and Dominik Kaaser. "$\mathcal{C}^2$ Approximation of Planar Curvilinear Profiles by Cubic B-Splines". In: *Computer-Aided Design and Applications* 11.2 (2014), pp. 206–219.
DOI: 10.1080/16864360.2014.846092.

[HL18] Martin Held and Stefan de Lorenzo. "On the Generation of Spiral-Like Paths Within Planar Shapes". In: *Journal of Computational Design and Engineering* 5.3 (July 2018), pp. 348–357.
DOI: 10.1016/j.jcde.2017.11.011.

[HL20] Martin Held and Stefan de Lorenzo. "An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi

Diagrams". In: *Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)*. Vol. 173. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Aug. 2020, 56:1–56:15. ISBN: 978-3-95977-162-7.
DOI: 10.4230/LIPIcs.ESA.2020.56.

[HL21] Martin Held and Stefan de Lorenzo. "Weighted Skeletal Structures for Computing Variable-Radius Offsets". In: *Computer-Aided Design and Applications* 18.5 (Jan. 2021), pp. 875–889.
DOI: 10.14733/cadaps.2021.875-889.

[HP17] Martin Held and Peter Palfrader. "Straight Skeletons with Additive and Multiplicative Weights and Their Application to the Algorithmic Generation of Roofs and Terrains". In: *Computer-Aided Design* 92 (Nov. 2017), pp. 33–41. ISSN: 0010-4485.
DOI: 10.1016/j.cad.2017.07.003.

[HP21] Martin Held and Peter Palfrader. "Modeling Coverage Areas of Anisotropic Transmitters by Voronoi-Like Structures". In: *Proceedings of the 18th Computer-Aided Design Conference (CAD 2021)*. Barcelona, Spain, July 2021, pp. 283–287.
DOI: 10.14733/cadconfP.2021.283-287.

[HR15] Sariel Har-Peled and Benjamin Raichel. "On the Complexity of Randomly Weighted Multiplicative Voronoi Diagrams". In: *Discrete & Computational Geometry* 53.3 (2015), pp. 547–568.
DOI: 10.1007/s00454-015-9675-0.

[HS08] Jan-Henrik Haunert and Monika Sester. "Area Collapse and Road Centerlines based on Straight Skeletons". In: *GeoInformatica* 12.2 (2008), pp. 169–191.
DOI: 10.1007/s10707-007-0028-x.

[HS09] Martin Held and Christian Spielberger. "A Smooth Spiral Tool Path for High Speed Machining of 2D Pockets". In: *Computer-Aided Design* 41.7 (2009), pp. 539–550.
DOI: 10.1016/j.cad.2009.04.002.

[HS14] Martin Held and Christian Spielberger. "Improved Spiral High-Speed Machining of Multiply-Connected Pockets". In: *Computer-Aided Design and Applications* 11.3 (2014), pp. 346–357.
DOI: 10.1080/16864360.2014.863508.

[Huf73] David L Huff. "The Delineation of a National System of Planning Regions on the Basis of Urban Spheres of Influence". In: *Regional Studies* 7.3 (1973), pp. 323–329.
DOI: 10.1080/09595237300185321.

[Kaz+17] Ali Kazemzadeh-Zow, Saeed Zanganeh Shahraki, Luca Salvati, and Najmeh N Samani. “A Spatial Zoning Approach to Calibrate and Validate Urban Growth Models”. In: *International Journal of Geographical Information Science* 31.4 (2017), pp. 763–782.
DOI: 10.1080/13658816.2016.1236927.

[KRS11] Haim Kaplan, Edgar Ramos, and Micha Sharir. “The Overlay of Minimization Diagrams in a Randomized Incremental Construction”. In: *Discrete & Computational Geometry* 45.3 (2011), pp. 371–382.
DOI: 10.1007/s00454-010-9324-6.

[KS02] Mark Keil and Jack Snoeyink. “On the Time Bound for Convex Decomposition of Simple Polygons”. In: *International Journal of Computational Geometry & Applications* 12.03 (2002), pp. 181–192.
DOI: 10.1142/S0218195902000803.

[KS06] Christian Knauer and Andreas Spillner. “Approximation Algorithms for the Minimum Convex Partition Problem”. In: *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*. Springer. 2006, pp. 232–241.
DOI: 10.1007/11785293_23.

[Lee82] Der-Tsai Lee. “Medial Axis Transformation of a Planar Shape”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4 (1982), pp. 363–369.
DOI: 10.1109/TPAMI.1982.4767267.

[Lin82] Andrzej Lingas. “The Power of Non-Rectilinear Holes”. In: *Proceedings of the 9th International Colloquium on Automata, Languages, and Programming (ICALP 1982)*. Springer. 1982, pp. 369–383.
DOI: 10.1007/BFb0012784.

[LS03] Francois Labelle and Jonathan Richard Shewchuk. “Anisotropic Voronoi Diagrams and Guaranteed-Quality Anisotropic Mesh Generation”. In: *Proceedings of the 19th Annual Symposium on Computational Geometry (SoCG 2003)*. 2003, pp. 191–200.
DOI: 10.1145/777792.777822.

[Meg+01] Seapahn Meguerdichian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B Srivastava. “Coverage Problems in Wireless Ad-Hoc Sensor Networks”. In: *Proceedings 20th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2001)*. Vol. 3. IEEE. 2001, pp. 1380–1387.
DOI: 10.1109/INFCOM.2001.916633.

[Oka+08] Atsuyuki Okabe, Toshiaki Satoh, Takehiro Furuta, Atsuo Suzuki, and Kyoko Okano. "Generalized Network Voronoi Diagrams: Concepts, Computational Methods, and Applications". In: *International Journal of Geographical Information Science* 22.9 (2008), pp. 965–994.
DOI: 10.1080/13658810701587891.

[PC03] Sang C Park and Yun C Chung. "Mitered Offset for Profile Machining". In: *Computer-Aided Design* 35.5 (2003), pp. 501–505.
DOI: 10.1016/S0010-4485(02)00065-9.

[Pes77] Gustav A V Peschka. *Kotirte Ebenen (Kotirte Projektionen) und deren Anwendung: Vorträge: zum Gebrauche für Ingenieure, für höhere Lehranstalten und zum Selbststudium*. Buschak & Irrgang, 1877.
DOI: 10.14463/GBV:865177619.

[PH15] Peter Palfrader and Martin Held. "Computing Mitered Offset Curves Based on Straight Skeletons". In: *Computer-Aided Design and Applications* 12.4 (2015), pp. 414–424.
DOI: 10.1080/16864360.2014.997637.

[SD91] Barry F Schaudt and Robert L S Drysdale. "Multiplicatively Weighted Crystal Growth Voronoi Diagrams". In: *Proceedings of the 7th Annual Symposium on Computational Geometry (SoCG 1991)*. ACM. 1991, pp. 214–223.
DOI: 10.1145/109648.109672.

[SH75] Michael I Shamos and Dan Hoey. "Closest-Point Problems". In: *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science (SFCS 1975)*. IEEE. 1975, pp. 151–162.
DOI: 10.1109/SFCS.1975.8.

[Thi11] Alfred H Thiessen. "Precipitation Averages for Large Areas". In: *Monthly Weather Review* 39.7 (1911), pp. 1082–1089.
DOI: 10.1175/1520-0493(1911)39<1082b:PAFLA>2.0.CO;2.

[Vor08] Georges Voronoi. "Nouvelles Applications des Paramètres Continus à la Théorie des Formes Quadratiques. Deuxième Mémoire: Recherches sur les Parallélloèdres Primitifs." In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 134 (1908), pp. 198–287.
DOI: 10.1515/crll.1908.134.198.

[Yap87] Chee-Keng Yap. "An $\mathcal{O}(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments". In: *Discrete & Computational Geometry* 2.4 (1987), pp. 365–393.
DOI: 10.1007/BF02187890.

[ZL02] Gabriel Zachmann and Elmar Langetepe. “Geometric Data Structures for Computer Graphics”. In: *Eurographics 2002 - Tutorials*. Eurographics Association, 2002.
DOI: 10.2312/egt.20021064.

II

Publications

On the Generation of Spiral-Like Paths within Planar Shapes

Martin Held and Stefan de Lorenzo [HL18]

Published in:
Journal of Computational Design and Engineering

July 2018

[HL18]

Martin Held and Stefan de Lorenzo. "On the Generation of Spiral-Like Paths Within Planar Shapes". In: *Journal of Computational Design and Engineering* 5.3 (July 2018), pp. 348–357. DOI: 10.1016/j.jcde.2017.11.011

Contents lists available at ScienceDirect

Journal of Computational Design and Engineering

journal homepage: www.elsevier.com/locate/jcde

On the generation of spiral-like paths within planar shapes

Martin Held, Stefan de Lorenzo *

Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria

ARTICLE INFO

Article history:
Received 17 August 2017
Received in revised form 15 November 2017
Accepted 23 November 2017
Available online 24 November 2017

Keywords:
Spiral-like path
Double spiral
Composite spiral
Medial axis
Smoothing
High-speed machining

ABSTRACT

We simplify and extend prior work by Held and Spielberger [CAD 2009, CAD&A 2014] to obtain spiral-like paths inside of planar shapes bounded by straight-line segments and circular arcs: We use a linearization to derive a simple algorithm that computes a continuous spiral-like path which (1) consists of straight-line segments, (2) has no self-intersections, (3) respects a user-specified maximum step-over distance, and (4) starts in the interior and ends at the boundary of the shape. Then we extend this basic algorithm to double-spiral paths that start and end at the boundary, and show how these double spirals can be used to cover complicated planar shapes by composite spiral paths. We also discuss how to improve the smoothness and reduce the curvature variation of our paths, and how to boost them to higher levels of continuity.

1. Introduction

1.1. Motivation

Several applications require to cover a planar shape by moving a circular disk along a path. E.g., in machining applications the disk models the cross-section of a tool and the area models a so-called pocket. Similarly, the disk may represent the area covered by a spray nozzle or the area of visibility of a camera device used for aerial surveillance. In our study the planar shape may be bounded by one outer contour and possibly a number of island contours (contained within the outer contour), where each contour is formed by straight-line segments and circular arcs.

Traditional strategies for path generation include zigzag patterns and the use of offset curves to form contour-parallel patterns. See, e.g., Held (1991) for a detailed discussion of both strategies in the context of pocket machining.

Common to these traditional strategies is the fact that the resulting paths contain lots of sharp corners, i.e., abrupt changes of the direction. The higher the speed or the moment of inertia of the moving object represented by the disk, the more these directional discontinuities cause problems. E.g., for a high speed machining (HSM) application, an abrupt change of direction requires the tool to slow down to near-zero speed, change its direction and then accelerate until the desired maximum speed is reached again. In a machining application, sharp corners also lead to a high variation of the tool load.

1.2. Prior work

One way of generating a smooth continuous path is to rely on a traditional strategy and to reduce sharp directional discontinuities in a post-processing step: Pateloup, Duc, and Ray (2004) and Zhao, Wang, Zhou, and Qin (2007), Zhao, Liu, Zhang, Zhou, and Yu (2009) take a conventional tool path and smooth it by inserting circular fillet arcs.

Spiral-like paths are widely regarded as a suitable means for avoiding sharp directional discontinuities. Bieterman and Sandstrom (2002) present an approach based on partial differential equations (PDEs) to compute a spiral-like path inside a star-shaped pocket. Its border contour is successively offset inwards by evaluating the PDE at different points in time. Then these solution contours are connected through radial interpolation. Banerjee, Feng, and Bordatchev (2012) use a similar approach and solve the eigenvalue problem for an elliptic PDE. Neighboring contours are connected based on a winding-angle parameterization. In addition, they explain how to deal with one single island near the center of the planar shape.

Zhou, Zhao, Li, and Xia (2016) propose a strategy that produces smooth, double spirals which start as well as end at the boundary of the planar shape. A series of isothermal lines is derived from a parabolic PDE. By interpolating between successive isothermal lines a closed spiral-like path is produced. Embedding a second spiral-like path between adjacent revolutions of the initial one

Peer review under responsibility of Society for Computational Design and Engineering.
* Corresponding author.
E-mail addresses: held@cosy.sbg.ac.at (M. Held), slorenzo@cosy.sbg.ac.at (S. de Lorenzo).

https://doi.org/10.1016/j.jcde.2017.11.011
2288-4300/© 2017 Society for Computational Design and Engineering. Publishing Services by Elsevier.

yields the final double spiral. Multiply-connected input shapes, i.e., regions which contain islands, are subdivided into (nearly star-shaped) sub-shapes. One connected path is produced by computing a double spiral inside every sub-shape, and linking neighboring ones together at their ends.

Zhao et al. (2016) suggest space-filling curves ("Fermat spirals") to cover planar shapes. Another strategy which is based on space-filling curves is introduced by Romero-Carrillo, Torres-Jimenez, Dorado, and Díaz-Garrido (2015): An Archimedean spiral is deformed through linear morphing, and embedded into a convex two dimensional region.

Abrahamsen (2015) constructs a polygonal spiral-like path inside a planar shape bounded by straight-line segments. After calculating an enhanced medial axis tree, a sequence of uniformly distributed wavefronts is computed. Each wavefront is given by a sequence of vertices which are situated on the edges of the medial axis. A closed spiral-like path is generated by manipulating the positions of these vertices. The resulting path is then smoothed by inserting circular arcs at sharp corners.

Held and Spielberger (2009, 2014) generate spiral-like paths for general non-convex planar shapes with or without islands. A series of circles are placed on the medial axis whose radii increase as time progresses. Portions of these circles are interpolated and connected by other circular arcs to form a G^1-continuous path.

While our own work was originally motivated by an HSM application at our industrial partner, we note that a need for paths that cover specific areas while avoiding sharp directional discontinuities arises also outside of CAD/CAM: E.g., Chandler, Rasmussen, and Pachter (2010) insert fillet arcs into a polygonal path in order to take care of maneuverability constraints of an unmanned aerial vehicle. We refer to Keller (2017) for a recent detailed discussion of smooth paths for aerial surveillance.

1.3. Our contribution

Since the algorithm by Held and Spielberger (2009, 2014) is difficult to analyze theoretically and even more difficult to implement reliably, in this work we pick up their basic idea and simplify it significantly: A linearization of the medial axis of the input shape allows to come up with an algorithm for a polygonal spiral-like path that is easy to implement. (And, indeed, an implementation of this algorithm is already in commercial use at our industrial partner.) The path is continuous, without self-intersections, and respects a user-specified maximum "step-over" distance. This initial path is then smoothed by refining the positions of its vertices, which helps to reduce the curvature variation. It can be boosted to G^1-continuity or C^2-continuity by using an approximation by biarcs or cubic B-splines. (We use the PowerApx package (Heimlich & Held, 2008; Held & Kaaser, 2014).)

As in the work by Held and Spielberger (2009, 2014), our spiral-like paths start in the interior of the pocket and end at its boundary. The simplicity of our approach allows to generalize this scheme and to devise double-spiral paths that start and end at arbitrary points on the boundary. This makes it easier to cover a complex shape by one continuous spiral-like path by (1) decomposing the shape into simpler sub-areas, (2) computing a (double) spiral within every sub-area, and (3) linking the individual paths to form one continuous path. While such a double-spiral path is unsuited for machining, it may find use in other applications, such as layered manufacturing, spray painting, aerial surveillance, or path finding algorithms for search&rescue missions.

Our approach relies heavily on the medial axis of the input: (1) It serves as the key tool for capturing the geometry of the shape and for computing offset-like curves which form the basis for our paths. (2) It allows to determine an upper bound on the "step-over" distance between portions of the spiral. Besides its inherent simplicity, a major advantage of our approach is its generality: It can deal with arbitrarily complex planar shapes with and without islands, thereby guaranteeing a maximum "step-over" distance.

2. Preliminaries

Consider a planar input shape that is bounded by straight-line segments and circular arcs, and suppose that we want to move a disk of radius ρ inside this shape such that the area swept by the disk equals (most of) the shape. If the disk has to stay within the shape during the entire movement then it is obvious that its center can never get closer to the boundary of the shape than ρ, even if this constraint results in some areas of the shape being uncovered. (E.g., for a polygonal shape this will happen at convex vertices of the shape.) The loci of all permissible positions of the center of the disk can be obtained as the Minkowski difference of the shape and a disk of radius ρ centered at the origin. (The Minkowski difference $A - B$ of two sets A, B of position vectors in the Euclidean plane $\mathbb{R}^2$ is defined as $A - B := \{c \in \mathbb{R}^2 : c + B \subseteq A\}$.).

We call this set of permissible center positions a *pocket*, P. (But, again, our work is not necessarily restricted to traditional pocket machining applications.) It is well-known that (1) the boundary ∂P of P consists of $O(n)$ straight-line segments and circular arcs if the initial shape was bounded by n straight-line segments and circular arcs, and that (2) it can be obtained in $O(n \log n)$ time via Voronoi-based offsetting (Held, 1991). We use the VRONI/ArcVRONI (Held, 2001; Held & Huber, 2009) package to compute Voronoi diagrams, medial axes[1] and offsets.

Of course, in order to cover as much of P as possible, the disk will have to be moved along the boundary ∂P of P once during a finishing pass. In an actual machining application one may want to consider a Minkowski difference of the input shape and a disk of radius $\rho + \varepsilon$, for some $\varepsilon > 0$, thus pushing ∂P further inwards. This will help to avoid that the tool gets very close to the boundary of the input shape while traveling along our spiral-like path and leaves only a thin amount of material along the boundary for the finishing pass.

We assume that P is path-connected and simply-connected. If P were disconnected then we would run our algorithm separately for every connected component of P. If P contains islands—i.e., is multiply-connected—then we follow (Held & Spielberger, 2014) and convert it to a simply-connected area by introducing bridges, see Fig. 1. (Needless to say, this is a rather complicated pocket that is difficult to cover decently by only one spiral-like path.) Every bridge corresponds to two straight-line segments which have opposing orientations and which are added to the boundary of P in an appropriate way such that one single boundary contour is obtained. Human guidance in the selection of "good" bridges (relative to the intended application) is possible but, of course, the algorithm explained in Held and Spielberger (2014) can compute all bridges automatically without human interaction.

It is natural to break a spiral that winds around a point r for k times into a sequence of k individual portions, where each portion corresponds to one full turn around r. We call such a portion of a spiral a *lap*. Then the *step-over distance* at point p of lap L_{i+1} is the minimum distance from p to the next inner lap L_i, cf. Fig. 2. It is obvious that, in general, the step-over distance has to be less than the diameter of the disk which is being moved in order to avoid regions of P that are not covered. In practice, considerably smaller step-over distances are used, though. For HSM a good

[1] Since no efficient algorithm to compute the medial axis of a NURBS curve (or other freeform curve) is known, any freefrom input boundary would have to be approximated by straight-line segments and circular arcs prior to the application of our algorithm.

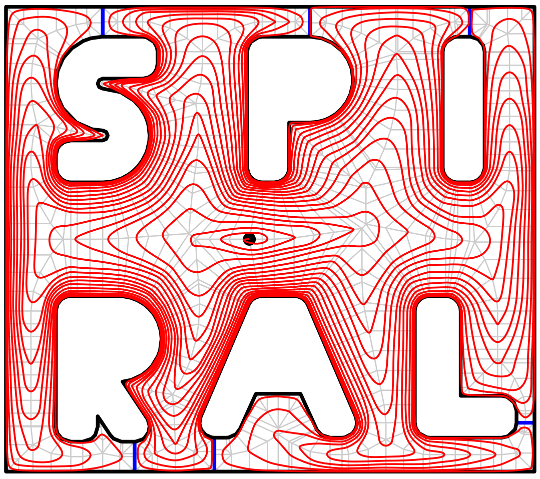


Fig. 1. A cubic B-spline as a spiral-like path inside a multiply-connected planar shape which was converted to a simply-connected shape by means of bridges (shown in blue).

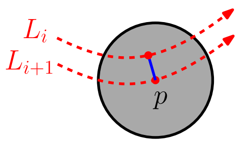


Fig. 2. The local step-over distance at a point p on lap L_{i+1} of a spiral is the minimum distance from p to the next inner lap L_i.

step-over value is a rather small fraction of the diameter that depends on the material of the cutter as well as on the workpiece. (It is largely independent of the geometry of the pocket.) E.g., for aluminum or (non-hardened) steel a typical maximum step-over is given by about 15% of the diameter. In any case, it is important that the user can control the maximum step-over Δ of a spiral path.

We note that in mathematics the term "spiral" has come to mean a curve that emanates from a center point c and winds around c at a monotonically increasing curvature and distance. Hence, every lap of a spiral lies between an inner circle and an outer circle centered at c: Every lap starts at its inner circle and reaches its outer circle after winding around c once. Thereby the fraction $\frac{d_o}{d_i}$ of the distance d_o to the outer circle over the distance d_i to the inner circle decreases monotonically.

In the sequel we investigate "spiral-like" paths that can be seen as a generalization of standard spirals. Our paths also start at a center point, r, and wind around it. And every lap of such a spiral-like path starts at an inner boundary curve and reaches its outer boundary curve after winding around r once, thereby also decreasing the fraction $\frac{d_o}{d_i}$ monotonically. However, for our spiral-like paths we allow general nested Jordan curves[2] in lieu of the concentric circles as boundary curves. Hence, the distances to r and to the inner boundary curve as well as the curvature may also decrease along a lap of such a path. Still, for the sake of terminological simplicity, we prefer to apply the term "spiral" also to our spiral-like paths in the rest of this paper.

[2] A Jordan curve is a closed curve that is simple, i.e., which has no self-intersections.

3. The medial axis tree

According to standard definition the medial axis $\mathcal{MA}(P)$ of the pocket P is the locus of all points inside P which have more than one closest point on the boundary of P, cf. Fig. 3(a). It is known to be a subset of the Voronoi diagram of P, and consist of straight-line segments and portions of conics as edges.

In order to simplify the algorithm by Held and Spielberger (2009) we approximate every edge of the medial axis $\mathcal{MA}(P)$ by a polygonal chain. The vertices of such a polygonal chain are obtained by placing uniformly distributed sample points on the edge such that the maximum length of a segment of the chain is less than a user-supplied or heuristically determined value λ. This process yields the discrete medial axis $\mathcal{MA}'(P)$. We refer to the sample points on $\mathcal{MA}(P)$ and the original nodes of $\mathcal{MA}(P)$ as *nodes* of $\mathcal{MA}'(P)$.

As usual, the *clearance*, $\mathrm{clr}(p)$, of a point p on $\mathcal{MA}'(P)$ is the radius of the largest disk ("clearance disk") centered at p that fits into P. For every node p of $\mathcal{MA}'(P)$ we consider the points $p_1, p_2, \ldots, p_k$ where the clearance disk of p touches the boundary ∂P of P, and construct the clearance line segments $\overline{pp_1}, \overline{pp_2}, \ldots, \overline{pp_k}$. If p happens to be the center of a circular arc a of ∂P then we select finitely many points on a which are uniformly spaced, with a spacing less than λ. Note that some clearance lines might share the same reflex vertex of the boundary ∂P of P as start point.

We add the set of all clearance line segments to $\mathcal{MA}'(P)$ and get the new (planar straight-line graph) $\mathcal{MA}''(P)$. The medial axis $\mathcal{MA}(P)$ is known to form a tree because P does not contain islands. This property carries over to $\mathcal{MA}''(P)$ if we regard the start points of two different clearance lines as different nodes even if they coincide at the same reflex vertex of the boundary of P. Hence, by choosing one (inner) node r of $\mathcal{MA}''(P)$ as root we can turn $\mathcal{MA}''(P)$ into a rooted tree $\mathcal{T}_r$, the *discrete medial axis tree* derived from $\mathcal{MA}''(P)$. (Since we will use this symbol for the discrete medial axis tree of P at various places and also within mathematical equations we keep the notation simple and do not make the dependence of $\mathcal{T}_r$ on P explicit in the notation.) All points that correspond to the leaves of $\mathcal{T}_r$ lie on ∂P. In particular, every start point of a clearance line on ∂P forms a leaf node of $\mathcal{T}_r$.

Since all edges of $\mathcal{T}_r$ are given by line segments, it is easy to compute the (Euclidean) length $d_{\mathcal{T}_r}(p, q)$ of the unique path along $\mathcal{T}_r$ between two nodes p, q of $\mathcal{T}_r$. This allows us to define the *Euclidean height* of a node p of $\mathcal{T}_r$ as

$$h_{\mathcal{T}_r}(p) := \max_q d_{\mathcal{T}_r}(p, q),$$

where the maximum is taken over all nodes q of the sub-tree(s) of $\mathcal{T}_r$ rooted at p.

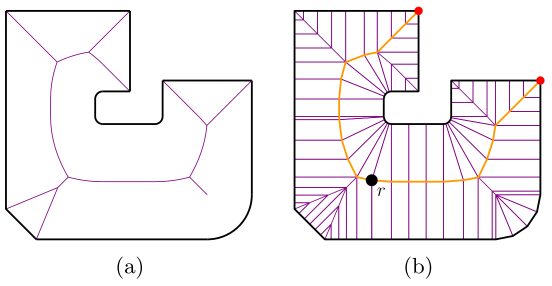


Fig. 3. (a) Medial axis of a pocket P; (b) the height-balanced discrete medial axis tree $\mathcal{T}_r$ rooted at r, with the two leaves that define the Euclidean height $h_{\mathcal{T}_r}(r)$ of r shown in red and the corresponding two radial paths shown in orange.

As in Held and Spielberger (2014) we assume that $\mathcal{T}_r$ is *height-balanced*: We assume that $h_{\mathcal{T}_r}(r)$ is defined by at least two different leaves of $\mathcal{T}_r$. That is, we assume that there exist $k \geqslant 2$ distinct leaf nodes $v_1, v_2, \ldots, v_k$ of $\mathcal{T}_r$ such that

$$h_{\mathcal{T}_r}(r) = d_{\mathcal{T}_r}(r, v_1) = d_{\mathcal{T}_r}(r, v_2) = \cdots = d_{\mathcal{T}_r}(r, v_k).$$

Every path from r to such a leaf v_i is called a *radial path* of $\mathcal{T}_r$. See Fig. 3. (For the sake of visual clarity we show this toy example with a very coarse discretization and (in subsequent figures) with an unrealistically large step-over distance.) If no such node r exists in $\mathcal{T}_r$ then we insert a new node within an edge of $\mathcal{T}_r$ in order to achieve such a perfect height balance. The computation of all Euclidean heights of the nodes of $\mathcal{T}_r$ and the height-balancing can be done easily in time linear in the number of edges of $\mathcal{T}_r$ (Held & Spielberger, 2014). In particular, no human interaction is needed for choosing the root r. In the sequel we will use r as the start point for our spiral-like paths.

Of course, the algorithms explained in the rest of our paper remain applicable if a point other than the height-balanced root r is chosen as start point. As a matter of principle, any point p in the interior of the pocket P could be chosen as start point of the spiral-like path and root r of the medial-axis tree. If p does not lie on $\mathcal{MA}''(P)$ then we consider the (closest) projection of p onto the boundary ∂P of P, and add the elongation of this projection between $\mathcal{MA}''(P)$ and ∂P as dummy Voronoi edge (Held & Spielberger, 2009). We note, though, that choosing a start point other than the height-balanced root r will result in (1) an increased length of the final spiral-like path, (2) in an increased number of laps, and (3) in a highly irregular spacing of the laps. See Fig. 4 for sample (polygonal) spirals computed by the algorithm presented in Section 5 for five different start points on $\mathcal{MA}''(P)$. Note that the same maximum step-over distance Δ was used for all five paths. We refer to Held and Spielberger (2014) for a detailed discussion of the impact of a variation of the start point.

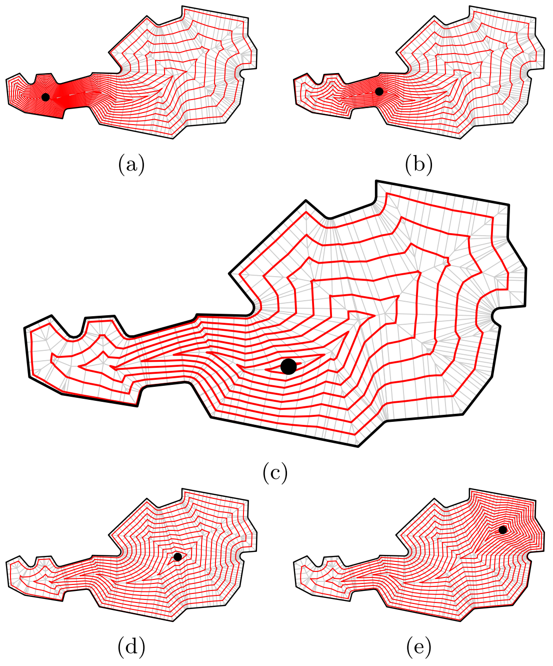


Fig. 4. Moving the start point of a spiral path away from the height-balanced root r may have a significant impact on the length of the path and on the spacing of its laps. The middle (larger) figure shows the path that starts at r.

4. Impulse propagation

Similar to Held and Spielberger (2009) we consider an impulse which is active during the time interval $[0, 1]$, which starts at the root r of the discrete medial axis tree $\mathcal{T}_r$ at time $t := 0$, and discharges concurrently at all leaves of $\mathcal{T}_r$ at time $t := 1$. Suppose that we want the impulse to travel along every radial path of $\mathcal{T}_r$ with constant velocity. Then the impulse has to cover a distance of $h_{\mathcal{T}_r}(r)$ within unit time, which implies that the velocity v of the impulse along every edge of a radial path equals $h_{\mathcal{T}_r}(r)$. Hence, a node p on a radial path of $\mathcal{T}_r$ is reached at the "start time"

$$t = \frac{h_{\mathcal{T}_r}(r) - h_{\mathcal{T}_r}(p)}{h_{\mathcal{T}_r}(r)}.$$

This simple observation can be used in a recursive manner to determine the time when the impulse reaches a specific node (or even any point within an edge of $\mathcal{T}_r$) together with the impulse velocity for all edges of $\mathcal{T}_r$. Initially, the start times for all nodes on the radial paths of $\mathcal{T}_r$ are known. (Recall that the start time t_r of the root r was set as $t_r := 0$.) Now imagine removing all edges of all radial paths from $\mathcal{T}_r$. Peeling off these "longest branches" splits $\mathcal{T}_r$ into a number of rooted sub-trees, where every sub-tree is rooted at a node of a radial path. Let p be the root of the sub-tree T_p, and let us denote its start time by t_p. We choose a leave node p' in T_p such that

$$d_{\mathcal{T}_r}(p, p') = \max_{v \in T_p} \{d_{\mathcal{T}_r}(p, v)\},$$

with ties being broken arbitrarily. That is, the path from p to p' is a longest path in T_p (and also in $\mathcal{T}_r$) from p to a leaf of T_p. Let q be the child of p on this path, and let l_e denote the length of the edge e between p and q. The length $d_{\mathcal{T}_r}(p, p')$ of the entire path from p to p' is denoted by l_b. Since the impulse has to reach p' at time $t := 1$, the (constant) velocity of the impulse along e and all other edges of the path from p to p' is given by

$$v_e = \frac{l_b}{1 - t_p} = \frac{h_{\mathcal{T}_r}(q) + l_e}{1 - t_p}.$$

We conclude that the impulse reaches q at the start time

$$t_q = t_p + \frac{l_e}{v_e}.$$

Similarly, due to the fact that the velocity of the impulse stays constant along the whole edge e, the start time t_s of a point s within (the relative interior of) e is simply given by

$$t_s = t_p + \frac{d_{\mathcal{T}_r}(p, s)}{v_e}.$$

As in the case of the nodes on the radial paths, the start times of all other nodes on the path from p to p' can be computed easily, too. Once all these start times are known we remove from T_p all edges of the path from p to p', thereby splitting T_p into a number of sub-trees. Then we apply this scheme recursively to these newly generated sub-trees.

Note that we have $v_e \leqslant v$, where v is the velocity along a radial path of $\mathcal{T}_r$. Furthermore, the equality $v_e = v$ holds only if the path from r to p' forms a radial path, too.

This recursive scheme allows us to determine all edge velocities and start times in time linear in the number of edges of $\mathcal{T}_r$. It is an easy exercise to prove that this scheme guarantees that the impulse will reach all leaves of $\mathcal{T}_r$ at time $t := 1$. Effectively, this scheme splits $\mathcal{T}_r$ into a number of branches, with constant impulse velocity per branch. See Fig. 5. We denote this set of branches by B.

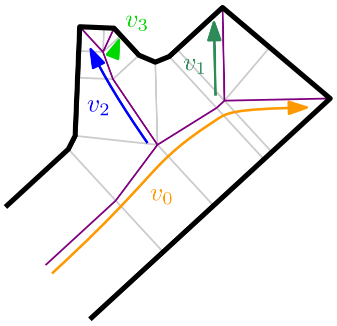


Fig. 5. The velocities on some branches of $\mathcal{T}_r$.

As the impulse flows through $\mathcal{T}_r$, it covers an increasing portion of $\mathcal{T}_r$. The point which the impulse reaches at time t on its way from r to some leaf of $\mathcal{T}_r$ is called a *vertex* at time t. Clearly, for any time $t \in [0, 1]$ there exist at most as many vertices as there are leaves in $\mathcal{T}_r$. By computing all vertices at a specific moment in time, and arranging them in the order in which they appear when $\mathcal{T}_r$ is traversed in depth-first manner, it is possible to construct a closed polygonal chain, a so-called *wavefront* $w(t)$ at time t.

The spacing of the wavefronts has to be chosen carefully in order to guarantee that the user-specified maximum step-over Δ is respected. Consider a uniform decomposition of time $(t_0, t_1, \dots, t_m)$, for some (yet unknown) $m \in \mathbb{N}$, with $0 = t_0 < t_1 < \cdots < t_m = 1$. The vertices of the wavefront $w(t_i)$ are given by the positions of the impulse at time t_i, see Fig. 6.

Let $t^* := t_{i+1} - t_i$ denote the constant time difference between the times of two neighboring wavefronts. Recall that the (symmetric) Hausdorff distance $H(X, Y)$ between two closed and bounded sets $X, Y \subset \mathbb{R}^2$ is defined as

$$H(X, Y) := \max\left\{ \max_{x \in X} \min_{y \in Y} d(x, y), \max_{y \in Y} \min_{x \in X} d(x, y) \right\},$$

where $d(x, y)$ denotes the standard Euclidean distance of two points $x, y \in \mathbb{R}^2$. Our goal is to choose t^*, such that

$$H(w(t_i), w(t_{i+1})) \leqslant \Delta \quad \text{for all} \quad i \in \{0, 1, \dots, m-1\}.$$

We recall that the impulse velocity is bound by $h_{\mathcal{T}_r}(r)$ for every edge of $\mathcal{T}_r$. This implies that the impulse travels a distance of at most $s \cdot h_{\mathcal{T}_r}(r)$ in time s along $\mathcal{T}_r$. Hence, we are able to establish an upper bound on the symmetric Hausdorff distance between $w(s_0)$ and $w(s_0 + s)$, with $s_0 \in [0, 1 - s]$, as follows:

$$H(w(s_0), w(s_0 + s)) \leqslant s \cdot h_{\mathcal{T}_r}(r).$$

This implies that the impulse travels a distance of at most Δ along $\mathcal{T}_r$ during the time s if we set

$$s := \frac{\Delta}{h_{\mathcal{T}_r}(r)}.$$

Summarizing, in order to ensure $H(w(t_i), w(t_{i+1})) \leqslant \Delta$ for all $i \in \{0, 1, \dots, m-1\}$, it suffices to set m as

$$m := \left\lceil \frac{1}{s} \right\rceil.$$

This gives

$$t^* := \frac{1}{m}$$

as the constant time distance between two impulse times that correspond to neighboring wavefronts. Note that this construction implies that the radial paths are split by the wavefronts into sections with length at most Δ.

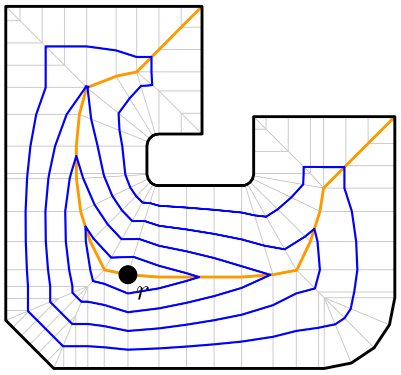


Fig. 6. A series of uniformly spaced wavefronts inside the pocket for $m := 5$. The wavefront $w(t_0)$ equals r, and $w(t_5)$ coincides with the boundary of P; both are not shown. The two radial paths in $\mathcal{T}_r$ between r and leaves of $\mathcal{T}_r$ are shown in orange.

5. Generating one spiral

We now focus on the generation of the actual spiral path, which is fundamentally different to the strategy applied by Held and Spielberger (2009). We explain and depict counter-clockwise (CCW) spiral paths; the modifications needed to obtain clockwise (CW) spirals are trivial. A spiral path $\mathcal{S}(P, \Delta)$ is made up of m laps $L_1, L_2, \dots, L_m$. In addition, we have $L_0 := \{r\}$ and $L_{m+1} := \partial P$ as two "trivial" laps. Each of the laps is a polygonal chain whose vertices lie on $\mathcal{T}_r$. In a nutshell, we compute the innermost (non-trivial) lap L_1 by interpolating between the wavefronts $w(t_0)$, i.e., the root r of $\mathcal{T}_r$, and $w(t_1)$. Similarly, L_m is formed by an interpolation between $w(t_{m-1})$ and $w(t_m)$, i.e., ∂P. See Fig. 7. All other (non-trivial) laps are formed by interpolations between L_1 and L_m. Every lap starts and ends at one specific clearance line incident at r. The important technical issue is to generate these laps in such a way that the step-over distance between neighboring laps does not exceed the user-specified maximum step-over Δ.

We start with explaining how L_1 is generated, see Fig. 8. Recall that $w(t_0)$ degenerates to r. Suppose that q_0 is the vertex of $w(t_1)$ that is intersected by the clearance line $\overline{rv_0}$, on which all laps start and end. Thus, L_1 starts at r and ends at q_0. We number the vertices of $w(t_1)$ in CCW order, starting at q_0. Now consider some vertex of $w(t_1)$, e.g., q_4 in Fig. 8. Let d denote the circumference of $w(t_1)$, let d_4 denote the length of the polygonal chain $q_0 q_1 \dots q_4$, and let $\delta_4 := d_{\mathcal{T}_r}(r, q_4)$, i.e., the distance from r to q_4 along $\mathcal{T}_r$. Then a candidate corner c of L_1 is placed on the path from q_4 to r at a distance (along $\mathcal{T}_r$) of

$$\left(1 - \frac{d_4}{d}\right) \cdot \delta_4$$

from q_4. We store c at the corresponding edge of $\mathcal{T}_r$. Note that some vertices of $w(t_1)$ might end up storing candidate corners on the same edge or path towards r. These candidate corners are classified as "type-1" candidate corners.

After setting the weight d to the circumference of ∂P and letting the vertices of $w(t_{m-1})$ play the role of r, we obtain type-1 candidate corners for L_m in a similar way by moving from the vertices of $w(t_m)$, i.e., ∂P, towards vertices of $w(t_{m-1})$. If required, we can also let L_m wind around r a bit more than once, and let it end at some point on ∂P other than v_0, by making d larger than the circumference of ∂P.

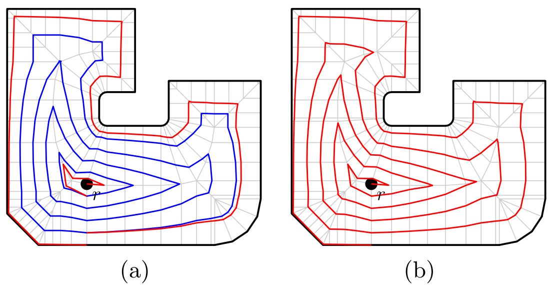


Fig. 7. (a) The first and the last lap are derived by interpolating neighboring wavefronts. (b) The final spiral path that starts at r and ends on ∂P.

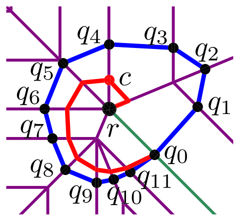


Fig. 8. The first lap starts at the root r of $\mathcal{T}_r$ and ends at a vertex q_0 of $w(t_1)$ on a clearance line (shown in green), on which all laps start and end.

In order to actually generate L_1 we scan $\mathcal{T}_r$ in a depth-first order, starting at r and moving along $\overline{rv_0}$ as first branch of $\mathcal{T}_r$. The recursive scan stops whenever a candidate corner for L_1 is encountered. This depth-first scan establishes all vertices of L_1 in the desired (CCW) order.

Now we start a new depth-first scan towards the leaves of $\mathcal{T}_r$ at every vertex q of L_1. The recursion of the depth-first scan is stopped whenever we get to a distance $(m-1)\cdot\Delta$ from q along $\mathcal{T}_r$ or, trivially, if we reach the boundary ∂P. At every such stopping point of the recursion a new "type-2" candidate corner for L_m is placed. Then another depth-first scan starting at r reveals all vertices of L_m by stopping the recursion whenever a candidate corner for L_m (of either type-1 or type-2) is encountered.

Our construction implies the following two distance properties:

$$H(L_0, L_1) \leqslant \Delta \quad \text{and} \quad H(L_1, L_m) \leqslant (m-1)\cdot\Delta.$$

We now argue that L_m is guaranteed to be contained in the annulus defined by $w(t_{m-1})$ and $w(t_m)$: Every type-1 candidate corner for L_m lies in this annulus since it is generated by an interpolation between $w(t_{m-1})$ and $w(t_m)$. Every type-2 candidate corner which does not lie on ∂P is at a distance of $(m-1)\cdot\Delta$ along $\mathcal{T}_r$ from a vertex of L_1 and, thus, at a distance of at least $(m-1)\cdot\Delta$ from r. However, all vertices of $w(t_{m-1})$ are at a distance of at most $(m-1)\cdot\Delta$ from r. Thus, also every type-2 candidate corner lies within the annulus defined by $w(t_{m-1})$ and $w(t_m)$. As a result, L_m lies also in this annulus. (More precisely, all of L_m lies within the interior of this annulus except for the start point and end point of L_m.) In particular, we get

$$H(L_m, \partial P) = H(L_m, L_{m+1}) \leqslant \Delta$$

as the third distance property.

The remaining laps $L_2, \ldots, L_{m-1}$ can be generated similar to the generation of the initial wavefronts if we take the freedom to regard one lap as a special type of wavefront between L_1 and L_m: Again we let an impulse propagate along $\mathcal{T}_r$. However, this modified impulse propagation starts at time $t := 0$ at the vertices of L_1, and ends at time $t := 1$ at the vertices of L_m. Then, for properly chosen velocities of the impulse on the edges of $\mathcal{T}_r$, the "wavefront" that corresponds to the time $i/m-2$ forms the lap L_{i+1}, for $i \in \{1, 2, \ldots, m-2\}$.

By connecting all non-trivial laps $L_1, L_2, \ldots, L_m$ in the natural way we obtain a polygonal path $\mathcal{S}(P,\Delta)$ inside P. Trivially, $\mathcal{S}(P,\Delta)$ starts at r and ends on ∂P. Furthermore, $\mathcal{S}(P,\Delta)$ is not self-intersecting because we move outwards in a strictly monotonic fashion, starting at r, until we arrive at ∂P. And due to the construction, $\mathcal{S}(P,\Delta)$ respects the maximum step-over Δ: The $m-2$ laps $L_2, \ldots, L_{m-1}$ split a distance (along $\mathcal{T}_r$) of at most $(m-1)\cdot\Delta$ into $m-1$ portions of length at most Δ. Hence, the Hausdorff distance between L_i and L_{i+1} is at most Δ for all $i \in \{1, 2, \ldots, m-1\}$.

We summarize our result as follows:

$$H(L_i, L_{i+1}) \leqslant \Delta \quad \text{for all} \quad i \in \{0, 1, \ldots, m\},$$

which settles the claim that our spiral path $\mathcal{S}(P,\Delta)$ obeys the user-specified maximum step-over Δ. We note that Δ forms an upper bound on the true maximal step-over distance: We do not determine the actual Hausdorff distance but only measure distance along (possibly curved) edges of the medial axis of P. (An algorithm by Alt, Behrends, & Blömer (1995) would allow to compute one Hausdorff distance between polygonal curves with a total of n vertices in $O(n \log n)$ time but there is no obvious way for applying this algorithm to the laps of our spiral path under generation.)

6. Improving and smoothing a spiral

6.1. Impulse modification

Recall that the impulse moves with constant velocity per branch of B, cf. Fig. 5. In particular, it is constant within every edge of $\mathcal{T}_r$. Hence, the velocity of the impulse might change rapidly at some nodes of $\mathcal{T}_r$. This leads to exceedingly sharp corners along the spiral path. We now explain how to remedy this problem by modifying the impulse propagation.

In order to mitigate the effects of rapidly changing velocities whenever a shorter branch starts, we part from the simple scheme of using constant velocities and assign a linear velocity function to every element of B. As in Section 4, the dynamic velocity of r is set to $h_{\mathcal{T}_r}(r)$ and its start time t_r is set to 0. The branches in B are, again, considered in the order in which they appear when $\mathcal{T}_r$ is traversed in depth-first manner. Let b be the branch that is currently inspected, with p as its start node, p' as its end (leaf) node, and l_b as its length. According to Section 4, the constant "average" impulse velocity assigned to all edges of b is given by

$$v_{\text{avg}} = \frac{l_b}{1-t_p},$$

where t_p denotes the start time at p. Roughly, the new idea is to start with an initial velocity along b that (ideally) is identical to the velocity v_p with which the impulse reached p, and to decrease this velocity linearly as one gets closer to ∂P. Of course, even after this modification the impulse will have to travel a distance of l_b within time $1-t_p$.

We define the start velocity along b as

$$v_{\text{start}} := \begin{cases} v_p & \text{if } 2v_{\text{avg}} \geqslant v_p, \\ 2v_{\text{avg}} & \text{else}. \end{cases}$$

Furthermore, the end velocity v_{end} along b is defined as

$$v_{\text{end}} := \begin{cases} 2v_{\text{avg}} - v_p & \text{if } 2v_{\text{avg}} \geqslant v_p, \\ 0 & \text{else}. \end{cases}$$

The corresponding linear velocity function ϑ^b for the velocity along b is given by

$$\vartheta^b(s) := v_{\text{start}} - (v_{\text{start}} - v_{\text{end}})s,$$

with $s \in [0, 1]$. Obviously, the velocity along b at a specific time t, with $t_p < t \leqslant 1$, is given by

$$\vartheta^b\left(\frac{t - t_p}{1 - t_p}\right).$$

Finally, at time t the impulse has travelled a distance of

$$\frac{v_{\text{start}} + v_q}{2}\,(t - t_p)$$

along b. We note that the distance travelled by the impulse equals l_b for $t := 1$, for both cases in the settings of v_{start} and v_{end}.

We can now use this modified linear impulse velocity and apply the schemes discussed in Sections 4 and 5 to compute the wavefronts as well as the spiral path, see Fig. 9. We note that the modified impulse travels with the (standard) constant velocity $v = h_{\mathcal{T}_r}(r)$ along all radial paths of $\mathcal{T}_r$. Along all other branches the velocity varies but never exceeds v. This fact implies that the distance analysis of Section 5 is still applicable and that the maximum step-over Δ is respected everywhere along the final spiral path even for the modified impulse setting.

In order to reduce directional discontinuities even further we keep in mind that a vertex v of lap L_i of the spiral path could be moved along $\mathcal{T}_r$ towards ∂P as long as this movement does not (1) result in a violation of the maximum step-over Δ or (2) cause v to run over L_{i+1}. One could even require that v keeps a certain minimum distance from L_{i+1} in order to avoid that laps get extremely close to each other. In any case, every vertex v has a range of positions which are permissible for an outwards shift of v. (This range can also be empty for some particular vertex.)

Let (v_1, v_2, v_3) be a triple of consecutive vertices of the spiral. We say that the angle at v_2 is convex if v_2 lies to the left of the ray from v_1 to v_3, reflex if it lies to the right of this ray, and tangential otherwise. We compute the deviation of the angle at v_2 from 180°, and insert its absolute value into a priority queue PQ. We also keep a link from v_2 into the position of this value in PQ, and from it back to v_2. This is done for all vertices of the spiral path. The priority queue PQ is organized such that it maintains the maximum angular deviation at its top.

Once PQ has been filled we are ready to shift some vertices. Let v_2 be the vertex that is linked to the angular deviation currently fetched from PQ. If the angle at v_2 is convex then we shift v_2 outwards. If it is reflex then we shift its predecessor v_1 and its successor v_3 outwards. Of course, the shifting of one or two vertices of the triple (v_1, v_2, v_3) shall not result in deviations of the angle(s) from 180° at the unshifted vertices which are greater than the one which we try to reduce at v_2. In theory, the optimum amount(s) for shifting could be determined by solving a (non-linear) optimization problem. We resort to a much simpler approach and sample 10 uniformly distributed positions within the maximum permissible range of new positions. (The sample number 10 turned out to be good enough for our purposes; there is no theoretical justification for it.) If the optimal shift determined this way does indeed reduce the maximum absolute deviation of the angles at v_1, v_2 and v_3 from 180° then we delete the three entries for v_1, v_2, v_3 from PQ and insert the three absolute values of the new deviations from 180° at v_1, v_2 and v_3 into PQ. Otherwise, the entry for v_2 is deleted from PQ but no shift is carried out. See Fig. 10(a) for a result of this shifting strategy applied to the setting of Fig. 9(b). Additional sample paths are shown in Fig. 11; the polygonal path derived from a constant impulse propagation for the sample pocket of Fig. 11(b) is shown in Fig. 4.

A minor technical problem is given by the fact that shifting a vertex towards ∂P might cause it to run over a node of $\mathcal{T}_r$. In such a case we have to split the vertex into several individual copies that move independently towards ∂P.

6.2. Higher-order smoothing

For now we have obtained a spiral path which is described by a polygonal chain. Practical experiments made it apparent quickly

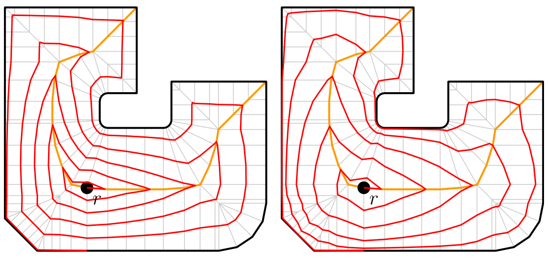


Fig. 9. (a) Spiral path according to piecewise constant velocities of the impulse, cf. Section 5. (b) Spiral path according to the modified linear velocities of the impulse.

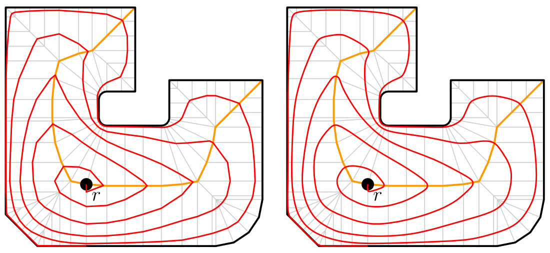


Fig. 10. (a) Spiral path after some vertices were shifted outwards; (b) approximation of this path by a cubic B-spline.

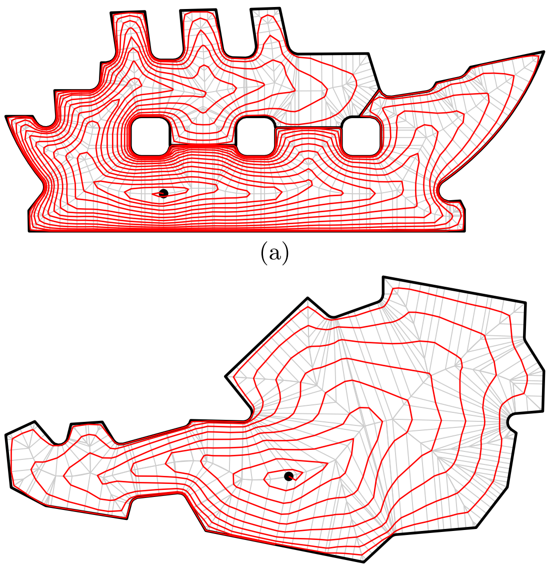


Fig. 11. Sample polygonal spiral paths generated based on the modified impulse propagation.

that there is nothing to gain by employing non-linear functions for the impulse velocity: The higher the algebraic degree of the velocity function, the more "tricky" freedom for choosing "good" parameters and the more work to implement such a function.

Experiments made it also apparent that resorting to a very fine sampling of the medial axis and, thus, to a large amount of clearance lines does not help to make the spirals look smoother. Rather, the finer the sampling, the more the resulting spirals seemed to "converge" to some limit curve. This can be understood if one analyzes the mathematics of the impulse propagation in the neighborhood of a sharp corner of a spiral: For parallel clearance lines the propagation of the impulse obeys the intercept theorem and, thus, the wavefront locally follows a straight-line segment even if the sampling rate is increased significantly.

As a rule of thumb, using up to five times as many clearance lines as we used in our sample Fig. 10 seems to yield decent results. The sampling can be coarser along straight-line edges of the medial axis of P and should be finer along conic edges.

In any case, a purely polygonal path will always show directional discontinuities at its corners, no matter how much effort were invested in an improved impulse propagation. Hence, it seems natural to resort to an approximation of our polygonal spirals by higher-order primitives if the smoothness of the path is of a concern.

Of course, an approximation of a spiral path should still have Δ as maximum step-over distance, and it must not leave the pocket P. These two requirements place constraints on an approximation. Suppose that our spiral path $\mathcal{S}(P, \Delta_1)$ has a maximum step-over distance of Δ_1. If we can guarantee $H(\mathcal{S}(P, \Delta_1), \mathcal{A}) \leqslant \Delta_2$ for its approximation $\mathcal{A}$ then we know that $\mathcal{A}$ has a maximum step-over distance of $\Delta_1 + \Delta_2$. Hence, we can proceed as follows: (1) We choose an approximation threshold ε with $0 < \varepsilon < \Delta$, (2) we compute $\mathcal{SP} := \mathcal{S}(P, \Delta - \varepsilon)$, and (3) we compute an approximation $\mathcal{A}$ of $\mathcal{SP}$ such that $H(\mathcal{SP}, \mathcal{A}) \leqslant \varepsilon$. This approach ensures that the maximum step-over distance Δ is not exceeded by $\mathcal{A}$. In order to guarantee that $\mathcal{A}$ does not leave P it suffices to ensure that the approximation of the last lap stays locally on the left side of that lap. All other laps can be approximated using a symmetric tolerance.

For this work we used the PowerApx-package (Heimlich & Held, 2008; Held & Kaaser, 2014). Amongst other things, it supports the approximation of polygonal chains by biarcs and cubic B-splines, thus achieving G^1 continuity or even C^2 continuity. The approximation curve $\mathcal{A}$ is guaranteed to lie within a user-specified tolerance of the original input $\mathcal{SP}$, and $\mathcal{SP}$ is guaranteed to lie within a user-specified tolerance of $\mathcal{A}$. Hence, a bound on the Hausdorff distance between $\mathcal{A}$ and $\mathcal{SP}$ can be established. These tolerances can be either symmetric, asymmetric, or even one-sided. (A one-sided tolerance is used for the last lap of $\mathcal{SP}$.)

In Fig. 10(b) we see the approximation of the spiral path shown in Fig. 10(a) by a cubic B-spline. For the sake of simplicity, we subjected the actual spiral of Fig. 10(a) to the approximation, without

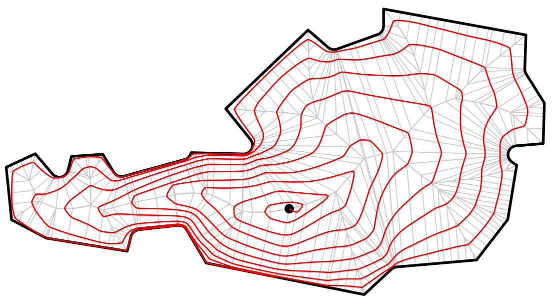


Fig. 12. Cubic B-spline approximation of the polygonal spiral path of Fig. 11(b).

reducing the maximum step-over distance Δ. Hence, although we used a tiny approximation threshold ε which, if plotted, would hardly exceed the pen width used for drawing ∂P, the step-over distance of the resulting cubic B-spline might exceed Δ ever so slightly. Other sample cubic B-Spline spirals are shown in Figs. 1 and 12.

7. Double and composite spirals

7.1. Double spiral

All spirals discussed so far have one fact in common: They start at some point of the medial axis and end at the boundary ∂P of the pocket P. We now generalize our approach to a double spiral that starts and ends at the boundary ∂P.

As in the case of a single spiral, the user-specified step-over Δ implies a certain minimum number of wavefronts. For the sake of descriptional simplicity, suppose that this number is odd and that we have $2k+1$ wavefronts $w(t_0), w(t_1), \ldots, w(t_{2k})$, with $w(t_0)$ equal to r and $w(t_{2k})$ equal to ∂P. We use the algorithm of Section 5 to compute one single "inner" spiral with maximum step-over 2Δ which starts at r and ends at v_0 on ∂P. Let $L_1, L_3, \ldots, L_{2k-1}$ denote the successive laps of this spiral. Hence, L_1 starts at r and ends at the intersection q of $w(t_2)$ with $\overline{rv_0}, L_3$ starts at q and ends on $w(t_4)$, and so on. In particular, L_{2k-1} ends at v_0 on ∂P.

Let L_{2k+1} be identical to ∂P. For $i \in \{1, 3, \ldots, 2k-1\}$, we plant an impulse at every vertex of lap L_i that moves along $\mathcal{T}_r$ towards the leaves of $\mathcal{T}_r$, starting on L_i at time $t := 0$ and reaching L_{i+2} at time $t := 1$. Stopping the impulse at time $t = 1/2$ yields the vertices of the laps $L_2, L_4, \ldots, L_{2k}$ of the "outer" spiral, where L_2 starts at q and L_{2k} ends at v_0 on ∂P. As for a single spiral, the positions of the end-points of L_{2k-1} and L_{2k} on ∂P can be adjusted to meet specific needs. In Fig. 13(a), the outer spiral and the vertices of the inner spiral are shown.

In order to connect the start of L_2 at q with the start of L_1 at r we move from the vertices of L_1 towards r along $\mathcal{T}_r$ for a distance of Δ, thus obtaining candidate corners of a polygonal path that connects L_1 and L_2. (This is similar to the generation of L_1 in Section 5.) We note that this construction ensures that the resulting double spiral is not self-intersecting and respects the maximum step-over Δ. In Fig. 13(b), a full double spiral is shown for our sample pocket.

Of course, the smoothing operations of Section 6.1 are applicable again. Fig. 14 shows the outer polygonal spiral computed according to the modified impulse propagation and smoothing, and Fig. 14 shows an approximation of the full double spiral by a cubic B-spline. The outer spiral was stopped in the upper-left corner of ∂P. (Again, we used the powerapx package (Heimlich & Held, 2008; Held & Kaaser, 2014) to obtain this approximation.)

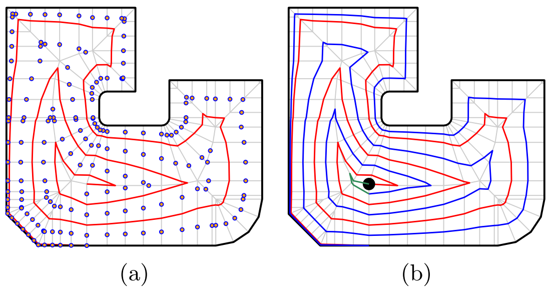


Fig. 13. (a) The vertices of the outer spiral (highlighted by blue circles) are placed halfway between the corresponding vertices of the inner spiral. (b) Final double spiral consisting of the inner spiral (red), outer spiral (blue) and connecting polygonal path (green).

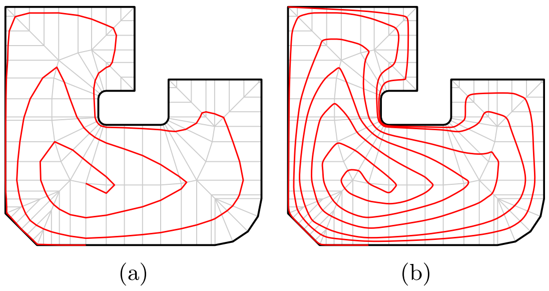


Fig. 14. (a) Outer polygonal spiral generated based on the modified impulse propagation. (b) Resulting double spiral as a cubic spline.

7.2. Composite spiral path

As suggested in Held and Spielberger (2014), we can decompose a complex (possibly multiply-connected) pocket into simpler sub-pockets and then compute spiral paths within these sub-shapes. The obvious disadvantage of having multiple spirals is the need to link them into one path. In general, this will require the application to pause during these linking portions of the path, like during retraction moves in machining.

We now employ our machinery for computing single and double spirals to obtain composite spiral path. In a nutshell, we compute suitable spirals within every sub-pocket and splice them together appropriately.

Let $\mathcal{D}$ be a set of sub-pockets obtained by decomposing the pocket P by some means. (See, e.g., Held & Spielberger, 2014 for methods to achieve a decent decomposition.) The common boundary between two sub-pockets is called a decomposition edge. We derive a graph G from $\mathcal{D}$ in the following way: The nodes of G represent the sub-pockets of $\mathcal{D}$. Two nodes are linked by an edge of G if the corresponding sub-pockets share a decomposition edge. For the sake of descriptional simplicity we assume that G is a tree. (Recall that we can use bridge edges to convert a multiply-connected shape into a simply-connected shape.) A sample pocket together with its decomposition and resulting graph G are shown in Fig. 15(a).

We start with computing two leaf nodes v_1, v_2 of G which determine the diameter of G. That is, no path in G between any pair of nodes of G contains more edges than the path between v_1 and v_2. The sub-pockets that correspond to v_1 and v_2 are the only ones in which a single spiral is computed, cf. Fig. 15(b). In every other sub-pocket we generate a double spiral. Now recall that we can let our spirals end at arbitrary points on the pocket boundary. In particular, we can make them start and end on the decomposition edges. This makes it easy to link all spirals within the sub-pockets that correspond to the diameter path between v_1 and v_2 into one composite spiral path.

In a similar way, the other spirals can be linked to paths and spliced into the composite spiral path obtained so far. We do not go into details of the linking since the actual geometry of the linking portions of the spirals depends on the geometry of the decomposition edges. (For the sake of simplicity, in our own work we use straight-line segments as decomposition edges.) See Fig. 15(c) for a full composite spiral path.

8. Discussion and conclusion

We introduce a simple and easy-to-implement algorithm for computing polygonal spirals to cover planar shapes bounded by straight-line segments and circular arcs. The paths do not self-intersect and respect a user-specified maximum step-over distance. Smoothing heuristics help to prevent excessively sharp corners, thus avoiding a drastic variation of the curvature. If our paths are applied in an HSM application then smoothing will also help to avoid a rapid change of the engagement angle. And, indeed, at least our single spirals have already mastered a practical test at the shop-floor level. See Fig. 16 for two pockets machined by our industrial partner using flat-end milling.

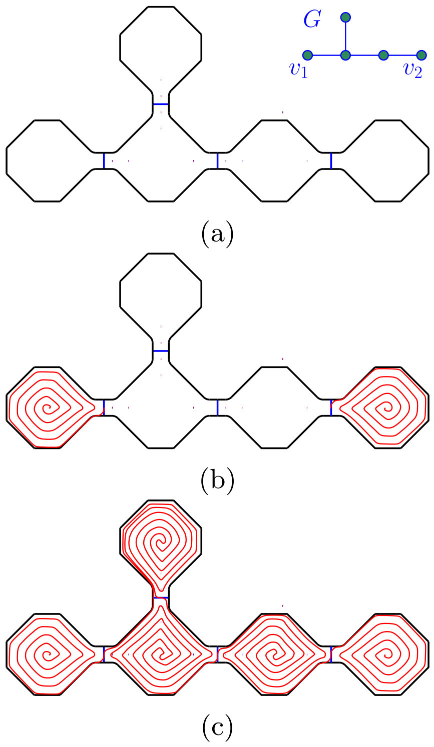


Fig. 15. (a) Subdivision into five sub-pockets and resulting graph G (in the top right corner); (b) first and last single spiral; (c) cubic B-spline as full composite spiral path.

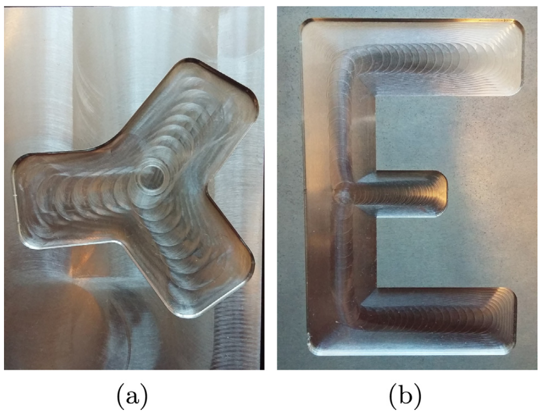


Fig. 16. Two parts machined in aluminum.

Currently we use PowerApx to approximate a polygonal spiral by biarcs or cubic B-splines. While using a package like PowerApx is certainly the simplest approach to boost a polygonal spiral to higher continuity, it is not necessarily the best approach: PowerApx is a general-purpose tool which "blindly" approximates a polygonal path such that specific tolerances are met. As discussed, this allows to obtain smooth spirals that still respect a user's maximum step-over distance Δ. However, it cannot take advantage of the fact that some portions of our spirals would allow a much coarser approximation since we are still far from exceeding Δ. Trying to exploit this additional information for a better approximation that either has fewer approximation primitives or a lower variation of the curvature seems to be a promising avenue for future research.

Conflict of interest

None.

Acknowledgements

We would like to thank the Austrian Science Fund (FWF, Grant P25816-N15) as well as EMCO GmbH (Hallein, Austria) for supporting this research.

References

Abrahamsen, M. (2015). Spiral toolpaths for high-speed machining of 2D pockets with or without islands. In: *Proceedings of the ASME IDETC/CIE 2015 conference* (pp. V02BT03A017–V02BT03A017).

Alt, H., Behrends, B., & Blömer, J. (1995). Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence, 13*(3), 251–265. https://doi.org/10.1007/BF01530830.

Banerjee, A., Feng, H.-Y., & Bordatchev, E. V. (2012). Process planning for floor machining of 21/2D pockets based on a morphed spiral tool path pattern. *Computers & Industrial Engineering, 63*(4), 971–979.

Bieterman, M. B., & Sandstrom, D. R. (2002). A curvilinear tool-path method for pocket machining. In: *Proceedings of IMECE2002, ASME international mechanical engineering congress* (pp. 149–158).

Chandler, P., Rasmussen, S., & Pachter, M. (2010). UAV cooperative path planning. In: *AIAA guidance, navigation and control conference*.

Heimlich, M., & Held, M. (2008). Biarc approximation, simplification and smoothing of polygonal curves by means of Voronoi-based tolerance bands. *International Journal of Computational Geometry & Applications, 18*(03), 221–250.

Held, M. (1991). *On the computational geometry of pocket machining. Lecture notes in computer science* (Vol. 500). Springer-Verlag. ISBN 3-540-54103-9.

Held, M. (2001). VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Computational Geometry: Theory and Applications, 18*(2), 95–123.

Held, M., & Huber, S. (2009). Topology-oriented incremental computation of Voronoi diagrams of circular arcs and straight-line segments. *Computer-Aided Design, 41*(5), 327–338.

Held, M., & Kaaser, D. (2014). C2 approximation of planar curvilinear profiles by cubic B-splines. *Computer-Aided Design and Applications, 11*(2), 206–219.

Held, M., & Spielberger, C. (2009). A smooth spiral tool path for high speed machining of 2D pockets. *Computer-Aided Design, 41*(7), 539–550.

Held, M., & Spielberger, C. (2014). Improved spiral high-speed machining of multiply-connected pockets. *Computer-Aided Design and Applications, 11*(3), 346–357.

Keller, J. F. (2017). *Path planning for persistent surveillance applications using fixed-wing unmanned aerial vehicles*. PhD dissertation available from ProQuest. AAI10247340. URL <http://repository.upenn.edu/dissertations/AAI10247340>.

Pateloup, V., Duc, E., & Ray, P. (2004). Corner optimization for pocket machining. *International Journal of Machine Tools and Manufacture, 44*(12), 1343–1353.

Romero-Carrillo, P., Torres-Jimenez, E., Dorado, R., & Díaz-Garrido, F. (2015). Analytic construction and analysis of spiral pocketing via linear morphing. *Computer-Aided Design, 69*, 1–10.

Zhao, Z., Liu, B., Zhang, M., Zhou, H., & Yu, S. (2009). Toolpath optimization for high speed milling of pockets. In: *Second international conference on information and computing science* (Vol. 1, pp. 327–330).

Zhao, H., Gu, F., Huang, Q.-X., Garcia, J., Chen, Y., Tu, C., Benes, B., Zhang, H., Cohen-Or, D., Chen, B., et al. (2016). Connected fermat spirals for layered fabrication. *ACM Transactions on Graphics, 35*(4), 100.

Zhao, Z., Wang, C., Zhou, H., & Qin, Z. (2007). Pocketing toolpath optimization for sharp corners. *Journal of Materials Processing Technology, 192*, 175–180.

Zhou, B., Zhao, J., Li, L., & Xia, R. (2016). Double spiral tool-path generation and linking method for complex pocket machining. *Machining Science and Technology, 20*(2), 262–289.

Computing Low-Cost Convex Partitions for Planar Point Sets Based on Tailored Decompositions

Günther Eder, Martin Held, Stefan de Lorenzo, and Peter Palfrader [Ede+20]

Published in:
Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020)

June 2020

[Ede+20]

Günther Eder, Martin Held, Stefan de Lorenzo, and Peter Palfrader. "Computing Low-Cost Convex Partitions for Planar Point Sets Based on Tailored Decompositions". In: *Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020)*. Vol. 164. Leibniz International Proceedings in Informatics (LIPIcs). Zürich, Switzerland: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, June 2020, 85:1–85:10. ISBN: 978-3-95977-143-6. DOI: 10.4230/LIPIcs.SoCG.2020.85

Computing Low-Cost Convex Partitions for Planar Point Sets Based on Tailored Decompositions

Günther Eder
Universität Salzburg, FB Computerwissenschaften, Austria
geder@cs.sbg.ac.at

Martin Held
Universität Salzburg, FB Computerwissenschaften, Austria
held@cs.sbg.ac.at

Stefan de Lorenzo
Universität Salzburg, FB Computerwissenschaften, Austria
slorenzo@cs.sbg.ac.at

Peter Palfrader
Universität Salzburg, FB Computerwissenschaften, Austria
palfrader@cs.sbg.ac.at

Abstract

Our work on minimum convex decompositions is based on two key components: (1) different strategies for computing initial decompositions, partly adapted to the characteristics of the input data, and (2) local optimizations for reducing the number of convex faces of a decomposition. We discuss our main heuristics and show how they helped to reduce the face count.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational Geometry, geometric optimization, algorithm engineering, convex decomposition

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.85

Category CG Challenge

Supplementary Material The source code of our tools and heuristics is available at GitHub and can be used freely under the GPL(v3) license: `https://github.com/cgalab`.

Funding Work supported by Austrian Science Fund (FWF): Grants ORD 53-VO and P31013-N31.

1 Introduction

The task of the 2020 Computational Geometry Challenge – called Challenge in the sequel for the sake of brevity – was to compute minimum convex decompositions (MCD) of point sets in the plane. We refer to the survey by Demaine et al. [2] for background information.

We employed several tools and heuristics to tackle the Challenge. All tools submitted their solutions to a central database of ours, such that tool A could query and then improve on solutions obtained by tool B, and vice versa. Most of our heuristics are based on local search: Begin with a convex decomposition and iteratively modify it locally to reduce the number of convex faces. The source code of our tools and heuristics is available at GitHub and can be used freely under the GPL(v3) license: `https://github.com/cgalab`.

2 Algorithmic methods

2.1 3-Approximation

Our tool 3Apx implements the 3-approximation algorithm by Knauer and Spillner [3]. Tests quickly showed that this approach generates decompositions that are clearly not optimal. Hence, we extended 3Apx by an approach based on onion layers [1]: We construct all onion layers and then find convex decompositions of the annuli between the layers. Contrary to [3], this approach does not modify the layer boundaries. See Figure 1 for sample decompositions obtained via 3-approximation and the onion layers. Experiments showed that computing a convex decomposition based on onion layers is superior to the 3-approximation algorithm even without merging convex faces across different onion layers, see the plot in Figure 7.

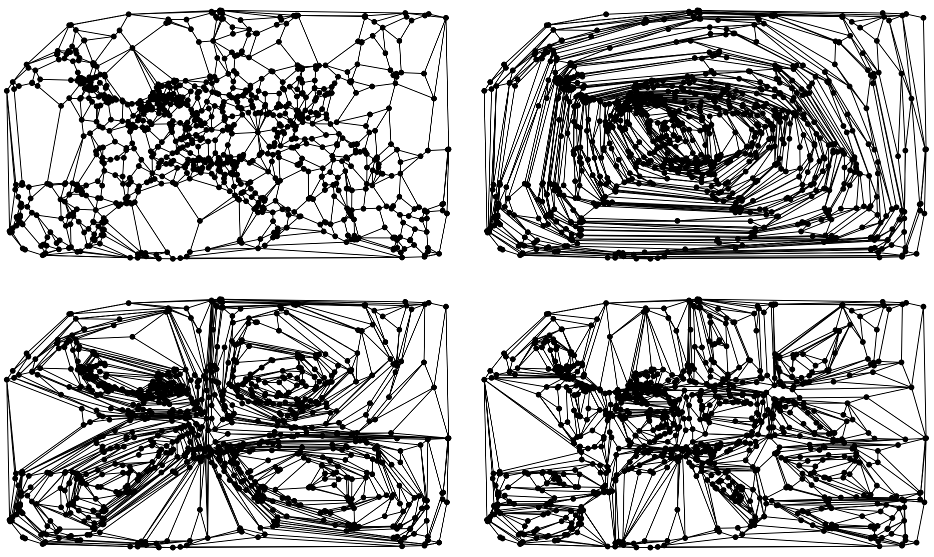


Figure 1 In reading order: When using the 3-approximation implemented in 3Apx for an instance with 1000 vertices we obtain a convex decomposition with 1350 faces; 1125 faces when using our approach based on onion layers without partitioning into cells; 1123 faces when partitioning into four cells and subsequent onion-layer based decomposition; and 1148 faces when using 16 cells.

A visual inspections of the results achieved by 3Apx quickly made it apparent that the decompositions computed contained lots of extremely long and thin triangles. Hence, we tried to partition a given input P into smaller "cells" and then run 3Apx on each cell individually. Then the individual decompositions are joined by triangulating the area between them and randomly dropping triangulation edges if this is possible without violating convexity. This produced visually nicer images such as the last two decompositions in Figure 1 but did not reduce the face counts substantially.

2.2 Merging faces

One of our earliest ideas was to do the obvious: Start with a triangulation of P and then merge adjacent faces by randomly dropping triangulation edges as long as faces remain convex. Tests with an initial straightforward implementation, MergeRefine, suggested that this is a promising approach, easily beating 3Apx (Figure 7). In order to be better prepared for refined heuristics we quickly re-implemented it in a new tool called Recursor. In particular, we resorted to a more advanced data structure for storing our decompositions.

Recursor keeps its state in a variant of a doubly-connected edge list (dcel) or half-edge data structure. The base layer of our variant is a dcel of a triangulation of P. Additionally, each half-edge pair is considered either *constrained* or not constrained, depending on whether the edge is part of our convex decomposition of P. As a layer on top of the base dcel, each constrained half-edge, in addition to the pointers to the next triangulation edges encountered in clockwise (CW) or counter-clockwise (CCW) direction, also holds a reference to next CW and CCW constrained edges; cf. Figure 2. This enables constant-time testing whether an edge can be dropped, i.e., marked unconstrained, while keeping a fully fledged triangulation of P during the entire process. To obtain a decomposition, Recursor first uses Shewchuk's Triangle [4] to construct a Delaunay triangulation of P, and then iterates over the edges in a random order, dropping every edge that can be dropped. This process continues until no further edge can be dropped, i.e., the decomposition is locally optimal.

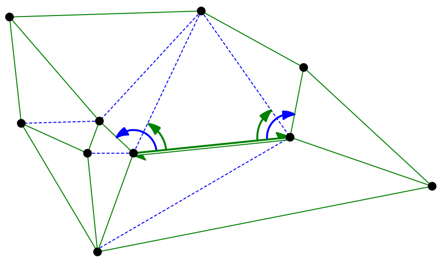


Figure 2 Our two-layer doubly-connected edge list stores two planar graphs simultaneously, with one planar graph being a subgraph of the other. A constrained half-edge h has references to its neighbors in the convex decomposition (green) and to the underlying triangulation (blue). The Challenge data set `euro-night-10` is shown.

Hole refinement. It is not surprising that a locally optimal decomposition may consist of many more faces than the true global optimum. Therefore, we worked on strategies that allow us to move away from local optima: Recursor picks a face f of the decomposition and a (random) number of its neighbors as a "hole" to work on. In general, it picks a face f that is incident to a high-degree vertex. We consider a vertex of the decomposition to be of high degree if its degree is larger than 3 or if its degree is equal to 3 and two incident edges span an angle of 180°. In other words, high-degree vertices are vertices whose degree could (locally) be reduced without violating convexity.

Once a hole has been selected, Recursor marks all its triangulation edges as constrained again. In the next step it tries to drop these edges in a (different) random order. If this results in a decomposition with no more faces than previously, we keep the new decomposition. Otherwise, we abandon the modifications and restore the old decomposition. See Figure 3 for a sample modification of a decomposition for the Challenge data set `euro-night-0000100`.

Recursor has several parameters to adjust, and we tried to fine-tune them "on the fly" as we applied it to the Challenge instances. Eventually we settled on hole sizes of $7 + P$ faces where P is a random number drawn from a geometric distribution with $p = 0.4$. In each hole, we try a number of decompositions that is equal to the number of triangulation edges in that hole.

Edge flips. Our initial decompositions were based on Delaunay triangulations of the input points. But there is no argument to justify why Delaunay edges were to be preferred over other triangulation edges. Hence, the next improvement of Recursor does a number of

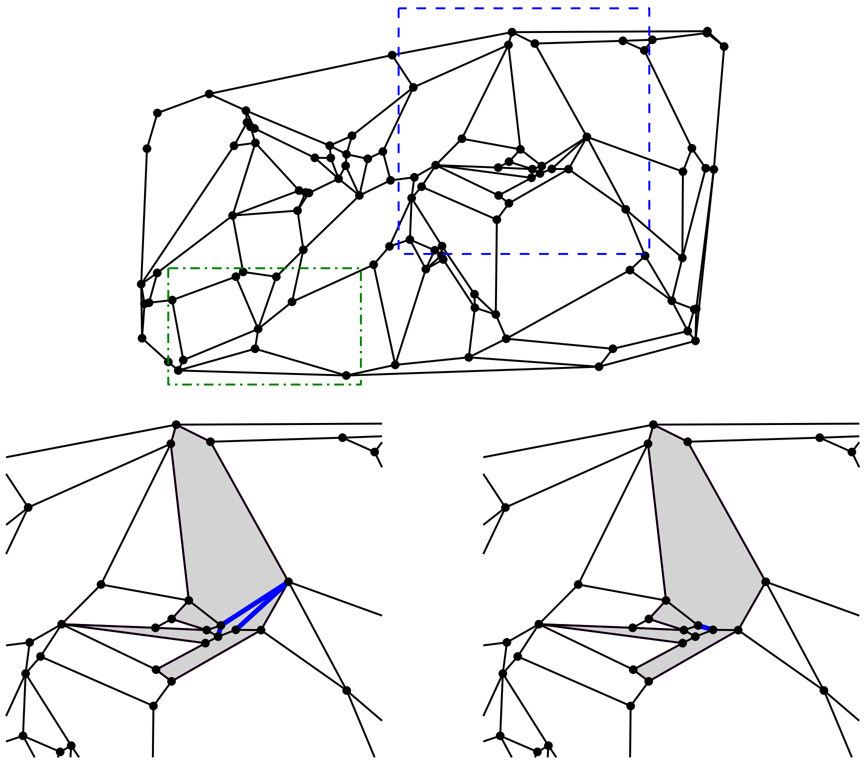


Figure 3 Top: An initial decomposition of `euro-night-0000100` by RECURSOR. Bottom: A detail of the initial decomposition (of the dashed blue frame in the full image), with those seven faces shaded in gray that were selected by the hole-refinement algorithm. The decomposition after one round of local optimization is shown on the right. The edges affected are shown in blue and bold. The improved variant has two faces less.

random edge flips on the triangulation of a hole before attempting to drop edges. The number of edge flips used by our code changed over time. After a series of quick experiments we ended up with using roughly $\sqrt[5]{t}$ edge flips, where t is the number of triangles in the hole.

Continuous refinement. So far, each run of RECURSOR always started from a triangulation of an input. We modified RECURSOR such that it could load a previous decomposition and work on it. This allowed us to run it on different instances whenever we had computational resources to spare, with no need to run it for long continuous stretches of time.

Parallel recursor. RECURSESPLIT is a wrapper around RECURSOR that partitions a given decomposition into a few dozen or a hundred non-overlapping sets of contiguous faces such that each set of faces forms a simply-connected region. Each such region is handed to a dedicated instance of RECURSOR which attempts to reduce the face count within that region. Note that RECURSOR does not require such a region to be convex. Since every individual run of RECURSOR operates strictly within its own region, the resulting decompositions can be merged trivially upon the completion of all runs of RECURSOR.

2.3 Flipper

Flipper was implemented relatively late during the time of the Challenge, not even a month prior to its end. It picks a point set and loads our currently best decomposition for that point set. Then it performs the following steps repeatedly: First, Flipper picks a high-degree vertex v and finds, if one exists, an incident edge (u, v) that can be rotated away from v without violating convexity at either u or v. That is, if u_0, u, u_1 is a CCW ordering of the vertices that share a decomposition edge with v then Flipper attempts to replace the edge (u, v) by either (u, u_0) or (u, u_1) if permissible. See the green edge in Figure 4, left. As shown in Figure 4, right, such a rotation may cause one of the edges incident at v or u to become unnecessary. In that case, we drop it. If, however, no edge can be removed, then the degree of v has decreased and the degree of either u_0 or u_1 has increased by one. Flipper then applies this process to u_0 or u_1.

Variations, added even later, try to pick a specific input point p at regular intervals. Then, with some probability, a rotation may only be carried out if the vertex whose degree is increased by one gets closer to p. The motivation for this decision was that finding edges that can be dropped gets easier if several vertices with higher degree are in close proximity.

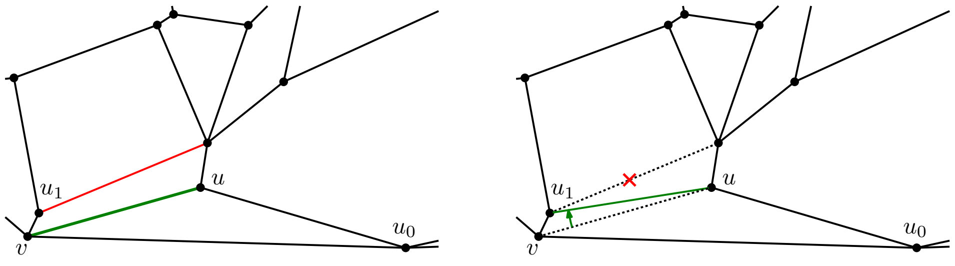


Figure 4 A detail of the initial decomposition (within the dash-dotted green frame of Figure 3): Rotating the green edge allows to drop the red edge while maintaining the convexity of all faces.

2.4 Orthogonal optimizer

Towards the end of the Challenge, a second batch of input instances was made available. While the organizers had warned a priori that the inputs may contain collinear points, the first batch of inputs contained relatively few subsets of collinear points per instance. In contrast, in the second batch of data, each input instance contained points sampled from a dense integer grid, resulting in every input instance containing many subsets of collinear points aligned along horizontal and vertical lines

A visual inspection quickly revealed that the approaches implemented so far did not generate decent decompositions for several inputs of the second batch. Therefore, we were forced to devise and implement a new heuristic. OrthoOpt generates initial convex decompositions geared towards this new type of input instances. It proceeds as follows: First, it sorts the input points of P lexicographically. Then it connects input points that share the same x-coordinate in order of increasing y-coordinates. Finally, it constructs a bottom bounding chain B and a top bounding chain T by linking the bottom-most (top-most, resp.) input points, and it triangulates all pockets between the convex hull of P and the current decomposition, as bounded by B and T. Of course, OrthoOpt can also proceed relative to y-coordinates rather than x-coordinates; see Figure 5. These initial decompositions were passed to Flipper and Recursor for further optimization. In particular, these tools helped to get rid of unnecessary triangulation edges inside of the pockets formed by the convex hull of P and the two chains B and T.

3 Practical computation

3.1 Computational environment

Our tools were run on a diverse set of computers operated by our lab as well as by other groups at the University of Salzburg. We used a varying number of standard PCs plus some (rather small) compute servers, whenever a machine was available. (We did not have access to a genuine high-performance computer.) In particular, we used our own desktop machines whenever they were (partially) idle. One of them, an Intel Core i7-6700 CPU clocked at 3.40 GHz, was used to obtain the performance plot of Figure 6, which shows CPU-time consumptions of several of our tools for Challenge instances with different numbers of points.

Our low-profile way of accessing computers resulted in a highly non-uniform consumption of computational resources, which in turn had highly non-uniform performance levels, ranging from 15-year-old compute servers to machines acquired just a year ago. The availability of a particular machine or of some of its cores was discussed with the operator of that machine on a day-by-day or week-by-week basis. We set up a database and engineered some scripts that allowed all machines to fetch problem instances from and send results back to a home base.

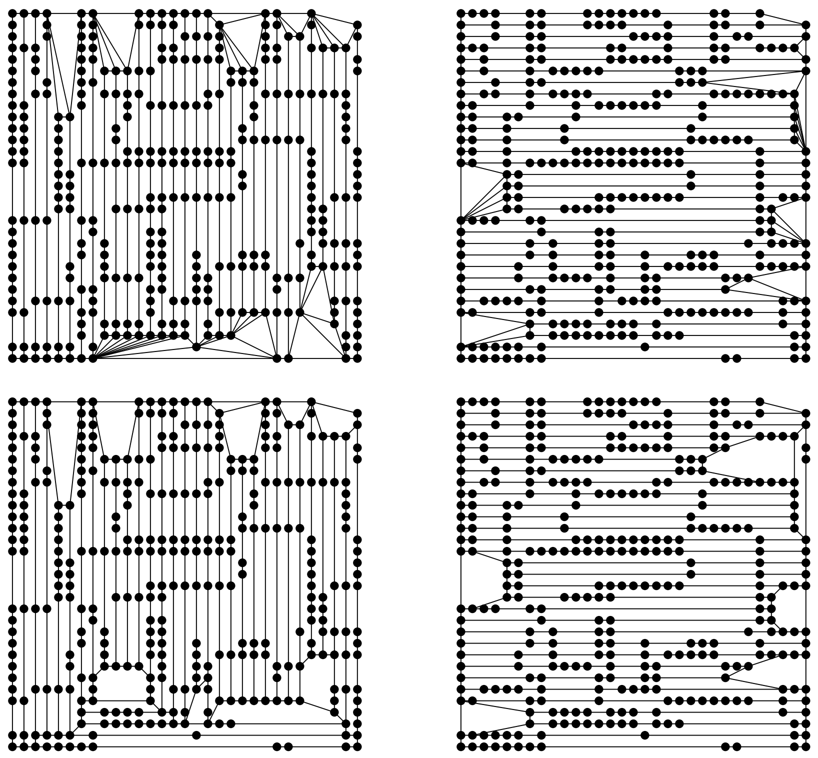


Figure 5 The two top figures show initial decompositions generated by OrthoOpt for the 355-vertex instance `rop0000355`. The bottom left figure shows the best decomposition (with 44 faces) derived from an initial triangulation of `rop0000355`. The bottom right figure shows our overall best decomposition (with 36 faces) derived from an initial decomposition generated by OrthoOpt.

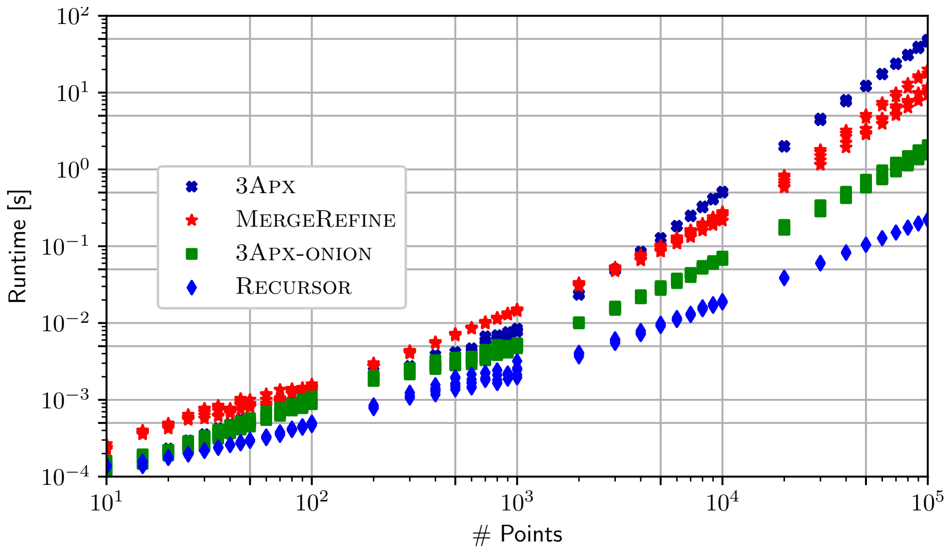


Figure 6 Time needed to obtain one initial decomposition for the competition inputs.

The heterogeneity of (our use of) the computational resources makes it very difficult to come up with a reliable ball-park figure of the total CPU time consumed. We estimate, though, that our tools would have kept a standard desktop machine busy for a few years.

3.2 Experimental results

The estimated quality of a specific convex decomposition is based upon its *score*, where

$$\text{score} := \frac{\text{number of edges in convex partition}}{\text{number of edges in triangulation}}.$$

Figure 7 plots the score for the Challenge instance `euro-night-0100000` over time. It reflects the improvements achieved by refining our tools. While we did not generate such a plot for each and every instance, we did compare sample plots for a few instances: No significant differences were observed. That is, the plot shown in Figure 7 can be regarded as representative for the progress that we made on the Challenge instances of the first batch. The plot shows nicely how Recursor and Flipper interacted. Near the end of the competition, Recursor and Flipper by themselves rarely found better decompositions. However, even when a tool did not reduce the total number of faces, it still restructured the decomposition and uploaded it to our central server, which in turn may have enabled another tool to find some small improvement. The plot also indicates that each new tool yielded a substantial improvement at the beginning, with the gains tapering off as time progressed. So, likely, investing drastically more computational resources than what we had at our disposal would have hardly led to truly substantial improvements. In our case, the availability of human resources for devising and implementing new tools was the decisive limiting factor.

The second batch of Challenge instances made it apparent that our heuristics had been (implicitly) geared towards the inputs that they had to handle. The `rop*` input class proved to be particularly challenging for our initial strategy. Therefore, we introduced OrthoOpt

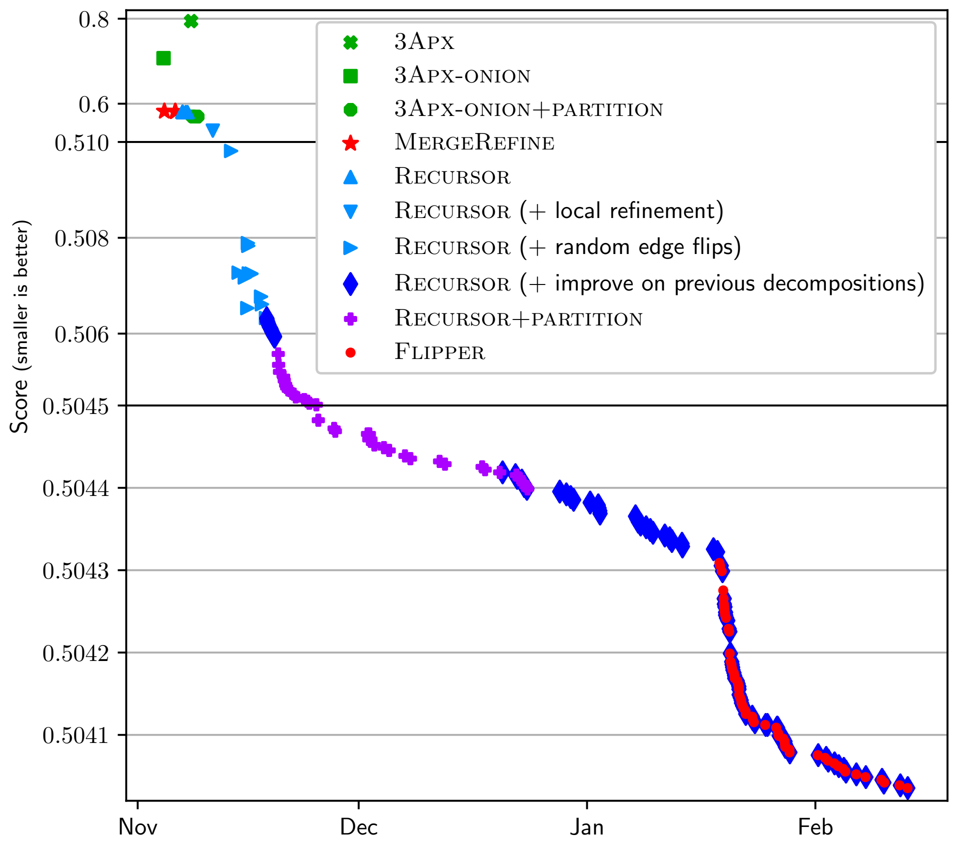


Figure 7 Score over time for `euro-night-0100000`. Note that the y-axis changes scale twice.

to generate initial decompositions that are tailored towards inputs with lots of dense, grid-aligned and, thus, collinear points. Figure 8 illustrates the score over time for `rop0064054` and `ortho_rect_union_47381`, which act as representatives for their corresponding input classes. Apparently, the introduction of OrthoOpt improved our solutions for the `rop*` instances, whereas it provided no improvement for the `ortho_rect_union*` input class.

In Figure 9, we show the scores of the overall best decompositions for various Challenge instances. Additionally, Figure 10 illustrates the development of the average score over time. Note that the significant improvement of the average score in mid January is due to the introduction of Flipper.

4 Conclusion

Our work makes it apparent that well-crafted heuristics run on moderate computing equipment are good enough to achieve decent minimum convex decompositions. But the second batch of Challenge instances made it also apparent that heuristics need not be universally applicable. Rather, they may require an adaption relative to the characteristics of the input data.

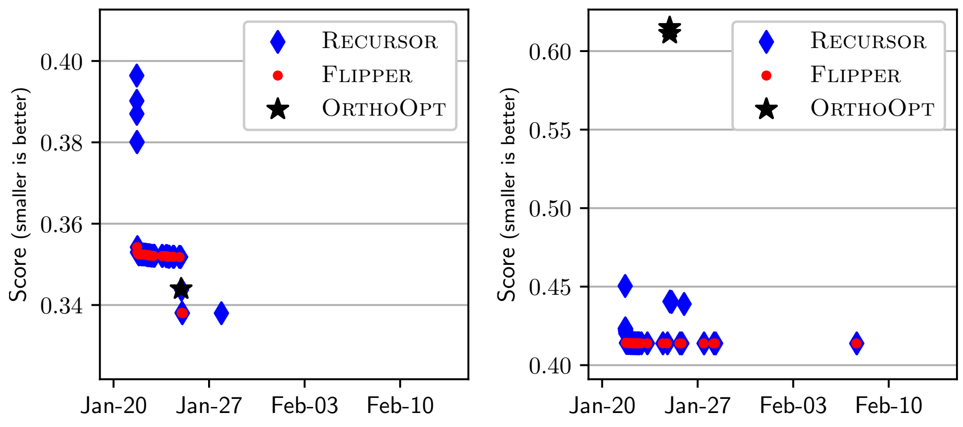


Figure 8 Score over time for `rop0064054` and `ortho_rect_union_47381`.

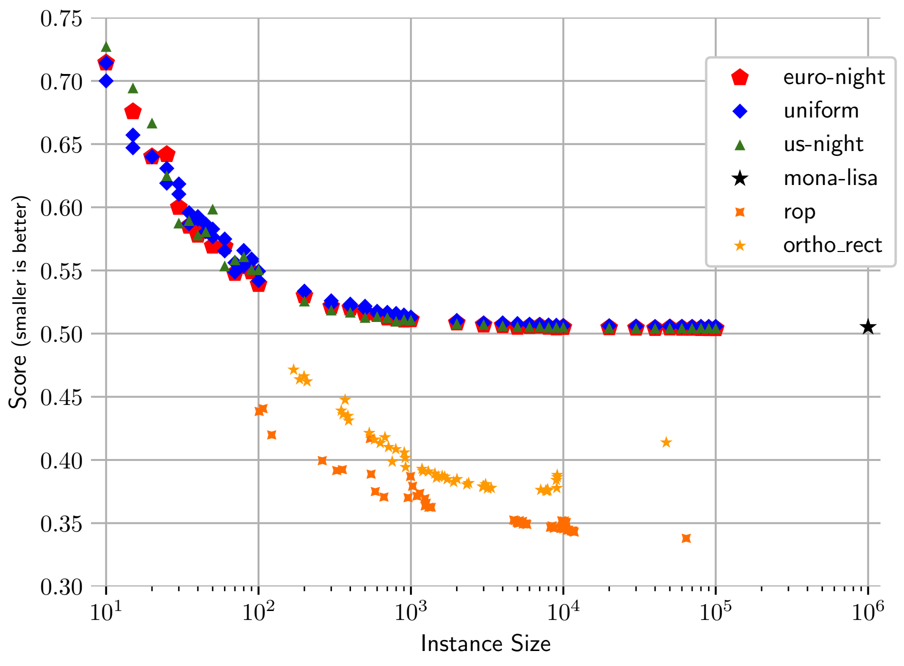
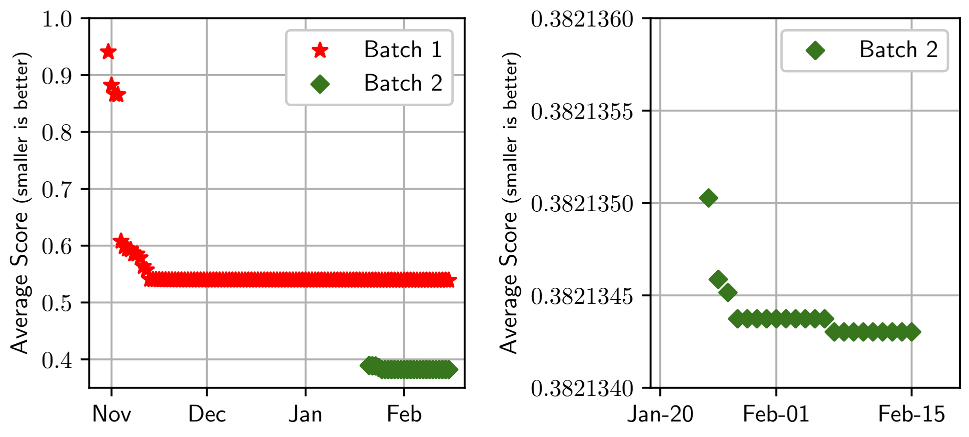


Figure 9 Score per instance.



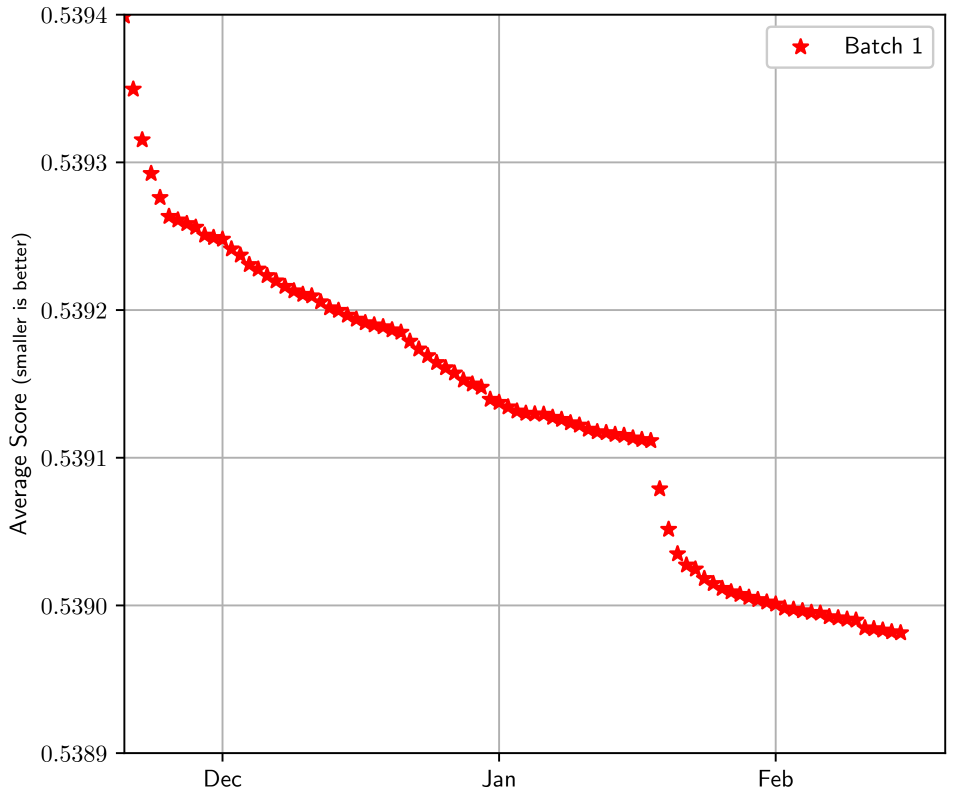


Figure 10 Average score over time.

References

1 Bernard Chazelle. On the Convex Layers of a Planar Set. *IEEE Transactions on Information Theory*, 31(4):509–517, July 1985. doi:10.1109/TIT.1985.1057060.

2 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing Convex Partitions for Point Sets in the Plane: The CG:SHOP Challenge 2020, 2020. arXiv:2004.04207.

3 Christian Knauer and Andreas Spillner. Approximation Algorithms for the Minimum Convex Partition Problem. In *Algorithm Theory – SWAT 2006*, pages 232–241, 2006.

4 Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. ISBN 3-540-61785-X.

An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams

Martin Held and Stefan de Lorenzo [HL20]

Published in:
Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)

Aug. 2020

[HL20]

Martin Held and Stefan de Lorenzo. “An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams”. In: *Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)*. Vol. 173. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Aug. 2020, 56:1–56:15. ISBN: 978-3-95977-162-7. DOI: 10.4230/LIPIcs.ESA.2020.56

An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams

Martin Held
Universität Salzburg, FB Computerwissenschaften, Austria
held@cs.sbg.ac.at

Stefan de Lorenzo
Universität Salzburg, FB Computerwissenschaften, Austria
slorenzo@cs.sbg.ac.at

Abstract

We present a simple wavefront-like approach for computing multiplicatively weighted Voronoi diagrams of points and straight-line segments in the Euclidean plane. If the input sites may be assumed to be randomly weighted points then the use of a so-called overlay arrangement [Har-Peled&Raichel, Discrete Comput. Geom. 53:547–568, 2015] allows to achieve an expected runtime complexity of $\mathcal{O}(n \log^4 n)$, while still maintaining the simplicity of our approach. We implemented the full algorithm for weighted points as input sites, based on CGAL. The results of an experimental evaluation of our implementation suggest $\mathcal{O}(n \log^2 n)$ as a practical bound on the runtime. Our algorithm can be extended to handle also additive weights in addition to multiplicative weights, and it yields a truly simple $\mathcal{O}(n \log n)$ solution for solving the one-dimensional version of this problem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Voronoi Diagram, multiplicative weight, additive weight, arc expansion, overlay arrangement, implementation, experiments, CGAL, exact arithmetic

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.56

Related Version A full version of this paper is available at [9], `https://arxiv.org/abs/2006.14298`.

Supplementary Material The source code of our implementation is available at GitHub and can be used freely under the GPL(v3) license: `https://github.com/cgalab/wevo`.

Funding Work supported by Austrian Science Fund (FWF): Grant P31013-N31.

1 Introduction

The multiplicatively weighted Voronoi diagram (MWVD) was introduced by Boots [4]. Aurenhammer and Edelsbrunner [2] present a worst-case optimal incremental algorithm for constructing the MWVD of a set of n points in $\mathcal{O}(n^2)$ time and space. They define spheres on the bisector circles (that are assumed to be situated in the xy-plane) and convert them into half-planes in $\mathbb{R}^3$ using a spherical inversion. Afterwards, these half-planes are intersected. Thus, every Voronoi region is associated with a polyhedron. Finally, the intersection of every such polyhedron with a sphere that corresponds to the xy-plane is inverted back to $\mathbb{R}^2$. We are not aware of an implementation of their algorithm, though. (And it seems difficult to implement.) In any case, the linear-time repeated searches for weighted nearest points indicate that its complexity is $\Theta(n^2)$ even if the combinatorial complexity of the resulting Voronoi diagram is $o(n^2)$. Later Aurenhammer uses divide&conquer to obtain an $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space algorithm for the one-dimensional weighted Voronoi diagram [1].

Har-Peled and Raichel [8] show that a bound of $\mathcal{O}(n \log^2 n)$ holds on the expected combinatorial complexity of a MWVD if all weights are chosen randomly. They sketch how to compute MWVDs in expected time $\mathcal{O}(n \log^3 n)$. Their approach is also difficult to implement because it uses the algorithm by Aurenhammer and Edelsbrunner [2] as a subroutine.

Vyatkina and Barequet [13] present a wavefront-based strategy to compute the MWVD of a set of n lines in the plane in $\mathcal{O}(n^2 \log n)$ time. The Voronoi nodes are computed based on edge and break-through events. An edge event takes place when an wavefront edge disappears. A break-through event happens whenever a new wavefront edge appears.

Since the pioneering work of Hoff et al. [10] it has been well known that discretized versions of Voronoi diagrams can be computed using the GPU framebuffer. More recently, Bonfiglioli et al. [3] presented a refinement of this rendering-based approach. It is obvious that their approach could also be extended to computing approximate MWVDs. However, the output of such an algorithm is just a set of discrete pixels instead of a continuous skeletal structure. Its precision is limited by the resolution of the framebuffer and by the numerical precision of the depth buffer.

2 Our Contribution

Our basic algorithm allows us to compute MWVDs in worst-case $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space. A refined version makes use of the result by Har-Peled and Raichel [8]: We use their overlay arrangement to keep the expected runtime complexity bounded by $\mathcal{O}(n \log^4 n)$ if the point sites are weighted randomly. Hence, for the price of a multiplicative factor of $\log n$ we get an algorithm that is easier to implement. Our experiments suggest that this bound is too pessimistic in practice and that one can expect the actual runtime to be bounded by $\mathcal{O}(n \log^2 n)$. However, our experiments also show that one may get a quadratic runtime if the weights are not chosen randomly. Our algorithm does not require the input sites to have different multiplicative weights, and it can be extended to additive weights and to (disjoint) straight-line segments as input sites. Furthermore, it yields a truly simple $\mathcal{O}(n \log n)$ solution for computing MWVDs in one dimension, where all input points lie on a line.

Our implementation is based on exact arithmetic and the Computational Geometry Algorithms Library (CGAL) [12]. It is publicly available on GitHub under `https://github.com/cgalab/wevo`. To the best of our knowledge, this is the first full implementation of an algorithm for computing MWVDs that achieves a decent expected runtime complexity.

3 Preliminaries

Let $S := \{s_1, s_2, \ldots, s_n\}$ denote a set of n distinct weighted points in $\mathbb{R}^2$ that are indexed such that $w(s_i) \leq w(s_j)$ for $1 \leq i < j \leq n$, where $w(s_i) \in \mathbb{R}^+$ is the weight associated with s_i. It is common to regard the weighted distance $d_w(p, s_i)$ from an arbitrary point p in $\mathbb{R}^2$ to s_i as the standard Euclidean distance $d(p, s_i)$ from p to s_i divided by the weight of s_i, i.e., $d_w(p, s_i) := \frac{d(p,s_i)}{w(s_i)}$. The *(weighted) Voronoi region* $\mathcal{VR}_w(s_i, S)$ of s_i relative to S is the set of all points of the plane such that no site s_j in $S \setminus \{s_i\}$ is closer to p than s_i, that is, $\mathcal{VR}_w(s_i, S) := \big\{p \in \mathbb{R}^2 : d_w(p, s_i) \leq d_w(p, s_j) \text{ for all } j \in \{1, 2, \ldots, n\}\big\}$. Then the multiplicatively weighted Voronoi diagram (MWVD), $\mathcal{VD}_w(S)$, of S is defined as $\mathcal{VD}_w(S) := \bigcup_{s_i \in S} \partial \mathcal{VR}_w(s_i, S)$.

A connected component of a Voronoi region is called a *face*. For two distinct sites s_i and s_j of S, the *bisector* $b_{i,j}$ of s_i and s_j models the set of points of the plane that are at the same weighted distance from s_i and s_j. Hence, a non-empty intersection of two Voronoi regions is a subset of the bisector of the two defining sites. Following common terminology, a connected component of such a set is called a *(Voronoi) edge* of $\mathcal{VD}_w(S)$. An end-point of an edge is called a *(Voronoi) node*. It is known that the bisector between two unequally weighted sites forms a circle[1]. An example of a MWVD is shown in Figure 1.

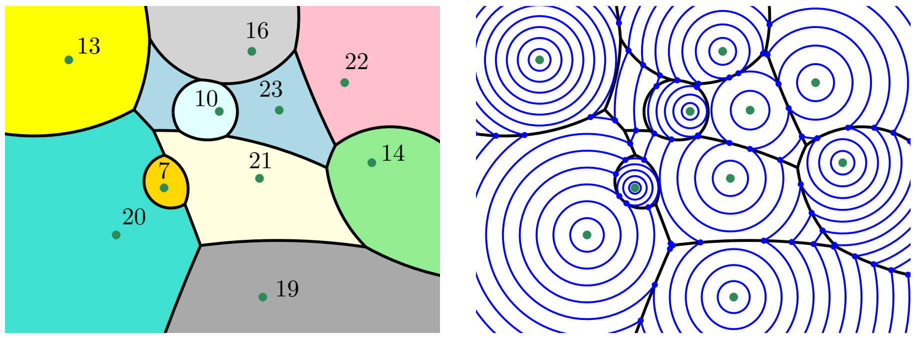


Figure 1 Left: The numbers next to the points indicate their weights and the corresponding MWVD is shown. Right: Wavefronts (in blue) for equally-spaced points in time.

The wavefront $\mathcal{WF}(S,t)$ emanated by S at time $t \geq 0$ is the set of all points p of the plane whose minimal weighted distance from S equals t. More formally,

$$\mathcal{WF}(S,t) := \left\{ p \in \mathbb{R}^2 : \min_{s_i \in S} d_w(p, s_i) = t \right\}.$$

The wavefront consists of circular arcs which we call *wavefront arcs*. A common end-point of two consecutive wavefront arcs is called *wavefront vertex*; see the blue dots in Figure 1.

4 Offset Circles

For the sake of descriptional simplicity, we start with assuming that no point in the plane has the same weighted distance to more than three sites of S. For $t \geq 0$, the *offset circle* $c_i(t)$ of the i-th site s_i is given by a circle centered at s_i with radius $t \cdot w(s_i)$. We find it convenient to regard $c_i(t)$ as a function of either time or distance since at time t every point on $c_i(t)$ is at Euclidean distance $t \cdot w(s_i)$ from s_i, i.e., at weighted distance t. We specify a point of $c_i(t)$ relative to s_i by its polar angle α and its (weighted) polar radius t and denote it by $p_i(\alpha, t)$.

For $1 \leq i < j \leq n$, consider two sites $s_i, s_j \in S$ and assume that $w(s_i) \neq w(s_j)$. Then there exists a unique closed time interval $[t_{ij}^{min}, t_{ij}^{max}]$ during which the respective offset circles of s_i, s_j intersect. We say that the two offset circles *collide* at their mutual *collision time* t_{ij}^{min}, and s_j starts to *dominate* s_i at the domination time t_{ij}^{max}. For all other times t within this interval the two offset circles $c_i(t)$ and $c_j(t)$ intersect in two disjoint points $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$. These *(moving) vertices* trace out the bisector between s_i and s_j; see Figure 2. Since $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$ are defined by the same pair of offset circles we refer to $v_{i,j}^l(t)$ as the *vertex married to* $v_{i,j}^r(t)$, and vice versa. Every other pair of moving vertices defined by two different pairs of intersecting offset circles is called *unmarried*. To simplify the notation, we will drop the parameter t if we do not need to refer to a specific time. Similarly, we drop the superscripts l and r if no distinction between married and unmarried vertices is necessary.

[1] Apollonius of Perga defined a circle as a set of points that have a specific distance ratio to two foci.

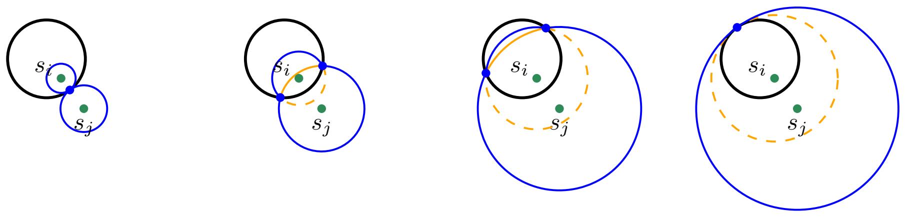


Figure 2 Two married vertices (highlighted by the blue dots) trace out the bisector b_{ij} (in black).

5 A Simple Event-Based Construction Scheme

In this section we describe a simulation of a propagation of the wavefront $\mathcal{WF}(S,t)$ to compute $\mathcal{VD}_w(S)$. Since the wavefront is given by a subset of the arcs of the arrangement of all offset circles, one could attempt to study the evolution of all arcs of that arrangement over time. However, it is sufficient to restrict our attention to a subset of arcs of that arrangement. We note that our wavefront can be seen as a kinetic data structure [7].

Clearly, the arc along $c_i(t)$ which is inside $c_j(t)$ will not belong to $\mathcal{WF}(\{s_i,s_j\},t^*)$ for any $t^* > t$. We will make use of this observation to define *inactive* and *active arcs* that are situated along the offset circles.

▶ **Definition 1** (Active point). *A point p on the offset circle $c_i(t)$ is called* inactive *at time t (relative to S) if there exists $j > i$, with $1 \le i < j \le n$, such that p lies strictly inside of $c_j(t)$. Otherwise, p is* active *(relative to S) at time t. A vertex $v_{i,j}(t)$ is an* active vertex *if it is an active point on both $c_i(t)$ and $c_j(t)$ at time t; otherwise, it is an* inactive vertex.

▶ **Lemma 2.** *If $p_i(\alpha,t)$ is inactive at time t then $p_i(\alpha,t')$ will be inactive for all times $t' \ge t$.*

An inactive point $p_i(\alpha,t)$ cannot be part of the wavefront $\mathcal{WF}(S,t)$. Lemma 2 ensures that none of its future incarnations $p_i(\alpha,t')$ can become part of the wavefront $\mathcal{WF}(S,t')$.

▶ **Definition 3** (Active arc). *For $1 \le i \le n$ and $t \ge 0$, an* active arc *of the offset circle $c_i(t)$ at time t is a maximal connected set of points on $c_i(t)$ that are active at time t. The closure of a maximal connected set of inactive points of $c_i(t)$ forms an* inactive arc *of $c_i(t)$ at time t.*

Every end-point of an active arc of $c_i(t)$ is given by the intersection of $c_i(t)$ with some other offset circle $c_j(t)$, i.e., by a moving vertex $v_{i,j}(t)$. This vertex is active, too.

▶ **Definition 4** (Arc arrangement). *The* arc arrangement *(AA) of S at time t, $\mathcal{A}(S,t)$, is the arrangement induced by all active arcs of all offset circles of S at time t.*

As time t increases, the offset circles expand. This causes the vertices of $\mathcal{A}(S,t)$ to move, but it will also result in topological changes of the arc arrangement.

▶ **Definition 5** (Collision event). *Let $p_i(\alpha,t_{ij}^{min}) = p_j(\alpha+\pi,t_{ij}^{min})$ be the point of intersection of the offset circles of s_i and s_j at the collision time t_{ij}^{min}, for some fixed angle α. A* collision event *occurs between these two offset circles at time t_{ij}^{min} if the points $p_i(\alpha,t)$ and $p_j(\alpha+\pi,t)$ have been active for all times $0 \le t \le t_{ij}^{min}$.*

At the time of a collision a new pair of married vertices $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$ is created. Of course, we have $v_{i,j}^l(t_{ij}^{min}) = v_{i,j}^r(t_{ij}^{min}) = p_i(\alpha,t_{ij}^{min})$.

▶ **Definition 6** (Domination event). *Let $p_i(\alpha, t_{ij}^{max}) = p_j(\alpha, t_{ij}^{max})$ be the point of intersection of the offset circles of s_i and s_j at the domination time t_{ij}^{max}, for some fixed angle α. A* domination event *occurs between these two offset circles at time t_{ij}^{max} if the points $p_i(\alpha, t)$ and $p_j(\alpha, t)$ have been active for all times $0 \leq t \leq t_{ij}^{max}$.*

At the time of a domination event the married vertices $v_{i,j}^l(t_{ij}^{max})$ and $v_{i,j}^r(t_{ij}^{max})$ coincide and are removed.

▶ **Definition 7** (Arc event). *An* arc event *e occurs at time t_e when an active arc a_i shrinks to zero length because two unmarried vertices $v_{i,j}(t_e)$ and $v_{i,k}(t_e)$ meet in a point p_e on $c_i(t_e)$.*

Lemma 2 implies that $p_i(\alpha, t)$ has been active for all times $t \leq t_e$ if $p_i(\alpha, t_e) = p_e$. At the time of an arc event two unmarried vertices trade their places along an offset circle. Now suppose that the two unmarried vertices $v_{i,j}(t_e)$ and $v_{i,k}(t_e)$ meet in a point p_e along $c_i(t_e)$ at the time t_e of an arc event, thereby causing an active arc of $c_i(t_e)$ to shrink to zero length. Hence, the offset circles of s_i, s_j and s_k intersect at the point p_e at time t_e. If $c_j(t)$ and $c_k(t)$ did not intersect for $t < t_e$ then we also get a collision event between $c_j(t)$ and $c_k(t)$ at time t_e, see Figure 3a. (This configuration can occur for any relative order of the weights $w(s_i), w(s_j), w(s_k)$.) Otherwise, one or both of the married vertices $v_{j,k}^l(t_e)$ and $v_{j,k}^r(t_e)$ must also coincide with p_e. If both coincide with p_e then we also get a domination event between $c_j(t)$ and $c_k(t)$ at time t_e and we have $w(s_j) < w(s_k)$, see Figure 3b. The scenarios remaining for the case that only one of $v_{j,k}^l(t_e)$ and $v_{j,k}^r(t_e)$ coincides with p_e are detailed in the following lemma.

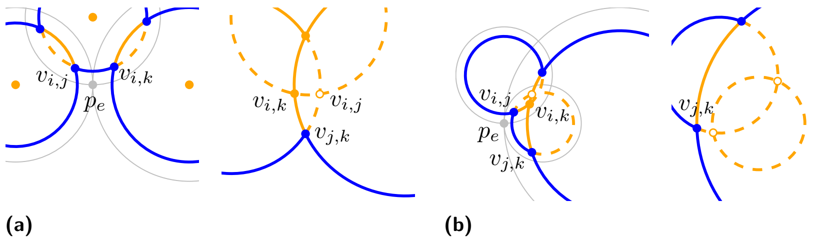
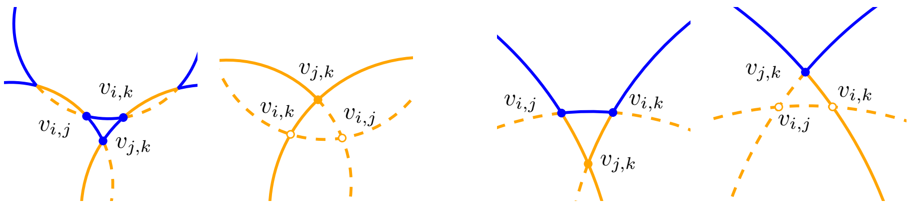


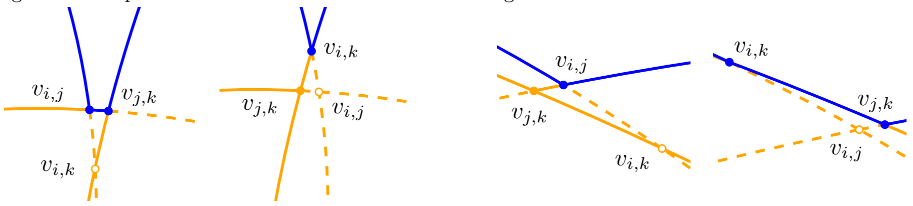
Figure 3 (a) The configuration shortly before (left) and after (right) a collision event as well as an arc event occur simultaneously at the same point p_e. In the left figure the offset arcs at the time of the event are shown in gray. Arcs and vertices that are on $\mathcal{WF}(\{s_i, s_j, s_k\}, t)$ are highlighted in blue. Other active arcs and vertices are depicted by solid orange lines and filled disks, while inactive arcs and vertices are depicted by dashed orange lines and circles. (b) The configuration shortly before and after a domination event and an arc event occur simultaneously at the same point p_e.

▶ **Lemma 8.** *Let $i < j < k$ and consider an arc event such that exactly the three vertices $v_{i,j}(t_e)$, $v_{i,k}(t_e)$, and $v_{j,k}(t_e)$ coincide at time t_e. Then either*
- *all three vertices were active before the event, see Figure 4a, or*
- *$v_{i,j}$ and $v_{j,k}$ were active and $v_{i,k}$ was inactive before the event, see Figure 4b, or*
- *$v_{i,k}$ and $v_{j,k}$ were active and $v_{i,j}$ was inactive before the event, see Figure 4c.*

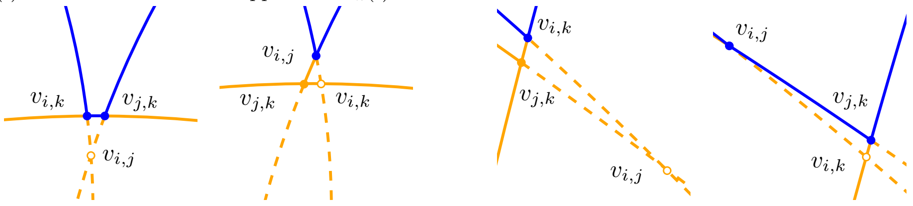
We now describe an event-handling scheme that allows us to trace out $\mathcal{VD}_w(S)$ by simulating the expansion of the arcs of $\mathcal{A}(S, t)$ as t increases, see Figure 5. We refer to this process as *arc expansion.*



(a) The two possible configurations shortly before (shown in the left figures) and after (shown in the right figures) one active arc disappears on $c_i(t)$ if no collision or domination event occurs at the same point. We get the collapse of all three arcs of an active-arc triangle.



(b) The two possible configurations shortly before (left) and after (right) one active arc disappears on $c_j(t)$ and another active arc appears on $c_k(t)$.



(c) The two possible configurations shortly before (left) and after (right) one active arc disappears on $c_k(t)$ and another active arc appears on $c_j(t)$.

Figure 4 The six different configurations that can occur for arc events for $1 \leq i < j < k \leq n$.

For each site we maintain a search data structure to keep track of all active arcs during the arc expansion. This *active offset* o_i of s_i holds the set of all arcs of $c_i(t)$ which are active at time t sorted in counter-clockwise angular order around s_i, and supports the following basic operations in time logarithmic in the number of arcs stored:

- It supports the insertion and deletion of active arcs as well as the lookup of their corresponding vertices.
- It supports point-location queries, allowing us to identify that active arc within o_i which contains a query point p on $c_i(t)$.

Every active offset contains at most $2(n-1)$ vertices and, thus, $O(n)$ active arcs. Hence, each such operation on an active offset takes $\mathcal{O}(\log n)$ time in the worst case.

Checking and handling the configurations shown in Figures 3a to 4 can be done by using only basic operations within the respective active offsets. The events themselves are stored in a priority queue $\mathcal{Q}$ ordered by the time of their occurrence. If two events take place simultaneously at the same point then collision events are prioritized higher than arc events, and arc events have to be handled before domination events. Four auxiliary operations are utilized that allow a more compact description of this process. Each one takes $\mathcal{O}(\log n)$ time.

- The *collapse-operation* takes place from $v_{i,x}$ to $v_{j,k}$ within an active offset o_x, with $x \in \{j,k\}$, in which $v_{i,x}$ and $v_{j,k}$ bound an active arc a_x that is already part of o_x; see Figure 6a. It determines the neighboring active arc a'_x of a_x that is bounded (on one side) by $v_{i,x}$, deletes a_x from o_x, and replaces $v_{i,x}$ by $v_{j,k}$ in a'_x.

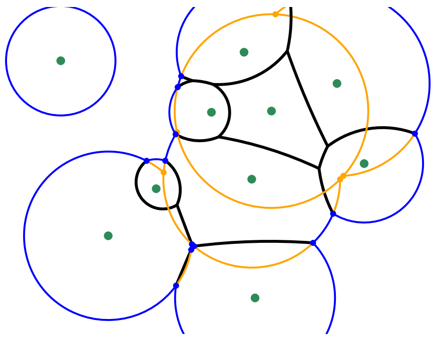


Figure 5 A snapshot of the arc expansion for the input shown in Figure 1. Active arcs that are currently not part of the wavefront are drawn in orange.

- The counterpart of the collapse-operation is the *expand-operation*; see Figure 6b. It happens from $v_{j,k}$ to $v_{i,x}$ in which $v_{j,k}$ bounds an active arc a'_x within o_x. The expansion will either move along a currently inactive or an already active portion of the offset circle of s_x. In the latter case, $v_{j,k}$ is replaced by $v_{i,x}$ in a'_x. In any case, we insert the respective active arc that is bounded by $v_{i,x}$ and $v_{j,k}$ into o_x.
- A *split-operation* involves two active offsets o_i and o_j as well as a point p_e which is situated within the active arcs $a_i := (v_{i,s}, v_{i,e})$ and $a_j := (v_{j,s'}, v_{j,e'})$ within o_i and o_j, respectively; see Figure 7a. Two married vertices $v^l_{i,j}$ and $v^r_{i,j}$ are created. Afterwards a_i and a_j are removed from o_i and o_j, respectively. Two new active arcs $(v_{i,s}, v^l_{i,j})$ and $(v^r_{i,j}, v_{i,e})$ are created and inserted into o_i. Furthermore, the three active arcs $(v_{j,e}, v^r_{i,j})$, $(v^r_{i,j}, v^l_{i,j})$, and $(v^l_{i,j}, v_{j,e})$ are inserted into o_j. If a_i and a_j were wavefront arcs then the newly created married vertices coincide with wavefront vertices and the newly inserted active arcs except $(v^r_{i,j}, v^l_{i,j})$ are marked as wavefront arcs.
- During a *merge-operation*, exactly two offset circles interact; see Figure 7b. The active arcs a_i and a_j bounded by the two corresponding married vertices $v^r_{i,j}$ and $v^l_{i,j}$ are removed from o_i and o_j, respectively. Additionally, the active arcs $(v_{j,s}, v^r_{i,j})$ and $(v^l_{i,j}, v_{j,e})$ that were adjacent to a_j within o_j are removed. Finally, a new active arc $a'_j := (v_{j,s}, v_{j,e})$ is inserted into o_j. If a_j was a wavefront arc then a'_j is also marked as a wavefront arc.

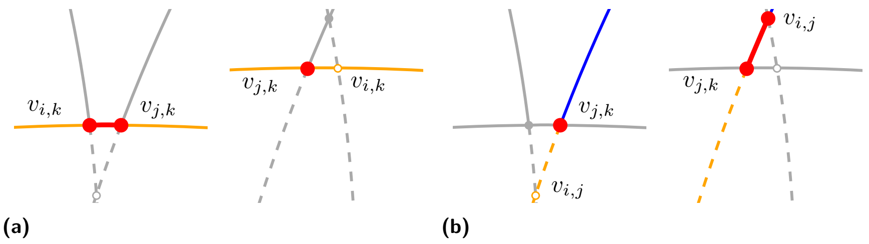


Figure 6 (a) A collapse-operation from $v_{i,k}$ to $v_{j,k}$ takes place within o_k. (b) An expand-operation happens within o_j from $v_{j,k}$ to $v_{i,j}$.

Domination events and arc events are easy to detect. The point and time of a collision is trivial to compute for any pair of offset circles, too. Unfortunately there is no obvious way to identify those pairs of circles for which this intersection will happen within portions of these offset circles which will still be active at the time of the collision. Hence, for the rest of

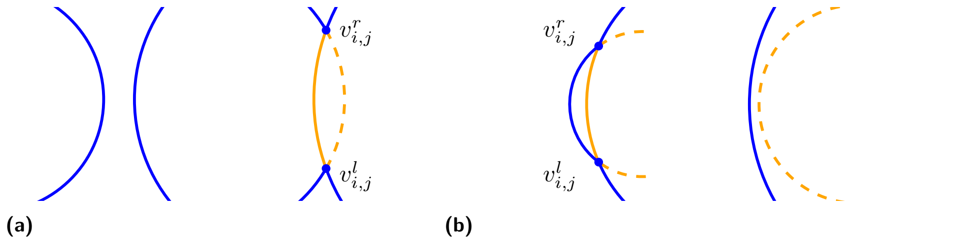


Figure 7 (a) A split-operation happens when at the time of a collision event. (b) A merge-operation happens at the time of a domination event.

this section we assume that all collisions among all pairs of offset circles are computed prior to the actual arc expansion. Lemma 9 verifies that our algorithm correctly simulates the arc expansion.

▶ **Lemma 9.** *For time $t > 0$, the arc arrangement $\mathcal{A}(S, t)$ can be obtained from $\mathcal{A}(S, 0)$ by modifying it according to all collision events, domination events and arc events that occur till time t, in the order in which they appear.*

If the maximum weight of all sites is associated with only one site then there will be a time t when the offset circle of this site dominates all other offset circles, i.e., when $\mathcal{WF}(S, t)$ contains only this offset circle as one active arc. Obviously, at this time no further event can occur and the arc expansion stops. If multiple sites have the same maximum weight then $\mathcal{Q}$ can only be empty once $\mathcal{WF}(S, t)$ contains only one loop of active arcs which all lie on offset circles of these sites and if all wavefront vertices move along rays to infinity.

▶ **Lemma 10.** *An active arc or active vertex within an active offset is identified and marked as a wavefront arc (wavefront vertex, resp.) at time $t \geq 0$ if and only if it lies on $\mathcal{WF}(S, t)$.*

If we allow points in $\mathbb{R}^2$ to have the same weighted distance to more than three sites then we need to modify our strategy. In particular, we need to take care of constellations in which more than three arc events happen simultaneously at the same point. In such a case it is necessary to carefully choose the sequence in which the corresponding arc events are handled. More precisely, an arc event may only be handled (without corrupting the state of the active offsets) whenever the respective active vertices are considered neighboring within the active offsets. If the active vertices that participate in an arc event are not currently neighboring then we can always find an arc event whose active vertices are neighboring that happens simultaneously at the same location by walking along the corresponding active offsets. By dealing with the arc events in this specific order, we generate multiple coinciding Voronoi nodes of degree three. Domination events that occur simultaneously at the same point p_e are processed in increasing order of the weights. Note that this order can already be established at the time when an event is inserted into $\mathcal{Q}$, at no additional computational cost. Simultaneous multiple collision events at the same point p_e either involve arcs that are not active or coincide with arc events. These arc events automatically establish a sorted order of the active arcs around p_e, thus allowing us to avoid an explicit (and time-consuming) sorting.

▶ **Lemma 11.** *During the arc expansion $\mathcal{O}(n^2)$ collision and domination events are computed.*

We know that collision events create and domination events remove active vertices (and make them inactive for good). A collapse of an entire active-arc triangle causes two vertices to become inactive. During every other arc event at least one active vertex becomes inactive, but at the same time one inactive vertex may become active again. In order to bound the number of arc events it is essential to determine how many vertices can be active and how often a vertex can undergo a *reactivation*, i.e., change its status from inactive to active. (Note that Lemma 2 is not applicable to a moving vertex since its polar angle does not stay constant.) We now argue that the total number of reactivations of inactive vertices is bounded by the number of different vertices that ever were active during the arc expansion.

▶ **Lemma 12.** *Every reactivation of a moving vertex during an arc event forces another moving vertex to become inactive and remain inactive for the rest of the arc expansion.*

▶ **Lemma 13.** *Let h be the number of different vertices that ever were active during the arc expansion. Then $\mathcal{O}(h)$ arc events can take place during the arc expansion.*

▶ **Theorem 14.** *The multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n weighted point sites can be computed in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space.*

Additionally, in the full version [9] we argue that the one-dimensional MWVD can be computed efficiently using a wavefront-based strategy.

▶ **Theorem 15.** *The multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n weighted point sites in one dimension can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.*

6 Reducing the Number of Collisions Computed

Experiments quickly indicate that the vast majority of pairwise collisions computed a priori never ends up on pairs of active arcs. Furthermore, the resulting Voronoi diagrams show a quadratic combinatorial complexity only for contrived input data. We make use of the following results to determine all collision events in near-linear expected time. Throughout this section, we assume that for each site $s_i \in S$ the corresponding weight $w(s_i)$ is independently sampled from some probability distribution.

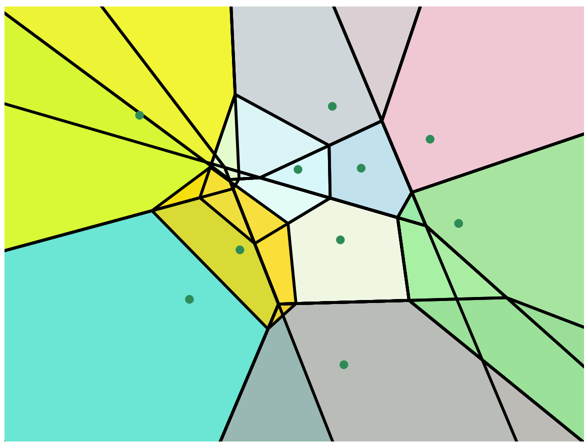


Figure 8 The overlay arrangement is generated by inserting the sites ordered by decreasing weights.

▶ **Definition 16** (Candidate Set). *Consider an arbitrary (but fixed) point $q \in \mathbb{R}^2$, and let s be its nearest neighbor in S under the weighted distance. Let $s' \in S \setminus \{s\}$ be another site. Since s is the nearest neighbor of q we know that either s has a higher weight than s' or a* *smaller Euclidean distance to q than s'. Thus, one can define a* candidate set *for a weighted nearest neighbor of q which consists of all sites $s \in S$ such that all other sites in S either have a smaller weight or a larger Euclidean distance to q.*

▶ **Lemma 17** (Har-Peled and Raichel [8]). *For all points $q \in \mathbb{R}^2$, the candidate set for q among S is of size $\mathcal{O}(\log n)$ with high probability.*

▶ **Lemma 18** (Har-Peled and Raichel [8]). *Let K_i denote the Voronoi cell of s_i in the unweighted Voronoi diagram of the i-th suffix $S_i := \{s_i, \dots, s_n\}$. Let $\mathcal{OA}$ denote the arrangement formed by the overlay of the regions $K_1, \dots, K_n$. Then, for every face f of $\mathcal{OA}$, the candidate set is the same for all points in f.*

Figure 8 shows a sample overlay arrangement. Kaplan et al. [11] prove that this overlay arrangement has an expected complexity of $\mathcal{O}(n \log n)$. Note that their result is applicable since inserting the points in sorted order of their randomly chosen weights corresponds to a randomized insertion. These results allow us to derive better complexity bounds.

▶ **Theorem 19** (Kaplan et al. [11]). *The expected combinatorial complexity of the overlay of the minimization diagrams that arises during a randomized incremental construction of the lower envelope of n hyperplanes in $\mathbb{R}^d$, for $d \geq 2$, is $\mathcal{O}(n^{\lfloor d/2 \rfloor})$, for d even, and $O(n^{\lfloor d/2 \rfloor} \log n)$, for d odd. The bounds for d even and for $d = 3$ are tight in the worst case.*

▶ **Lemma 20.** *If a collision event occurs between the offset circles of two sites $s_i, s_j \in S$ then there exists at least one candidate set which includes both s_i and s_j.*

▶ **Theorem 21.** *All collision events can be determined in $\mathcal{O}(n \log^3 n)$ expected time by computing the overlay arrangement $\mathcal{OA}$ of a set S of n input sites.*

Thus, the number h of vertices created during the arc expansion can be expected to be bounded by $\mathcal{O}(n \log^3 n)$. Lemma 13 tells us that the number of arc events is in $\mathcal{O}(h)$. Therefore, $\mathcal{O}(n \log^3 n)$ events happen in total.

▶ **Theorem 22.** *A wavefront-based approach allows to compute the multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n (randomly) weighted point sites in expected $\mathcal{O}(n \log^4 n)$ time and expected $\mathcal{O}(n \log^3 n)$ space.*

7 Extensions

Consider a set S' of n disjoint weighted straight-line segments in $\mathbb{R}^2$. A wavefront propagation among weighted line segments requires us to refine our notion of "collision". We call an intersection of two offset circles a *non-piercing collision event* if it marks the initial contact of the two offset circles. That is, it occurs when the first pair of moving vertices appear. We call an intersection of two offset circles a *piercing collision event* if it takes place when two already intersecting offset circles intersect in a third point for the first time; see Figure 9. In this case, a second pair of moving vertices appear.

Hence, a minor modification of our event-based construction scheme is sufficient to extend it to weighted straight-line segments; see Figure 10. We only need to check whether a piercing collision event that happens at a point p_e at time t_e currently is part of $\mathcal{WF}(S', t_e)$. In such a case the two new vertices as well as the corresponding active arc between them need to be flagged as part of $\mathcal{WF}(S', t_e)$.

An extension to additive weights can be integrated easily into our scheme by simply giving every offset circle a head-start of $w_a(s_i)$ at time $t = 0$, where $w_a(s_i) \geq 0$ denotes the real-valued additive weight that is associated with s_i.

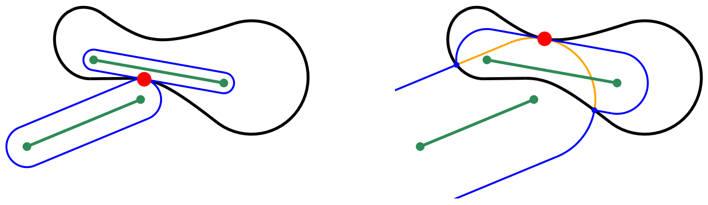


Figure 9 An example of a non-piercing (left) as well as a piercing collision event (right).

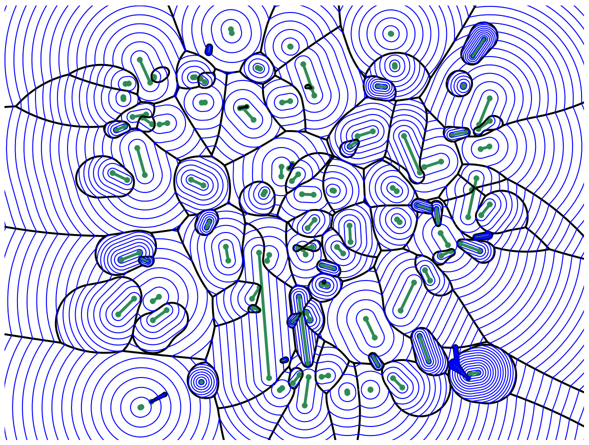


Figure 10 The MWVD of a set of weighted points and weighted straight-line segments together with a family of wavefronts for equally-spaced points in time.

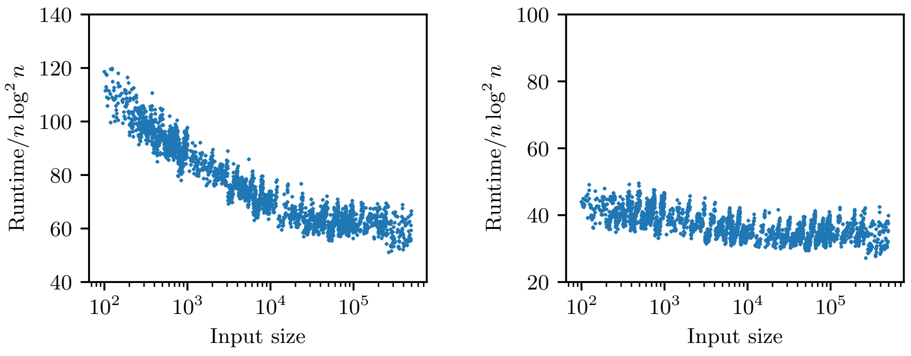
8 Experimental Evaluation

We implemented our full algorithm for multiplicatively weighted points as input sites[2], based on CGAL and exact arithmetic[3]. In particular, we use CGAL's `Arrangement_2` package for computing the overlay arrangement and its `Voronoi_diagram_2` package for computing unweighted Voronoi diagrams. The computation of the MWVD itself utilizes CGAL's `Exact_circular_kernel_2` package which is based on the `Gmpq` number type. The obvious advantage of using exact number types is that events are guaranteed to be processed in the right order even if they occur nearly simultaneously at nearly the same place. One of the main drawbacks of exact number types is their memory consumption which is significantly (and sometimes unpredictably) higher than when standard floating-point numbers are used.

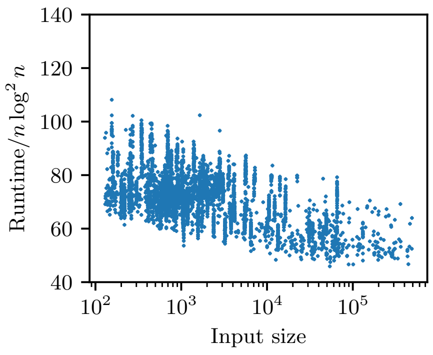
We used our implementation for an experimental evaluation and ran our code on over 8000 inputs ranging from 256 vertices to 500 000 vertices. For all inputs all weights were chosen uniformly at random from the interval $[0, 1]$. All tests were carried out with CGAL 5.0 on an Intel Core i9-7900X processor clocked at 3.3 GHz.

[2] We do also have a prototype implementation that handles both weighted points and weighted straight-line segments. It was used to generate Figure 10.

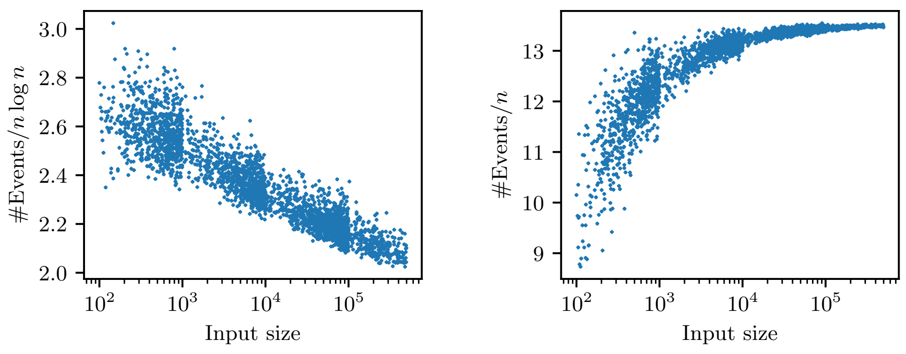
[3] We have not spent enough time on fine-tuning an implementation based on conventional floating-point arithmetic. The obvious crux is that inaccurately determined event times (and locations) may corrupt the state of the arc arrangement and, thus, cause a variety of errors during the subsequent arc expansion.



(a) Left: The overall runtime results for inputs with randomly generated weights and point coordinates. Right: The runtime consumed by the computation of the corresponding overlay arrangements. All runtimes were divided by $n \log^2 n$.



(b) The overall runtime results for inputs with randomly generated weights and vertices of real-world polygons and polygons of the Salzburg database of polygonal data [5, 6] taken as input points. The runtimes were divided by $n \log^2 n$.



(c) The left plot shows the total number of (valid and invalid) collision events (divided by $n \log n$); the right plot shows the number of arc events (divided by n) processed during the arc expansion. All point coordinates and weights were generated randomly.

Figure 11 Experimental evaluation.

In any case, the number of events is smaller than predicted by the theoretical analysis. This is also reflected by our runtime statistics: In Figures 11a and 11b the runtime that was consumed by the computation of a MWVD is plotted. We ran our tests on two different input classes: The point locations were either generated randomly, i.e., they were chosen according to either a uniform or a normal distribution, or obtained by taking the vertices of real-world polygons or polygons of the brand-new Salzburg database of polygonal data [5, 6]. Summarizing, our tests suggest an overall runtime of $\mathcal{O}(n \log^2 n)$ for both input classes. In particular, the actual geometric distribution of the sites does not have a significant impact on the runtime if the weights are chosen randomly: For real-world, irregularly distributed sites the runtimes are scattered more wildly than in the case of uniformly distributed sites, but they do not increase. The numbers of collision events and arc events that occurred during the arc expansion are plotted in Figure 11c. Our tests suggest that we can expect to see at most $3n \log n$ collision events and at most most $14n$ arc events to occur. Note that the number of arc events forms an upper bound on the number of Voronoi nodes of the final MWVD. That is, random weights seem to result in a linear combinatorial complexity of the MWVD.

It is natural to ask how much these results depend on the randomness of the weights. To probe this question we set up a second series of experiments: We sampled points uniformly within a square with side-length $\sqrt{2}$ and then tested different weights. Let $d(s)$ be the distance of the site $s \in S$ from the center of the square, and let $r(s)$ be a number uniformly distributed within the interval $[0, 1]$. Of course, $0 \leq d(s) \leq 1$. Then we assign ${}^{\alpha\cdot d(s)+\beta\cdot r(s)}/_{(\alpha+\beta)}$ as weight to s, with α and β being the same arbitrary but fixed non-negative numbers for all sites of S. Figure 12 shows the results obtained for the same sets of points and the (α, β)-pairs $(1, 0)$, $(9, 1)$, $(7, 3)$, $(1, 1)$ and $(0, 1)$. This test makes it evident that the bounds on the complexities need not hold if the weights are not chosen randomly, even for a uniform distribution of the sites. Rather, this may lead to a linear number of candidates per candidate set and a quadratic runtime complexity, as shown in Figure 12.

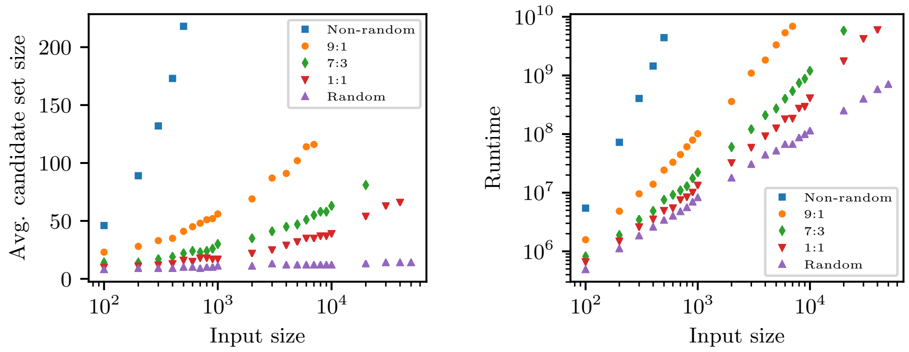


Figure 12 The plots show how the average number of candidates (left) and the total runtime (right) depend on the weights assigned to the sites. Each marker on the x-axes indicates the number n of input sites uniformly distributed within a square.

9 Conclusion

We present a wavefront-like approach for computing the MWVD of points and straight-line segments. Results by Kaplan et al. [11] and Har-Peled and Raichel [8] allow to predict an $\mathcal{O}(n \log^4 n)$ expected time complexity for point sites with random weights. We also discuss a robust, practical implementation which is based on CGAL and exact arithmetic. Extensive tests of our code indicate an average runtime of $\mathcal{O}(n \log^2 n)$ if the sites are weighted randomly. To the best of our knowledge, there does not exist any other code for computing MWVDs that is comparatively fast. A simple modification of our arc expansion scheme makes it possible to handle both additive and multiplicative weights simultaneously. Our code is publicly available on GitHub under `https://github.com/cgalab/wevo`. Figure 13 shows several examples of MWVDs computed by our implementation.

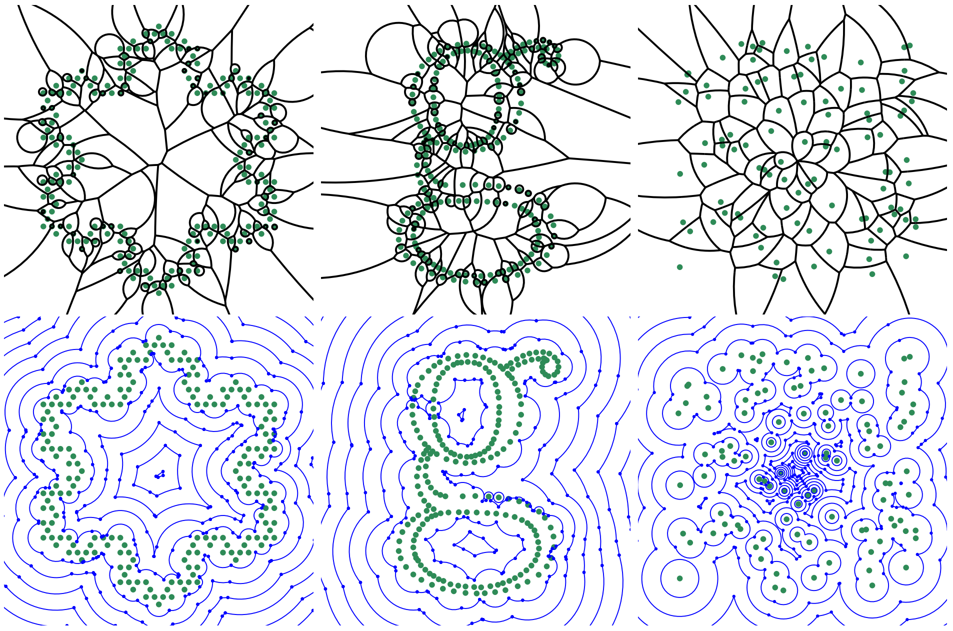


Figure 13 Several examples of MWVDs are shown in the top figures. The bottom figures illustrate a series of uniformly distributed wavefronts that have been derived from the corresponding MWVDs.

References

1 Franz Aurenhammer. The One-Dimensional Weighted Voronoi Diagram. *Information Processing Letters*, 22(3):119–123, 1986. `doi:10.1016/0020-0190(86)90055-4`.

2 Franz Aurenhammer and Herbert Edelsbrunner. An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane. *Pattern Recognition*, 17(2):251–257, 1984. `doi:10.1016/0031-3203(84)90064-5`.

3 Rudi Bonfiglioli, Wouter van Toll, and Roland Geraerts. GPGPU-Accelerated Construction of High-Resolution Generalized Voronoi Diagrams and Navigation Meshes. In *Proceedings of the Seventh International Conference on Motion in Games*, pages 26–30, 2014. `doi:10.1145/2668084.2668093`.

4 Barry N. Boots. Weighting Thiessen Polygons. *Economic Geography*, 56(3):248–259, 1980. `doi:10.2307/142716`.

5 Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. On Generating Polygons: Introducing the Salzburg Database. In *Proceedings of the 36th European Workshop on Computational Geometry*, pages 75:1–75:7, March 2020.

6 Computational Geometry and Applications Lab Salzburg. Salzburg Database of Geometric Inputs. `https://sbgdb.cs.sbg.ac.at/`, 2020.

7 Leonidas Guibas. Kinetic Data Structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, pages 23.1–23.18. Chapman and Hall/CRC, 2001. ISBN 9781584884354.

8 Sariel Har-Peled and Benjamin Raichel. On the Complexity of Randomly Weighted Multiplicative Voronoi Diagrams. *Discrete & Computational Geometry*, 53(3):547–568, 2015. `doi:10.1007/s00454-015-9675-0`.

9 Martin Held and Stefan de Lorenzo. An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams, 2020. `arXiv:2006.14298`.

10 Kenneth E. Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast Computation of Generalized Voronoi Diagrams using Graphics Hardware. In *Proceedings of the the 26th Annual International Conference on Computer Graphics and Interactive Techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999. `doi:10.1145/311535.311567`.

11 Haim Kaplan, Edgar Ramos, and Micha Sharir. The Overlay of Minimization Diagrams in a Randomized Incremental Construction. *Discrete & Computational Geometry*, 45(3):371–382, 2011. `doi:10.1007/s00454-010-9324-6`.

12 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0 edition, 2019. URL: `https://doc.cgal.org/5.0/Manual/packages.html`.

13 Kira Vyatkina and Gill Barequet. On Multiplicatively Weighted Voronoi Diagrams for Lines in the Plane. *Transactions on Computational Science*, 13:44–71, 2011. `doi:10.1007/978-3-642-22619-9_3`.

Weighted Skeletal Structures for Computing Variable-Radius Offsets

Martin Held and Stefan de Lorenzo [HL21]

Published in:
Computer-Aided Design and Applications

Jan. 2021

[HL21]

Martin Held and Stefan de Lorenzo. "Weighted Skeletal Structures for Computing Variable-Radius Offsets". In: *Computer-Aided Design and Applications* 18.5 (Jan. 2021), pp. 875–889. DOI: `10.14733/cadaps.2021.875-889`

Weighted Skeletal Structures For Computing Variable-Radius Offsets

Martin Held[1], Stefan de Lorenzo[2]

[1]University of Salzburg, held@cs.sbg.ac.at
[2]University of Salzburg, slorenzo@cs.sbg.ac.at

Corresponding author: Stefan de Lorenzo, slorenzo@cs.sbg.ac.at

Abstract. Held et al. [CAD&A 2016] introduced generalized weighted Voronoi diagrams as a tool to construct variable-radius offsets. We revisit this structure and provide two algorithms to compute it. We also introduce variable-radius skeletons of polygons as a closely related structure which inherits most properties of generalized weighted Voronoi diagrams except that its skeletal regions are always connected. Experimental results obtained by our implementation of variable-radius skeletons are discussed. In addition to variable-radius offsets we demonstrate how this structure can be used to generate intricate roofs for polygonal footprints of buildings in an automated way.

Keywords: Voronoi diagram, multiplicative weight, wavefront propagation, variable-radius skeleton, variable-radius offsets
DOI: https://doi.org/10.14733/cadaps.2021.875-889

1 Introduction

Offsetting is an essential task in many industrial applications. In conventional *constant-radius offsetting* all parts of the input set expand or shrink uniformly at the same speed. A natural extension of this idea is to allow parts to expand or shrink in a non-uniform manner. Among other fields, such so-called *variable-radius offsets* find applications in brush-stroke modeling and the generation of ornamental seams. See Figure 1 for sample constant-radius and variable-radius offsets of a set of straight-line segments that are arranged in a star-like fashion. In the sequel we present skeletal structures that support the computation of variable-radius offsets.

Consider a set S^* of n points in the plane. The *Voronoi diagram* $\mathcal{VD}(S^*)$ of S^* partitions the plane into interior-disjoint, convex areas — so-called Voronoi regions — such that the Voronoi region $\mathcal{VR}(s, S^*)$ of $s \in S^*$ contains all points of the plane that are closer to s under the Euclidean distance metric than to any other point of S^*; see Figure 2a. It is well-known that $\mathcal{VD}(S^*)$ has $\mathcal{O}(n)$ nodes and (straight-line) edges. In the well-known prairie-fire analogy, $\mathcal{VD}(S^*)$ is given by those points of the plane where fire waves meet

- if fires are ignited simultaneously at all points of S^*, and
- if all fires spread uniformly at the same speed.

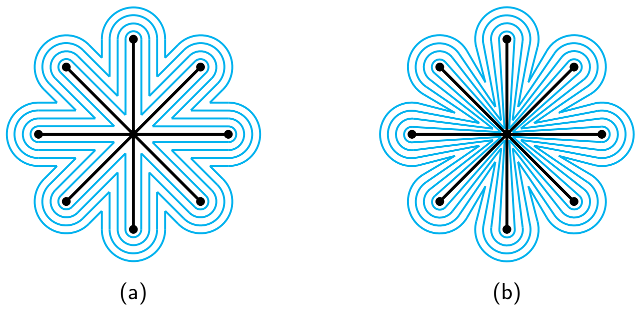


Figure 1: A family of constant-radius and variable-radius offsets (in blue) of straight-line segments (in black).

These fire waves form instances of (constant-radius) offset curves of the points of S^* for specific offset distances; see Figure 2b. In the field of computational geometry, offset curves are known as *wavefronts*, and a simulation of the wavefronts for steadily increasing offset distances is called *wavefront propagation*.

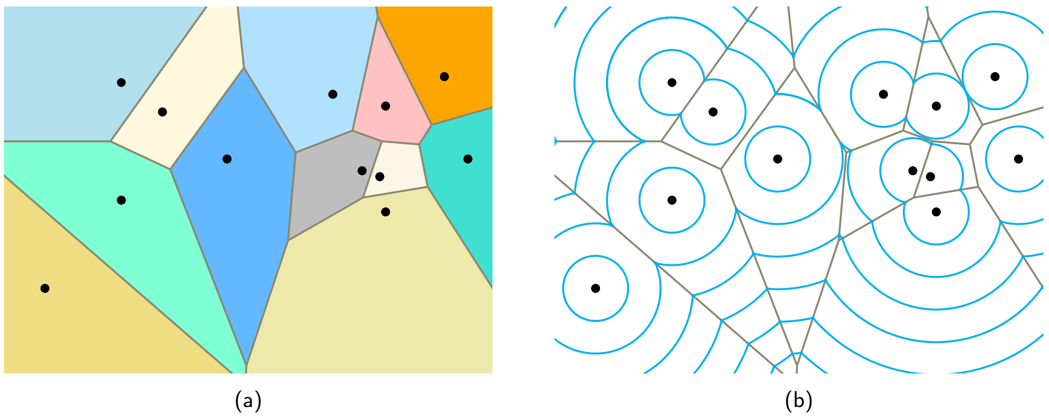


Figure 2: (a) Voronoi diagram and (b) a family of wavefronts for a set of points. Each Voronoi region is indicated by a colored area.

In more formal terms,

$$\mathcal{VD}(S^*) := \bigcup_{s \in S^*} \partial\mathcal{VR}(s, S^*),$$

where

$$\mathcal{VR}(s, S^*) := \{p \in \mathbb{R}^2 : d(p, s) \leq d(p, S^*)\},$$

and $\partial\mathcal{VR}(s, S^*)$ denotes the boundary of $\mathcal{VR}(s, S^*)$. Lots of generalizations of the standard Voronoi diagram have been studied. See, e,g., generalizations in terms of dimensionality [7], the types of permissible input objects [10], and distance metric [14].

Another way to generalize Voronoi diagrams is to assign multiplicative positive weights to the points of S^* and to let them influence the expansion speeds of the wavefronts. This yields the *multiplicatively weighted Voronoi diagram* of S^*; see Figure 3a. More formally, the weighted distance $d_w(p, s)$ between a point $p \in \mathbb{R}^2$

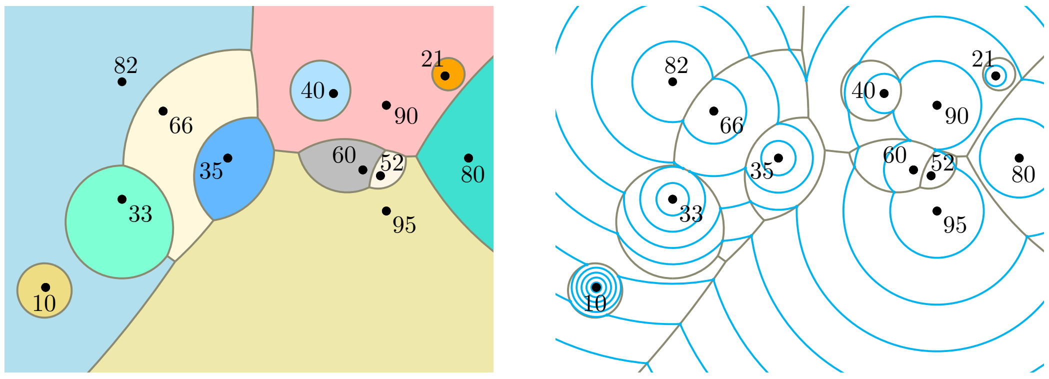


Figure 3: (a) The multiplicatively weighted Voronoi diagram and (b) corresponding family of wavefronts for a set of weighted points. The multiplicative weights are written next to the points; their positions are identical to the positions shown in Figure 2a.

and a weighted point $s \in S^*$ is defined as

$$d_w(p, s) := \frac{d(p, s)}{w(s)},$$

where $d(p, s)$ denotes the standard Euclidean distance between p and s, and $w(s)$ is the positive weight of s. Hence, the larger the weight of a point the quicker its fire wavefront spreads.

Held et al. [12] generalize this concept and introduce the *generalized weighted Voronoi diagram* (GWVD) of a set S of weighted points and variably-weighted straight-line segments: They assign weights to the end-points a and b of an input segment $\overline{ab}$ and then obtain the weight of a point q on $\overline{ab}$ by a linear interpolation of the weights of a and b. Then

$$d_w(p, \overline{ab}) := \min_{q \in \overline{ab}} d_w(p, q),$$

and the corresponding GWVD $\mathcal{VD}_w(S)$ is defined accordingly. See Figure 4 for a sample GWVD and corresponding wavefronts, i.e., variable-radius offsets. We note that even simple differences in the weights may have a significant impact. For instance, the variable-radius offsets shown in Figure 1b were achieved by assigning all other end-points twice the weight of the center point.

2 Our Contribution

We start with examining the generalized weighted Voronoi diagram (GWVD) in more detail: We define an explicit distance function between an arbitrary point in the plane and a variably-weighted straight-line segment. Additionally, two different strategies to compute the GWVD are outlined.

Since the individual Voronoi regions of a GWVD may be disconnected, we introduce a closely related structure whose regions stay connected. We call this structure a variable-radius skeleton (VRS), present an algorithm for computing a VRS inside of a polygon, and discuss results obtained by our prototype implementation.

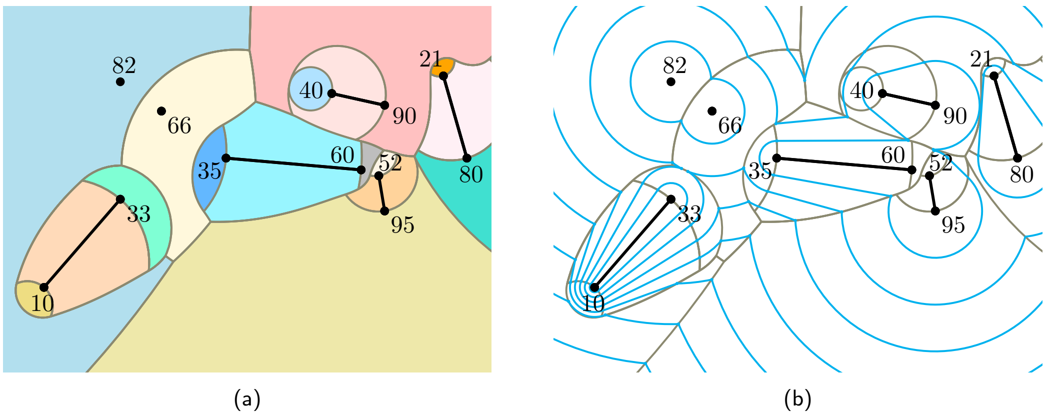


Figure 4: (a) Generalized weighted Voronoi diagram of a set of weighted points and straight-line segments (highlighted in black) and (b) family of corresponding wavefronts. All points have the same positions and weights as in Figure 3a.

3 Preliminaries

Let S be a set of weighted points and variably-weighted straight-line segments. All multiplicative weights are required to be positive. (Biedl et al. [5] show that a weighted skeletal structure may lose virtually all important properties of its unweighted sibling if negative weights are allowed.) No input point is allowed to lie on a line segment, and no pair of line segments may share a point except for a common end-point. All input segments and input points are called *sites*. For the sake of descriptional simplicity, it is assumed that no point in $\mathbb{R}^2$ has the same weighted distance to more than three sites of S.

Every input site $s \in S$ is associated with a so-called *offset circle* $c(s,t)$ which includes all points in $\mathbb{R}^2$ that are at weighted distance t to s. We find it convenient to regard $c(s,t)$ as a function of either time or distance since at time t every point on $c(s,t)$ is at Euclidean distance $t \cdot w(s)$ from s, i.e., at weighted distance t. As discussed in [12], the offset circle $c(\overline{ab},t)$ of a straight-line segment $\overline{ab}$ is formed by two circular arcs, which are induced by its end-points a and b, and two straight-line segments; see Figure 5c. (Of course, the offset circle of a variably-weighted straight-line segment is no genuine circle but we prefer to use the same term for the offsets of both points and straight-line segments.) A similar result holds for the offset circle of a circular arc if identical weights are assigned to its end-points.

The wavefront $\mathcal{W}(S,t)$ emanated by S at time $t \geq 0$ is the set of all points p of the plane whose minimal weighted distance from S equals t. More formally,

$$\mathcal{W}(S,t) := \left\{ p \in \mathbb{R}^2 : \min_{s \in S} d_w(p,s) = t \right\}.$$

Hence, for $t = 0$ the wavefront $\mathcal{W}(S,t)$ equals S. All wavefronts consist of portions of offset circles and, thus, consist only of straight-line segments and circular arcs. Every such *wavefront edge* is associated with its corresponding input site. A common end-point of two adjacent wavefront edges is called a *wavefront vertex*. Similar to standard Voronoi diagrams and straight skeletons [16], the wavefront vertices will trace out the edges of our skeletal structures. That is, they move along the bisectors of pairs of input sites (relative to the weighted distance).

Consider two sites $s_1, s_2 \in S$ and assume that their offset circles intersect. If s_1 and s_2 both are weighted points then their offset circles intersect in exactly one pair of points, which we call *moving intersections*. (These

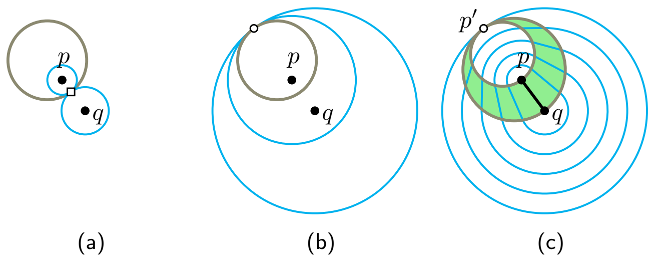


Figure 5: In (a) and (b) the collision and domination points, respectively, of two offset circles that are emanated by two weighted points p and q are displayed for $w(p) < w(q)$. In (c) p and q are the end-points of a variably-weighted straight-line segment $\overline{pq}$. The singular point p' of $\overline{pq}$ is indicated by a small black circle. The circular boundaries of the corresponding areas of influence are shown in dark green, and $\mathcal{AOI}(\widetilde{pq})$ is shaded in light green.

points may coincide, though.) Otherwise, the offset circles may define up to two pairs of moving intersections. Every such a moving intersection traces out a part of the bisector $b(s_1, s_2)$ of s_1 and s_2. These traces are non-overlapping (except possibly for the respective start- or end-points) and their union equals $b(s_1, s_2)$. We say that $c(s_1, t)$ and $c(s_2, t)$ *collide* at time t if a pair of moving intersections appears for the first time at time t. If a pair of moving intersections disappears at time t then $c(s_1, t)$ *dominates* $c(s_2, t)$ at t; see Figure 5.

To avoid two-dimensional bisectors between two variably-weighted straight-line segments that share a common end-point, we adapt Held's concept of *areas of influence* [11]: Every variably-weighted straight-line segment $\overline{pq}$ induces a subdivision of the plane into three areas which we denote by $\mathcal{AOI}(p)$, $\mathcal{AOI}(q)$, and $\mathcal{AOI}(\widetilde{pq})$, respectively, where $\widetilde{pq}$ is the open straight-line segment between p and q, i.e., $\overline{pq}$ without its end-points. Each area of influence includes the points of $\mathbb{R}^2$ that are closest to the corresponding portion of $\overline{pq}$. If $w(p) < w(q)$ then the areas of influence are bounded by two circles that touch in a single point p' which we refer to as *singular point*; see Figure 5c. Note that p' coincides with the point at which q starts to dominate p. If $w(p) = w(q)$ then the areas of influence are bounded by two lines that are perpendicular to $\overline{pq}$ and run through p and q.

4 Generalized Weighted Voronoi Diagrams

Algorithmic paradigms such as plane sweep [6] and incremental construction [15], which have been used extensively for computing Voronoi diagrams of unweighted as well as constantly weighted input sites, seem inapplicable for computing GWVDs. This is due to two main reasons. First of all, the individual Voronoi regions of a GWVD are (in general) not connected; see also Figure 6. Furthermore, it is (in general) not possible to establish an insertion order $(s_1, s_2, \ldots, s_n)$ of the elements of S such that the Voronoi region $\mathcal{VR}_w(s_i, S_i)$ of the i-th site s_i in the Voronoi diagram $\mathcal{VD}_w(S_i)$ of the i-th prefix set $S_i := (s_1, s_2, \ldots, s_i)$ stays connected for all $i \in \{1, 2, \ldots, n\}$. Therefore, we focus on two avenues for computing GWVDs that are more resilient to the inherent properties of this structure.

Edelsbrunner and Seidel [7] establish the connection between Voronoi diagrams in $\mathbb{R}^d$ and lower envelopes in $\mathbb{R}^{d+1}$. Agarwal et al. [3] present an algorithm for computing the lower envelope of a set of n algebraic bivariate functions in $\mathcal{O}(n^{2+\varepsilon})$ time[1], for any $\varepsilon > 0$. In order to employ this result we need to define an algebraic bivariate function that models the distance of a point to a specific input site, i.e., to a weighted point or straight-line segment. If $0 < w(a) = w(b)$ then the distance from a point $p \in \mathbb{R}^2$ to the straight-line

[1]A term of the form $k + \varepsilon$ means that there is some additive positive constant ε that needs to be added to k that may be regarded as arbitrarily small but it will never equal zero.

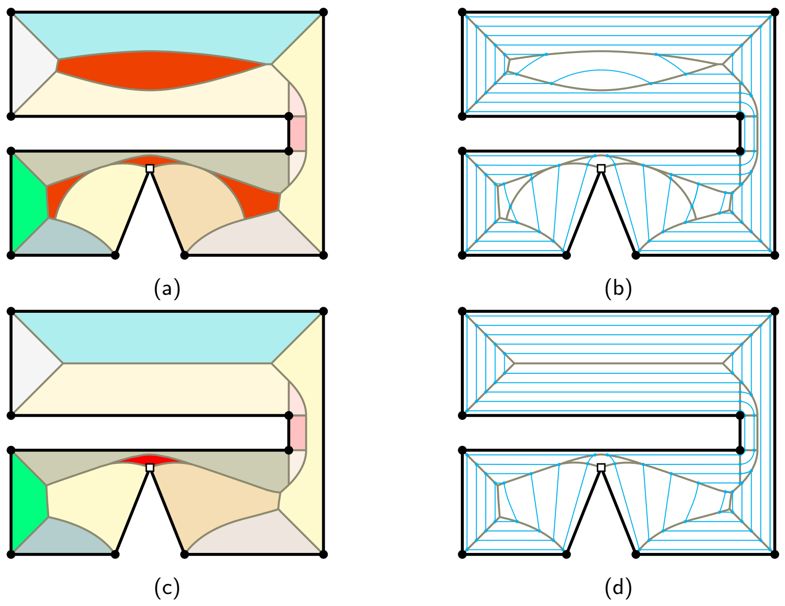


Figure 6: (a) GWVD and (b) corresponding wavefronts inside a polygon P where the node indicated by a square has three times the weight of all other nodes. Note that the Voronoi region of the "square" node consists of four connected components shown in red. (c) VRS and (d) corresponding wavefronts.

segment $\overline{ab}$ is simply given by $d(p,\overline{ab})/w(a)$. Otherwise, assume that $0 < w(a) < w(b)$. For the sake of mathematical simplicity we further assume that $\overline{ab}$ lies on the positive x-axis such that the singular point a' coincides with the coordinate origin. The closest weighted point $\psi_p\left(\overline{ab}\right)$ of p on $\overline{ab}$ is given as

$$\psi_p\left(\overline{ab}\right) := \begin{cases} a & p \in \mathcal{AOI}(a), \\ b & p \in \mathcal{AOI}(b), \\ \left(\frac{x^2+y^2}{x}, 0\right) & \text{otherwise.} \end{cases}$$

This gives us

$$d_w(p,\overline{ab}) := d_w(p, \psi_p\left(\overline{ab}\right))$$

as the weighted distance $d_w(p,\overline{ab})$ between p and $\overline{ab}$. Of course, every weighted straight-line segment can be aligned easily with the x-axis such that its singular point coincides with the origin. Thus, every input site is associated with an algebraic bivariate distance function. Held et al. [12] argue that the graph of such a distance function is a sub-surface of a right conoid.

Lemma 4.1 summarizes the fact that the general-purpose strategy of Agarwal et al. [3] can be utilized to compute GWVDs. Lemma 4.2 and Lemma 4.3 establish upper and lower bounds, respectively, on the combinatorial complexity of a GWVD in the worst case.

Lemma 4.1. The GWVD $\mathcal{VD}_w(S)$ of a set S of n weighted points and variably-weighted straight-line segments as input sites can be computed in $\mathcal{O}(n^{2+\varepsilon})$ time, for any $\varepsilon > 0$.

Lemma 4.2. The GWVD $\mathcal{VD}_w(S)$ of a set S of n input sites has a combinatorial complexity of $\mathcal{O}(n^{2+\varepsilon})$ in the worst case, for any $\varepsilon > 0$.

Proof. This bound is derived from the combinatorial complexity of the corresponding lower envelope. □

Lemma 4.3. The GWVD $\mathcal{VD}_w(S)$ of a set S of n input sites has a combinatorial complexity of $\Omega(n^2)$ in the worst case.

Proof. The multiplicatively weighted Voronoi diagram of points has a combinatorial complexity of $\Omega(n^2)$ in the worst case [4]. □

Even though lifting a problem to higher dimensions can be enlightening from a theoretical point of view, experience tells us that it often complicates matters when it comes to developing a practical implementation. And, indeed, the proof-of-concept implementation by Held et al. [12] that is based on this approach cannot go beyond tiny input sets. (And this limitation would hardly go away if their code were tuned for speed.)

Hence, we discuss an alternative wavefront-based strategy that operates entirely in the plane. Recall that the wavefront $\mathcal{W}(S,t)$ equals S for $t = 0$. Starting at $t = 0$, the expansion of the wavefront $\mathcal{W}(S,t)$ is simulated by continuously increasing the time (or weighted distance) t. We do already know that $\mathcal{W}(S,t)$ consists of (possibly multiple) closed curvilinear chains, for all $t \geq 0$. If all segments of S are disjoint then, for a sufficiently small time t_0, the wavefront $\mathcal{W}(S,t_0)$ consists of exactly two straight-line segments per input segment and of up to one circular arc per input point or end-point of a segment.

It is obvious that the vertices of $\mathcal{W}(S,t)$ change their positions as t is increased. During the wavefront propagation process, we keep track of these moving intersections. However, a change of the locations of the wavefront vertices is not the only change that will happen. Rather, new wavefront edges may appear, old wavefront edges may disappear, and the wavefront may split into two or more connected wavefront components. These combinatorial changes are witnessed by the following three types of events.

Definition 4.1 (Collision event)**.** A *collision event* occurs whenever a new pair of moving intersections appears.

Definition 4.2 (Domination event)**.** A *domination event* occurs whenever a pair of moving intersections disappears.

Definition 4.3 (Arc event)**.** An *arc event* occurs whenever two moving intersections coincide along an offset circle in such a way that this constellation can be neither categorized as a collision nor as a domination event.

An arc event may cause a wavefront edge to shrink to zero length and, thus, to disappear. As in the case of standard Voronoi diagrams, a new Voronoi node has been discovered whenever an old wavefront edge disappears or a new wavefront edge appears.

Every offset circle $c(s,t)$ holds the set of moving intersections at time t sorted in counter-clockwise angular order around the respective site in a self-balancing binary search-tree. In particular, it stores those portions of $c(s,t)$ which overlap with $\mathcal{W}(S,t)$. Furthermore, every moving intersection holds a flag that indicates whether it is a wavefront vertex. Initially, all collision and domination events are computed and inserted into a priority queue $\mathcal{Q}$. All offset circles are initially empty. Afterwards, all events are successively retrieved from $\mathcal{Q}$.

- If a collision or domination event occurs at time t_σ then a pair of moving intersections is inserted or removed, respectively, from their corresponding offset circles. If a collision event takes place on $\mathcal{W}(S,t_\sigma)$ then the newly created moving intersections are marked to be wavefront vertices.
- If an arc event takes place then three offset circles coincide at a single point. A Voronoi node has been discovered if at least one of the respective moving intersections has been a wavefront vertex.

Common to all these events is the necessity to compute and store a future arc event whenever two moving intersections become neighbors along an offset circle. The wavefront propagation is active until the highest-weighted input point dominates all other input sites. If multiple sites have the same maximum weight then $\mathcal{Q}$ can only be empty once $\mathcal{W}(S,t)$ contains only one loop of wavefront segments which all lie on offset circles of these sites and if all wavefront vertices move along rays to infinity.

Theorem 4.1. Wavefront propagation allows to compute $\mathcal{VD}_w(S)$ in $\mathcal{O}(n^3 \log n)$ time and $\mathcal{O}(n^3)$ space.

Proof. Every pair of input sites defines at most constantly many collision and domination events and at most $\mathcal{O}(n^3)$ arc events may take place. Each of these events consumes $\mathcal{O}(\log n)$ time, since every event requires a constant number of lookups, insertions, and/or deletions in a self-balancing binary search tree of size $\mathcal{O}(n)$ or in a priority queue of size $\mathcal{O}(n^3)$. □

We emphasize that the bound $\mathcal{O}(n^3)$ on the number of arc events is a trivial upper bound given by the fact that every triple of offset circles can define only a constant number of arc events. Unfortunately, no sharper bound on the number of arc events is known even just for weighted point sites.

5 Variable-Radius Skeleton

Assume that the weighted points and straight-line segments of S form the vertices and edges of a polygon P. We call a vertex v *reflex* (*convex*) if the interior angle at v is greater (smaller, resp.) than 180°. (For the sake of descriptional simplicity we assume that no interior angle equals 180°.) Recall that a (positive) weight $w(v)$ is assigned to every vertex v. A vertex is *dominant* if its weight is greater than the weight of one of its two neighbors.

Let A denote (the closure of) the area bounded by P. We now explain how we obtain a Voronoi-like structure within A that we named variable-radius skeleton (VRS), $\mathcal{S}_w(P)$; see Figure 6. Similar to a Voronoi diagram, it subdivides the interior of P into several *skeletal regions*, with at most one region $\mathcal{SR}_w(s, S)$ per edge s or vertex s, for all $s \in S$. However, unlike the GWVD, which may be defined based on an explicit distance function, we rely on a purely wavefront-based definition to specify the VRS such that the following two properties hold:

- For every $s \in S$ its skeletal region $\mathcal{SR}_w(s, S)$ is connected (or empty).
- A point p lies in $\mathcal{SR}_w(s, S)$ if and only if $s \in S$ is the closest site for which there exists a path γ within A from p to s such that the weighted distance decreases strictly monotonically as one moves from p to s along γ.

We will be using a slightly different type of wavefront than in the case of the GWVD. To make this distinction more clear we refer to this new wavefront as the *extinguishing wavefront* $\mathcal{EW}(P, t)$ of P at time t. For $t := 0$ we have $\mathcal{EW}(P, t) = P$, i.e., the extinguishing wavefront coincides with P. For a sufficiently small time t_0, a counter-clockwise traversal of $\mathcal{EW}(P, t_0)$ will match a counter-clockwise traversal of P such that every straight-line segment of $\mathcal{EW}(P, t_0)$ corresponds to exactly one edge of P and every circular arc of $\mathcal{EW}(P, t_0)$ corresponds to exactly one reflex vertex of P, and vice versa.

As in the case of a GWVD, the expansion of the wavefront $\mathcal{EW}(P, t)$ is simulated by continuously increasing the time (or weighted distance) t. We know that wavefronts are formed by (possibly several) closed curvilinear chains that consist of straight-line segments and circular arcs as wavefront edges. The key difference to the standard wavefront propagation is given by the fact that parts of the wavefront extinguish each other once they collide. In the prairie fire analogy, this means that even a rapidly expanding fire front stops expanding once it reaches an area of A that had already been burnt by another fire. The combinatorial changes that $\mathcal{EW}(P, t)$ can undergo are witnessed by the following three types of events; see Figure 7.

Definition 5.1 (Split event). A *split event* occurs at time t if an arc of $\mathcal{EW}(P, t)$ collides with another arc or a segment of $\mathcal{EW}(P, t)$, thereby splitting $\mathcal{EW}(P, t')$ into two wavefront components for $t' > t$.

Definition 5.2 (Edge event). An *edge event* occurs at time t if the two end-points of an edge e of $\mathcal{EW}(P, t)$ coincide in $\mathcal{EW}(P, t)$, thus causing e to disappear and to be absent from $\mathcal{EW}(P, t')$ for $t' > t$.

Definition 5.3 (Break-through event)**.** A *break-through event* occurs at time t if, for a dominant convex vertex v_i of P, the wavefront edges induced by the polygon edges $\overline{v_{i-1}v_i}$ and $\overline{v_iv_{i+1}}$ are collinear at time t. In this case $\mathcal{EW}(P,t')$ will contain a circular arc centered at v_i for $t' > t$.

That is, a break-through event witnesses the birth of a new skeletal region and causes a new circular arc to appear on the wavefront. Elementary mathematics shows that a break-through event can occur at most once per convex vertex. An edge event may occur as one offset circle dominates a second one. The locations of edge events and break-through events coincide with nodes of $\mathcal{S}_w(P)$. At an edge event one of the faces induced by $\mathcal{S}_w(P)$ is closed.

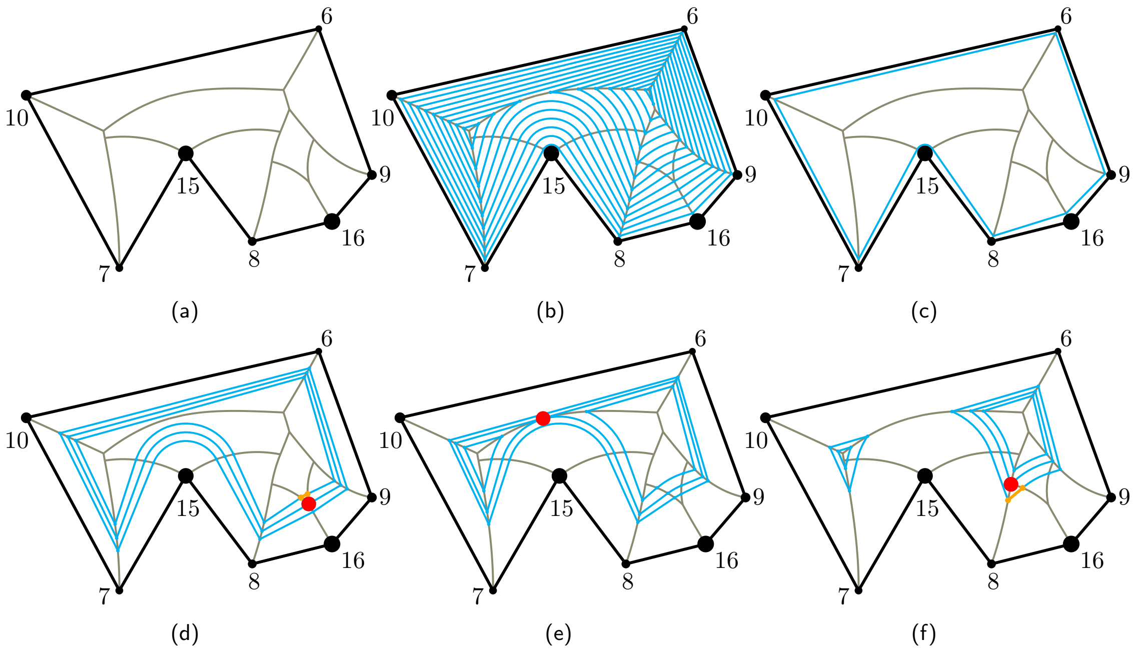


Figure 7: (a) Sample variable-radius skeleton (VRS) of a polygon and (b) a family of wavefronts. The numbers next to the vertices of the polygon indicate their weights. (c) Initial wavefront; (d) break-through event; (e) split event; (f) edge event. The location of each event is marked by a red dot, and the wavefront edge that appears or disappears is drawn in orange. The radii of the disks centered at the polygon vertices are directly proportional to the weights that are associated with them. Note that after the split event that is depicted in (e) the wavefront consists of two separate wavefront components.

This event-based description of the VRS already implies a simple construction strategy. All events are stored in a priority queue $\mathcal{Q}$ ordered by the times of their occurrences. Every wavefront component is a curvilinear chain oriented counter-clockwise. It is represented by a doubly-linked list. Every wavefront edge holds a pointer to its predecessor and its successor in the corresponding wavefront component. Additionally, every input site stores the wavefront edges which it is associated with in a self-balancing binary search-tree in counter-clockwise order.

Initially, the times of all potential split events as well as all break-through events are computed and pushed to $\mathcal{Q}$. Furthermore, for every wavefront segment that shrinks to zero length an edge event is inserted into $\mathcal{Q}$.

After the initialization phase is completed the individual events are successively popped from $\mathcal{Q}$. Assume that σ is the current event that takes place at time $t_\sigma > 0$.

- If σ is a split event then we determine the two wavefront segments e_1 and e_2 along which the event point p_σ is situated and split the corresponding wavefront component accordingly. Thus, a split event involves constantly many pointer manipulations and insertions/deletions in the corresponding self-balancing binary search-trees.
- If σ is an edge event then we remove the corresponding wavefront segment from its respective wavefront component. Special precautions have to be taken whenever a wavefront segment disappears since one offset circle $c(s_2, t)$ dominates another offset circle $c(s_1, t)$ and only one of the disappearing moving intersections is currently a wavefront vertex; see Figure 8. In this case, we know that $w(s_1) < w(s_2)$. We boost the weight of s_1 to $w(s_2)$ and continue the wavefront propagation. Hence, we will refer to such an edge event as a *boost event*. (This strategy is inherited from a similar situation that can occur for weighted straight skeletons [9].) In order to make sure that we will not miss a split event we have to compute all collisions of the offset circle of s_1 with the other wavefront edges of its wavefront component.
- Otherwise, σ is a break-through event. We insert a new wavefront edge e at p_σ into the respective wavefront component. Again, we check whether e may have future collisions wavefront edges of its wavefront component. In such a case we insert the corresponding split event into $\mathcal{Q}$.

Common to all these events is the need to watch for edge events: Whenever two wavefront vertices become neighbors for the first time we check whether they will coincide at a future point in time. In such a case we insert the corresponding edge event into $\mathcal{Q}$.

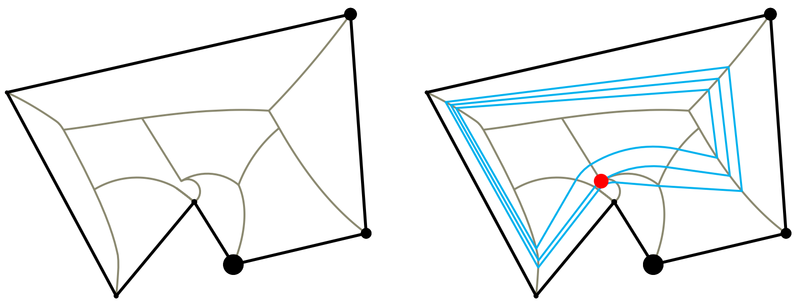


Figure 8: A boost event takes place along the wavefront (marked by the red dot).

The wavefront propagation terminates once all wavefront components have shrunk to points and, thus, vanished. Recall that the skeletal region of a site s is swept by portions of the offset circle of s. An inductive argument over all event times shows that every skeletal region does indeed have the desired properties. In particular, it is connected. With minor modifications the wavefront propagation used to compute a VRS of weighted points and straight-line segments can be extended to any set of weighted points, weighted straight-line segments and constantly-weighted circular arcs as input sites: We only need to demand that no pair of input sites intersects in a point other than a common end-point.

The attentive reader may wonder why we do not need to deal with break-through and boost events also in the case of GWVDs. After all, the input sites that define a GWVD could also represent the vertices and edges of a polygon. The key difference is given by the fact that a GWVD is computed within the entire plane. This implies that a boost event during the construction of a VRS would manifest itself as a domination event in the case of GWVDs, which causes the wavefront arc whose weight is boosted in the VRS to disappear completely from the wavefront of the GWVD. During a break-through event three moving intersections coincide in a single point. Thus, all break-through events are implicitly dealt with at arc events during the construction of the GWVD.

Lemma 5.1. Let P be a polygon with n vertices. Then VRS $\mathcal{S}_w(P)$ has a combinatorial complexity of $\mathcal{O}(n)$ in the worst case.

Proof. Recall that we have exactly one skeletal region for each straight-line segment and for each reflex vertex, and at most one region for each convex vertex. Hence, the number of skeletal region is linear in n. Furthermore, $\mathcal{S}_w(P)$ is a plane graph where every node that does not coincide with a vertex of P has degree (at least) three. Euler's formula for planar graphs implies that also the number of nodes and edges of $\mathcal{S}_w(P)$ is linear in n. □

Theorem 5.1. Wavefront propagation allows to compute the variable-radius skeleton of a polygon P with n vertices in worst-case time $\mathcal{O}(nr^2 + nr \log n)$ and $\mathcal{O}(nr)$ space, where r is the total number of reflex and dominant vertices of P.

Proof. Initially, $\mathcal{O}(nr)$ split events have to be computed. Additionally, $\mathcal{O}(r)$ break-through events may take place in the worst case. Every break-through event takes $\mathcal{O}(n)$ time since we need another round of checks for future split events. During every split and break-through event constantly many wavefront edges are generated that disappear at subsequent edge events. Up to $\mathcal{O}(r^2)$ boost events can occur, with one event consuming $\mathcal{O}(n)$ time due to a search for future split events. All other events consume at most $\mathcal{O}(\log n)$ time, since they require a constant number of lookups, insertions, and/or deletions in a self-balancing binary search tree of size $\mathcal{O}(n)$ or in a priority queue of size $\mathcal{O}(nr)$. □

6 Implementation and Discussion

We have been working on a prototype implementation of the wavefront-based construction strategy for variable-radius skeletons in C++. Our implementation operates entirely in two dimensions and is based on conventional IEEE 754 floating-point arithmetic. The skeletons and offsets shown in Figure 11 and the other figures of this publication were generated by means of our implementation. We performed runtime tests on approximately 600 weighted polygons with $n \in \{16, 32, 64, \ldots, 8192\}$ vertices; see Figure 9. (Larger polygons could not be tested due to memory constraints.) All weights were chosen randomly in which we allowed the highest weight to be at most ten-times as big as the lowest one. The polygons were taken from real-world input provided by companies and from the Salzburg Database of polygonal data [8]. In our tests the number r of reflex and dominant vertices averaged about $0.8n$. All tests were carried out on an Intel Xeon E5-2687W v4 processor clocked at 3.0 GHz. These tests support our conjecture that the super-quadratic time bound given in Theorem 5.1 is far too pessimistic for practical applications.

However, the bound $O(r^2)$ on the number of boost events is sharp in the worst case: One can specify a polygon and select appropriate vertex weights such that a cascade of boost events occurs, where the weights of $i-1$ circular arcs of the wavefront get boosted during the i-th boost event. Such a setting is highly contrived, though! Our test runs let us conclude that one is more likely to see no boost event at all than to see even just a few such events during one wavefront propagation.

Additionally, we performed several sample test runs in which we bounded the difference between the highest as well as the lowest weighted vertex by a small constant factor, or assigned gradually increasing weights to the vertices we encountered whilst walking along the boundary of the respective input polygon; see Figure 10. Our tests indicate that these special weight-assignment strategies have no significant impact on the overall performance of our implementation (compared to our standard weight-assignment strategy). In particular, the respective runtimes were well within the fluctuation range that we encountered during our standard test runs.

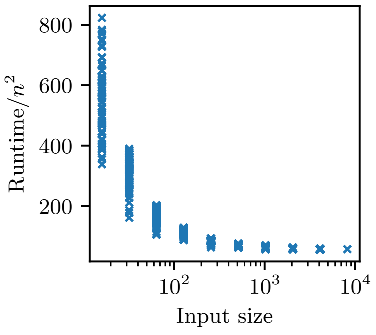


Figure 9: The plot shows the overall runtime in microseconds divided by n^2. Each marker on the x-axis indicates the number n of input vertices for one out of roughly 600 test polygons.

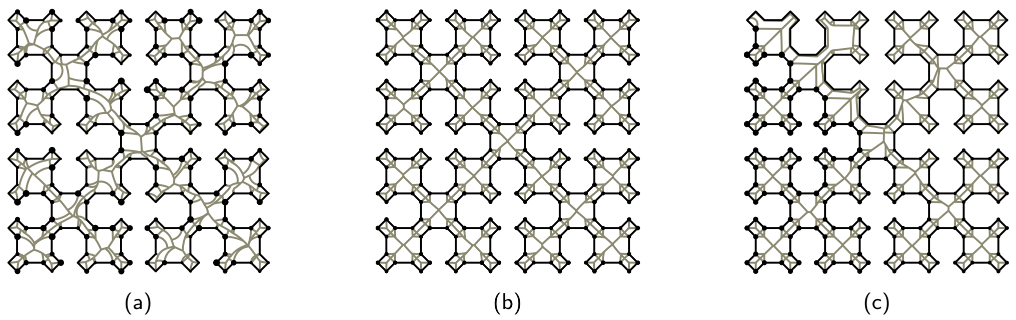


Figure 10: Three VRSs inside differently weighted instances of a Sierpinski curve are shown. In (a) and (b) the corresponding vertex weights have been chosen uniformly at random in which the highest and lowest weight are within a factor of 10 (for (a)) and 1.2 (for (b)) of each other. In (c) the vertex weights gradually increase as we walk along the boundary of the input polygon.

The high algebraic degree of the individual bisectors turns the reliable determination of edge events into a delicate problem on conventional floating-point arithmetic. It turned out to be particularly difficult to deal with multiple events that happen at almost the same location. Experience drawn from several industrial-strength implementations of geometric codes carried out within the first author's group during the last 30 years allowed us to mitigate the numerical problems. Still, we have seen our code produce obviously incorrect results due to numerical stability problems. Fortunately, random sampling of a few nodes of a VRS followed by distance computations allows to detect incorrect structures fairly reliably. (And incorrect offsets are also spotted easily by a human by simple visual inspection.) A potential solution to this problem would be to utilize exact arithmetic, e.g., by using the Computational Geometry Algorithms Library (CGAL) [2]. However, exact arithmetic has a hefty computational price tag. In particular, comparisons of event times can no longer be assumed to take (close to) constant time [9]. (Note that we would require the use of constructors and could not resort to a realization based entirely on predicates.)

From an algorithmic point of view a natural avenue for future research is to try to reduce the number of collision/split events computed. Experiments quickly show that many of the quadratically many split events that are computed a-priori are irrelevant. That is, they have no influence on the subsequent wavefront propagation. Thus, it would greatly improve the practical performance of our strategy if we were able to filter out at least some of the unnecessary event computations beforehand. This remark is particularly true if one would resort to exact arithmetic to schedule all events.

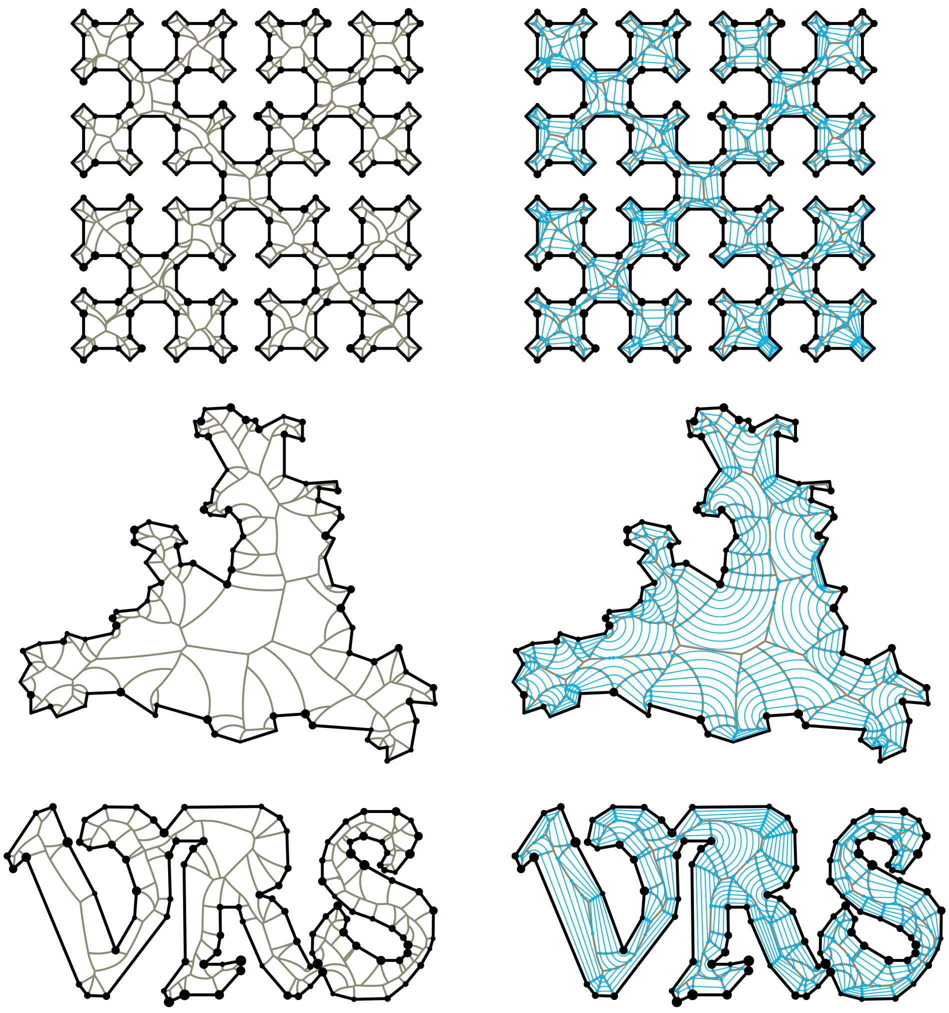


Figure 11: Variable-radius skeletons and families of variable-radius offsets.

7 Constructing Roofs

Similar to other skeletal structures [13], it is also possible to derive three-dimensional terrains from a VRS that can be used as intricate roofs over polygonal footprints of buildings. For a polygonal footprint P, a point p of

$\mathcal{S}_w(P)$, with coordinates (p_x, p_y), is lifted to a point in $\mathbb{R}^3$ with coordinates $(p_x, p_y, d_w(p, s))$. Alternatively, we can lift a wavefront $\mathcal{EW}(P, t)$ for the weighted distance t to z-coordinate t: We get $\mathcal{EW}(P, t) \times \{t\}$.

We refer to such a roof as a *variable-radius roof* and note that it is guaranteed to drain water. (That is, it does not contain local minima that form sinks in which water would accumulate.) The individual faces of a variable-radius roof are given by parts of conics and by ruled surfaces. Figure 12 illustrates examples of such roofs for some of the polygons and skeletons shown in Figure 11.

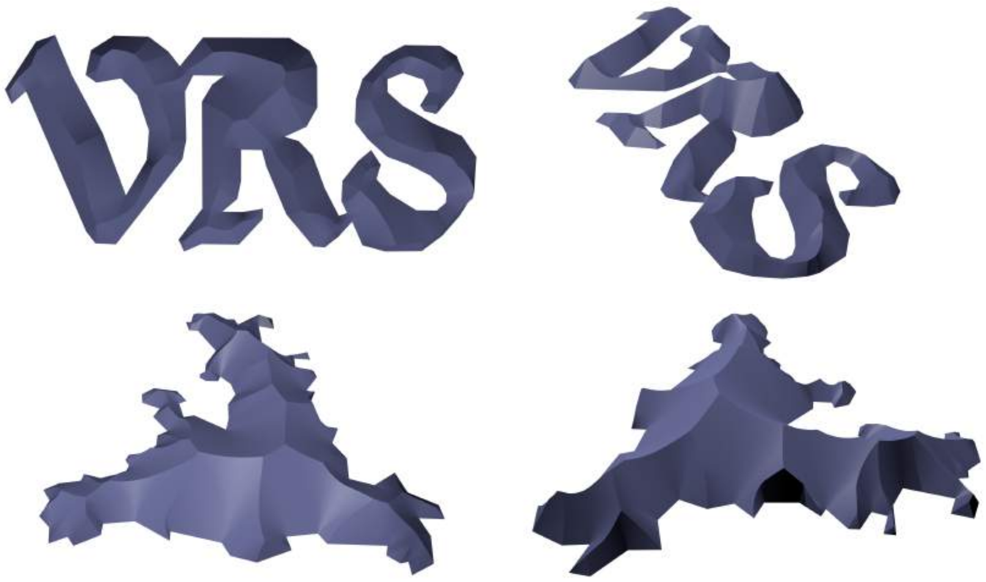


Figure 12: Two different views of sample variable-radius roofs. The roof data was generated by our implementation and then rendered by means of Blender [1].

8 Conclusion

We extend the work by Held et al. [12] and present a wavefront-based construction strategy for generalized weighted Voronoi diagrams. The obvious practical problem of a GWVD is that a Voronoi region may consist of several connected components even if the edges of S form a polygon P. Therefore, the GWVD of a polygon P may have a quadratic combinatorial complexity in the worst case. If such a structure is used to generate ornamental seams then we would get seam curves at locations inside of P where one would not expect to see them. Similarly, the faces of variable-radius roofs would hardly match one's intuition. To remedy this shortcoming, we introduce the variable-radius skeleton inside a polygon P. It allows to locally control the spacing of consecutive offset curves by associating multiplicative weights with the vertices of P while still maintaining connected skeletal regions. Our implementation shows the complexity bounds derived for the computation of a variable-radius skeleton tend to be too pessimistic in practical applications. An extension of our strategy to three dimensions would be possible from a purely theoretical point of view. However, an actual implemention of such an extension should be assumed to be difficult in practice.

Acknowledgements

Work supported by Austrian Science Fund (FWF): Grant P31013-N31.

ORCID

Martin Held, http://orcid.org/0000-0003-0728-7545
Stefan de Lorenzo, http://orcid.org/0000-0003-4981-805X

REFERENCES

[1] Blender. http://www.blender.org/.

[2] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org/.

[3] Agarwal, P.K.; Schwarzkopf, O.; Sharir, M.: The Overlay of Lower Envelopes and its Applications. Discrete & Comp. Geom., 15(1), 1–13, 1996. http://doi.org/10.1007/BF02716576.

[4] Aurenhammer, F.; Edelsbrunner, H.: An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane. Pattern Recognition, 17(2), 251–257, 1984. http://doi.org/10.1016/0031-3203(84)90064-5.

[5] Biedl, T.; Held, M.; Huber, S.; Kaaser, D.; Palfrader, P.: Weighted Straight Skeletons in the Plane. Comput. Geom. Theory and Appl., 48(2), 120–133, 2015. http://doi.org/10.1016/j.comgeo.2014.08.006.

[6] Dehne, F.; Klein, R.: "The Big Sweep": On the Power of the Wavefront Approach to Voronoi Diagrams. Algorithmica, 17(1), 19–32, 1997. http://doi.org/10.1007/BF02523236.

[7] Edelsbrunner, H.; Seidel, R.: Voronoi Diagrams and Arrangements. Discrete & Comp. Geom., 1(1), 25-44, 1986. http://doi.org/10.1007/BF02187681.

[8] Eder, G.; Held, M.; Jasonarson, S.; Mayer, P.; Palfrader, P.: On Generating Polygons: Introducing the Salzburg Database. In Proc. 36th Europ. Workshop Comput. Geom., 75:1–75:7, 2020.

[9] Eder, G.; Held, M.; Palfrader, P.: On Implementing Straight Skeletons: Challenges and Experiences. In Proc. 36th Int. Sympos. Comput. Geom. (SoCG'20), 38:1–38:16, 2020.

[10] Held, M.: Voronoi Diagrams and Offset Curves of Curvilinear Polygons. Comput. Aided Design, 30(4), 287–300, 1998. http://doi.org/10.1016/S0010-4485(97)00071-7.

[11] Held, M.: VRONI: An Engineering Approach to the Reliable and Efficient Computation of Voronoi Diagrams of Points and Line Segments. Comput. Geom. Theory and Appl., 18(2), 95–123, 2001. http://doi.org/10.1016/S0925-7721(01)00003-7.

[12] Held, M.; Huber, S.; Palfrader, P.: Generalized Offsetting of Planar Structures using Skeletons. Comput. Aided Design & Appl., 13(5), 712–721, 2016. http://doi.org/10.1080/16864360.2016.1150718.

[13] Held, M.; Palfrader, P.: Skeletal Structures for Modeling Generalized Chamfers and Fillets in the Presence of Complex Miters. Comput. Aided Design & Appl., 16(4), 620–627, 2019. http://doi.org/10.14733/cadaps.2019.620-627.

[14] Klein, R.; Langetepe, E.; Nilforoushan, Z.: Abstract Voronoi Diagrams Revisited. Comput. Geom. Theory and Appl., 42(9), 885–902, 2009. http://doi.org/10.1016/j.comgeo.2009.03.002.

[15] Klein, R.; Mehlhorn, K.; Meiser, S.: Randomized Incremental Construction of Abstract Voronoi Diagrams. Comput. Geom. Theory and Appl., 3(3), 157–184, 1993. http://doi.org/10.1016/0925-7721(93)90033-3.

[16] Palfrader, P.; Held, M.; Huber, S.: On Computing Straight Skeletons by Means of Kinetic Triangulations. In Proc. 20th Annu. Europ. Symp. Algorithms (ESA'12), 766–777, 2012. http://doi.org/10.1007/978-3-642-33090-2_66.

On the Recognition and Reconstruction of Weighted Voronoi Diagrams and Bisector Graphs

Günther Eder, Martin Held, Stefan de Lorenzo, and Peter Palfrader [Ede+21]

Submitted to:
Computational Geometry: Theory and Applications

Feb. 2021

[Ede+21]

Günther Eder, Martin Held, Stefan de Lorenzo, and Peter Palfrader. "On the Recognition and Reconstruction of Weighted Voronoi Diagrams and Bisector Graphs". Submitted to *Computational Geometry: Theory and Applications*. Feb. 2021

On the Recognition and Reconstruction of Weighted Voronoi Diagrams and Bisector Graphs

Günther Eder[a,*], Martin Held[a,*], Stefan de Lorenzo[a,*], Peter Palfrader[a,*]

[a] *Universität Salzburg, Computerwissenschaften, Salzburg, Austria*

Abstract

A weighted bisector graph is a geometric graph whose faces are bounded by arcs that are portions of multiplicatively weighted bisectors of pairs of (point) sites such that each of its faces is defined by exactly one site. A prominent example of a bisector graph is the multiplicatively weighted Voronoi diagram of a finite set of points which induces a tessellation of the plane into Voronoi faces bounded by circular arcs and straight-line segments. Several algorithms for computing various types of bisector graphs are known. In this paper we reverse the problem: Given a partition $\mathcal{G}$ of the plane into faces, find a set of points and suitable weights such that $\mathcal{G}$ is a bisector graph of the weighted points, if a solution exists. If $\mathcal{G}$ is a graph that is regular of degree three then we can decide in $\mathcal{O}(m)$ time whether it is a bisector graph, where m denotes the combinatorial complexity of $\mathcal{G}$. In the same time we can identify the uniquely defined candidate solution such that $\mathcal{G}$ could be its multiplicatively weighted Voronoi diagram. Additionally, we show that it is possible to recognize $\mathcal{G}$ as a multiplicatively weighted Voronoi diagram and find all possible solutions in $\mathcal{O}(m \log m)$ time if $\mathcal{G}$ is given by a set of disconnected lines and circles.

Keywords: Voronoi diagram, bisector graph, recognition, reconstruction, multiplicative weight

*Corresponding author

Email addresses: `geder@cs.sbg.ac.at` (Günther Eder), `held@cs.sbg.ac.at` (Martin Held), `slorenzo@cs.sbg.ac.at` (Stefan de Lorenzo), `palfrader@cs.sbg.ac.at` (Peter Palfrader)

1. Introduction

1.1. Motivation and Related Work

A *geometric graph* is the fixed embedding of a planar graph in the plane so that its vertices are represented by points and all its arcs belong to some specific family of curves, e.g., straight-line segments and circular arcs. We refer to a geometric graph as a *bisector graph* if there exists a set of input sites such that all its arcs lie on bisectors of pairs of sites. Furthermore, it is required that every face of a bisector graph is defined by exactly one site. This postulation implies that the degree of every vertex of a bisector graph is at least three. Several authors deal with the computation of particular types of weighted bisector graphs and present strategies to construct them efficiently [1, 2, 3].

In this work we focus on bisector graphs that correspond to a set S of weighted points in the plane and study the reverse problem: Given $\mathcal{G}$, a geometric graph that allegedly is a weighted bisector graph, can we recognize $\mathcal{G}$ as such, and if so, can we reconstruct the respective input sites S and weights σ such that the resulting bisector graph is equal to $\mathcal{G}$? Furthermore, we will discuss several settings in which we are even able to recognize $\mathcal{G}$ as a special type of bisector graph that is known as *multiplicatively weighted Voronoi diagram*; see Figures 1 and 2.

Multiplicatively weighted Voronoi diagrams of points in the plane were first introduced by Boots [4]. Aurenhammer and Edelsbrunner [1] present a worst-case optimal algorithm to compute the multiplicatively weighted Voronoi diagram under the Euclidean distance. Algorithms with a decent expected-case complexity are due to Har-Peled and Raichel [2] and Held and de Lorenzo [3]. Har-Peled and Raichel [2] also prove that the expected combinatorial complexity of the multiplicatively weighted Voronoi diagram is bounded by $\mathcal{O}(n \log^2 n)$ if the weights of all input points are chosen randomly. Eder and Held [5] describe an incremental algorithm for constructing the multiplicatively weighted Voronoi diagram under the maximum norm.

Multiplicatively weighted Voronoi diagrams are widely used in wireless communication to model the coverage areas of sensors or transmitters. If the devices are heterogeneous and distance to a device is measured by means of the Euclidean distance weighted by the sensing/transmitting power of the device then the service areas can be modeled as the regions of (multiplicatively weighted) Voronoi diagrams of the device positions. See, e.g., [6, 7, 8].

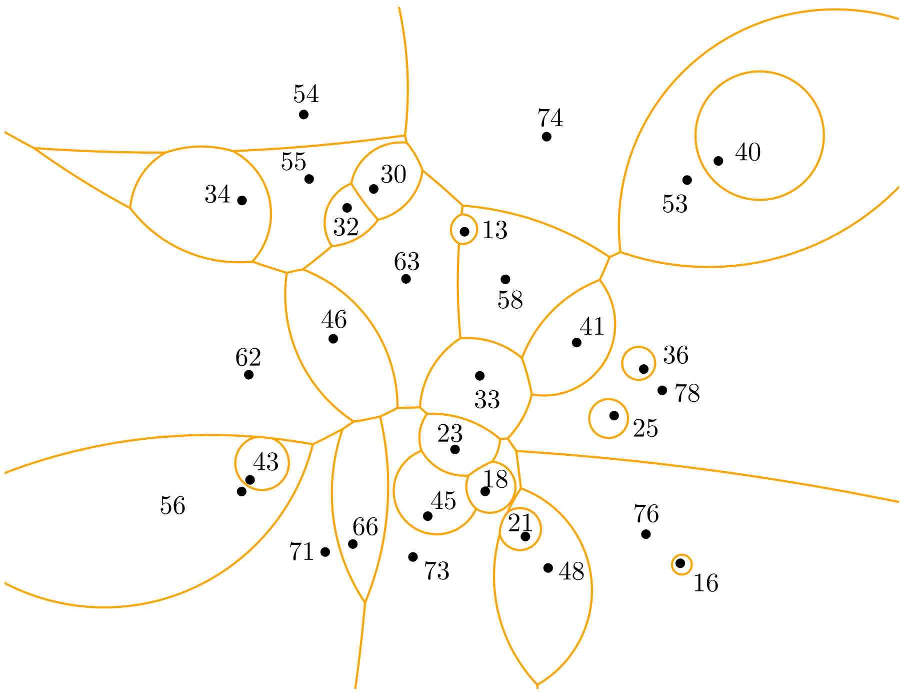


Figure 1: The multiplicatively weighted Voronoi diagram (in orange) of 30 input sites in which the point locations are highlighted by the black dots. The corresponding weights are written next to them.

The problem studied is motivated by a problem forwarded to us by a company working on wireless sensor networks: They get a geometric graph $\mathcal{G}$, a set of sensor positions S and weights σ from an application of one of their customers. The data received is of a low quality, with very low precision of all numerical values, such that a subsequent analysis reveals inconsistencies. That is, the graph $\mathcal{G}$ and the weighted Voronoi diagram of S do not seem to match. Taking only S and σ as input and (re-)computing the corresponding Voronoi diagram is no option since it may be strikingly different from $\mathcal{G}$. This is no surprise because it is known that minor changes in the positions or weights of point sites may change their Voronoi diagram substantially. Hence, the company's next-best idea was to take $\mathcal{G}$ and try to reconstruct S and σ.

Ash and Bolker [9] were among the first to study the recognition problem for unweighted Voronoi diagrams of point sites. Harvingsten [10] presents a polynomial-time algorithm that is based on linear programming, for recogniz-

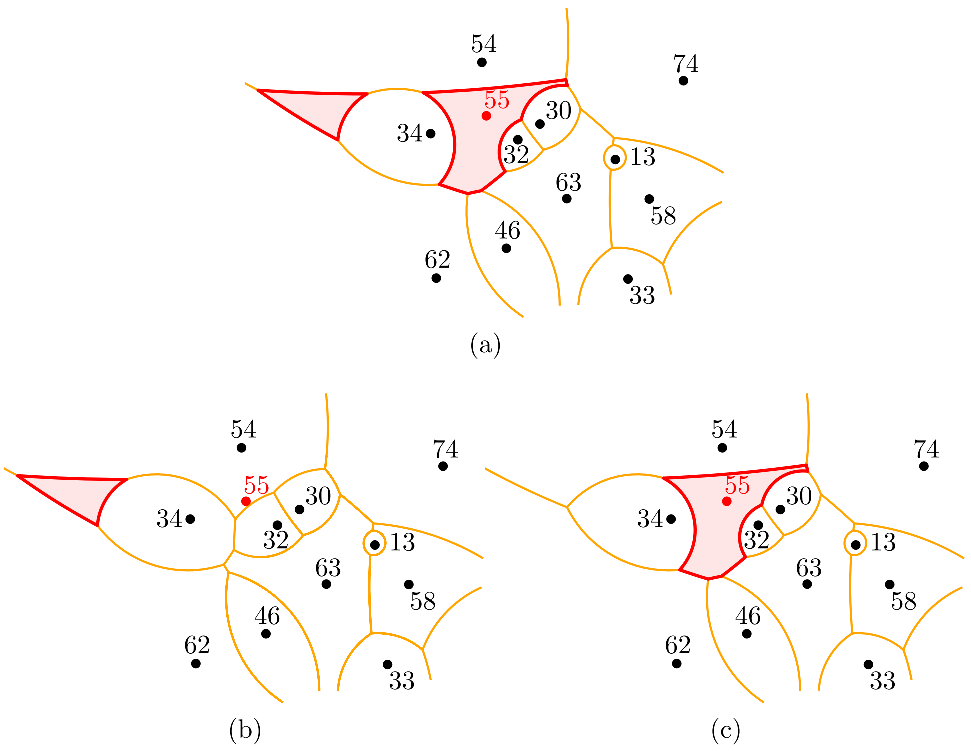


Figure 2: In (a) a section of the multiplicatively weighted Voronoi diagram that is depicted in Figure 1 is shown. The region of the site s that is associated with weight 55 (highlighted in red) consists of two connected components. Furthermore, (b) and (c) show two different bisector graphs in which s is only associated with exactly one connected component.

ing whether a given tessellation of $\mathbb{R}^d$ is an unweighted Voronoi diagram, and reconstructing the respective set of d-dimensional input points. Aurenhammer's work [11] on reciprocal figures and projection polyhedra also allows to characterize and recognize Voronoi diagrams in higher dimensions. Biedl et al. [12] present a strategy for reconstructing the polygon or planar straight-line graph from a given straight-skeleton or Voronoi diagram in $\mathcal{O}(n \log n)$ time, where n is the number of edges of the input graph.

Aichholzer et al. [13, 14] investigate the realizability of a tree as the straight skeleton of a polygon. Eder et al. [15] explain how to reconstruct weighted straight skeletons from geometric trees.

1.2. Preliminaries

Let S be a finite set of n distinct point sites and denote their weight function by $\sigma\colon S \to \mathbb{R}^+$. That is, $\sigma(s)$ specifies the weight of the site $s \in S$. For a point p in the Euclidean plane and a site $s \in S$, the (weighted) distance from p to s is defined as

$$\mathrm{d}_\sigma(p, s) := \frac{d(p, s)}{\sigma(s)},$$

where $d(.,.)$ denotes the standard Euclidean distance. Of course, $\mathrm{d}_\sigma(p, S) := \min\{\mathrm{d}_\sigma(p, s)\colon s \in S\}$. We follow common Voronoi terminology and define the (weighted) Voronoi region of $s \in S$ as the set of all points in $\mathbb{R}^2$ that are not farther from s than from any other site of S with respect to d_σ:

$$\mathcal{R}_\sigma(s, S) := \{p \in \mathbb{R}^2\colon\ \mathrm{d}_\sigma(p, s) \le \mathrm{d}_\sigma(p, S)\}.$$

Then the weighted Voronoi diagram $\mathcal{VD}_\sigma(S)$ of S relative to σ is the union of all region boundaries. (In the sequel, we will simplify the terminology by dropping the term "weighted" and simply refer to $\mathcal{VD}_\sigma(S)$ as Voronoi diagram of S.) Note that $\mathcal{VD}_\sigma(S)$ may have a quadratic combinatorial complexity [1].

Consider a planar geometric graph $\mathcal{G}$ embedded in $\mathbb{R}^2$ whose edges are formed by circular arcs. These arcs are allowed to intersect only at common end-points that we refer to as *nodes* of $\mathcal{G}$. Every arc of $\mathcal{G}$ forms either a full circle (or degenerates to a line), or ends at a node of degree at least three. We call such a graph a *planar circular-arc graph*. The number of faces of the planar subdivision induced by $\mathcal{G}$ is denoted by m. Euler's Theorem for planar graphs implies that $\mathcal{G}$ has $\mathcal{O}(m)$ nodes and $\mathcal{O}(m)$ edges. In the sequel we will use $\mathcal{G}$ as our input that we seek to recognize.

1.3. Our Contribution

Consider a circular-arc graph $\mathcal{G}$ with m faces. If the edges of $\mathcal{G}$ are given by disjoint circles and lines then we can compute all solutions (S, σ) in $\mathcal{O}(m \log m)$ time such that $\mathcal{VD}_\sigma(S)$ equals $\mathcal{G}$. If $\mathcal{G}$ has nodes and if all nodes of $\mathcal{G}$ are of degree three then we can identify in $\mathcal{O}(m)$ time up to two candidate solutions (S, σ) such that $\mathcal{G}$ is a weighted bisector graph of the points of S with weight function σ. We also show that two different inputs $\mathcal{G}$ may yield the same solution set (S, σ). Hence, whether or not $\mathcal{G}$ is an actual Voronoi diagram rather than only a bisector graph for (S, σ) seems difficult to decide without explicitly computing $\mathcal{VD}_\sigma(S)$.

2. Weighted Bisector

For every site s_i of S we define a family of circles $c_i(t)$ centered at s_i with radius $t \cdot \sigma(s_i)$. Then the *(weighted) bisector* $b(i,j)$ between two distinct sites s_i and s_j is given by the trace of the intersection points $c_i(t) \cap c_j(t)$ for $r \in \mathbb{R}^+$:

$$b(i,j) := \{c_i(t) \cap c_j(t) \colon t \in \mathbb{R}^+\}.$$

For the sake of descriptional simplicity, we do not explicitly indicate the dependence of a bisector on σ. And, again, in the sequel we will also drop the term "weighted". It is well-known that the bisector $b(i,j)$ between two sites s_i and s_j forms a circle. (This is easy to see if we recall that ancient Apollonius of Perga showed that a circle is the set of points of a fixed *ratio* of distances to two foci. The two foci in this case are the two input sites, and their bisector is the Apollonian circle which traces out the ratio of their two weights.) Furthermore, s_i and s_j lie on a ray that originates at the center of that circle, with one of them on each side of the circle. Aurenhammer and Edelsbrunner [1] state two equations to describe the center and radius of such a bisector circle.

Lemma 1. *Consider a circle C with radius r centered at c and two distinct sites s_i, s_j of S such that s_i, s_j lie on a ray that originates at c and such that s_i lies inside of C and s_j lies outside of C. Let r_i and r_j denote the distances from c to s_i and s_j. Then the circle C equals the bisector circle $b(i,j)$ relative to appropriate weights $\sigma(s_i)$ and $\sigma(s_j)$ if and only if*

$$r_i \cdot r_j = r^2. \tag{1}$$

Proof. We denote the intersection points of C with the supporting line of the ray from c to s_i and s_j by p and q. Assume that C equals $b(i,j)$ for appropriate weights $\sigma(s_i)$ and $\sigma(s_j)$. In particular, the points p and q are known to lie on $b(i,j)$. This implies

$$\frac{r - r_i}{r_j - r} = \frac{\sigma(s_i)}{\sigma(s_j)} = \frac{r + r_i}{r + r_j}.$$

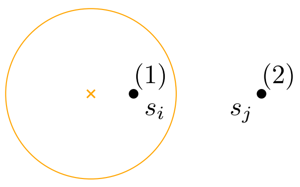
A simple algebraic manipulation yields $r_i \cdot r_j = r^2$.
Now assume that $r_i \cdot r_j = r^2$. Then

$$\frac{r - r_i}{r_j - r} = \frac{r - r^2/r_j}{r_j - r} = \frac{r}{r_j} \cdot \frac{r - r_j}{r - r_j} = \frac{r}{r_j} = \frac{r}{r_j} \cdot \frac{r_j + r}{r_j + r} = \frac{r + r^2/r_j}{r_j + r} = \frac{r + r_i}{r + r_j}.$$

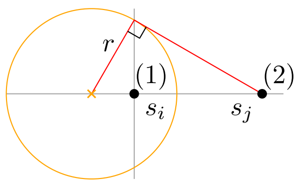
Therefore, the points p, q are guaranteed to lie on $b(i,j)$. Hence, the line segment $\overline{pq}$ forms the diameter of $b(i,j)$ and, thus, $b(i,j)$ equals C. Furthermore, appropriate weights fulfill the relation ${}^{r}/_{r_j} = {}^{\sigma(s_i)}/_{\sigma(s_j)}$. $\square$

Hence, if $r_i < r < r_j$ then $\sigma(s_i) < \sigma(s_j)$. Note that Equation (1) matches the relation that describes a circular inversion [16]: The inverse $\mathfrak{H}(C,p)$ of a point p about a circle C with center c and radius r is a point p' which lies on the ray from c through p such that $d(c,p) \cdot d(c,p') = r^2$. Let $\mathfrak{H}(C,A)$ denote the circular inversion of the area $A \subseteq \mathbb{R}^2$ about the circle C. Then Lemma 1 can be re-phrased as follows: Two sites $s_i, s_j \in S$ have a circle C as their bisector circle (for appropriate weights) if and only if $s_i = \mathfrak{H}(C, s_j)$, and vice versa.

The theory of circular inversion tells us that the inversion of a circle that is inside of C and passes through its center c is a line, while all other circles inside of C invert to circles outside of C. The interior of a circular disk D bounded by a circle C' maps to the interior of the inversion of C' if D does not contain c, and to its exterior otherwise. The center c of C is mapped to a point at infinity and vice versa. Furthermore, as sketched in Figure 3, there is a simple way to construct s_j if C and s_i are known. Hence, given a circle with radius r and center c, we may choose any point p inside or outside of the circle and obtain the inverse point p' using the equation $\overrightarrow{cp} \cdot \overrightarrow{cp'} = r^2$, where $\overrightarrow{uv}$ denotes the vector from u to v and the dot stands for the dot product of two vectors.



(a) Circular bisector



(b) Circular inversion

Figure 3: (a) The bisector circle (in orange) between the sites s_i and s_j. The weights are stated in brackets. (b) Simple construction of s_j using only s_i and the circle, based on circular inversion.

We say that a (non-empty) set $X \subset \mathbb{R}^2$ is *nested* inside a set $Y \subset \mathbb{R}^2$ if $\mathbb{R}^2 \setminus Y$ has a bounded connected component Z such that $X \subseteq Z$. We conclude this section with an insight into the topological structure of Voronoi diagrams.

Lemma 2. *If $\mathcal{R}_\sigma(s_i, S)$ is nested inside $\mathcal{R}_\sigma(s_j, S)$, for $s_i, s_j \in S$, then $\mathcal{R}_\sigma(s_i, S)$ lies entirely within the same bounded connected component of $\mathbb{R}^2 \setminus \mathcal{R}_\sigma(s_j, S)$.*

Proof. Let $\mathcal{R}_\sigma(s_i, S)$ be nested inside $\mathcal{R}_\sigma(s_j, S)$, for some $s_i, s_j \in S$ and let $S' \subset S$ such that $s_i \in S'$. We know that $\mathcal{R}_\sigma(s_i, S) \subseteq \mathcal{R}_\sigma(s_i, S')$. If $\mathcal{R}_\sigma(s_i, S)$ would lie in two different connected components of $\mathbb{R}^2 \setminus \mathcal{R}_\sigma(s_j, S)$ then even $\mathcal{R}_\sigma(s_i, \{s_i, s_j\})$ would have to be disconnected, in contradiction to the fact that $\mathcal{R}_\sigma(s_i, \{s_i, s_j\})$ is bounded by $b(i, j)$. $\square$

3. Non-Intersecting Circles and Lines

3.1. No Nested Circles in $\mathcal{G}$

Lemma 1 allows to recognize specific types of input graphs and to reconstruct suitable point sets. Let $\mathcal{G}$ be a collection of $m-1$ circles $C_1, C_2, \ldots, C_{m-1}$ which do not intersect pairwise and which are not nested. Then $\mathcal{G}$ partitions the plane into $m-1$ circular disks $D_1, D_2, \ldots, D_{m-1}$ and one unbounded region. Lemma 2 tells us that every circle of $\mathcal{G}$ has to contain a point site. Hence, $|S| = m$. We choose an arbitrary point in the unbounded region as site s_m of S. We may also choose its weight arbitrarily. Based on Lemma 1 we obtain point sites $s_1, \ldots, s_{m-1}$ of S, with s_i inside of C_i. The weights of the remaining sites are thus fixed since the inversion property needs to hold; cf. Lemma 1. By construction, s_m is the highest-weighted site and its Voronoi region is the unbounded face. Furthermore, due to Lemma 1, we know that C_i forms the bisector circle $b(i, m)$ for $i \in \{1, 2, \ldots, m-1\}$.

Suppose that $\mathcal{VD}_\sigma(S)$ differs from $\mathcal{G}$. Then there exist $1 \leq i < j < m$ such that $\mathcal{VD}_\sigma(S)$ contains a point p on the bisector $b(i, j)$ which does not lie on a circle of $\mathcal{G}$. Such a point p cannot simultaneously lie within D_i and D_j. W.l.o.g., p does not lie within D_i. Then $\mathrm{d}_\sigma(p, s_m) < \mathrm{d}_\sigma(p, s_i)$ and, thus, $p \notin \mathcal{VD}_\sigma(S)$. We conclude that $\mathcal{VD}_\sigma(S)$ equals $\mathcal{G}$. Of course, both recognition and reconstruction can be carried out in $\mathcal{O}(m)$ time.

3.2. Nested Circles in $\mathcal{G}$

Let $\mathcal{G}$ be a collection of $m-1$ circles $C_1, C_2, \ldots, C_{m-1}$ which do not intersect pairwise. They are allowed to be nested, though. Again we denote the disks defined by these circles by $D_1, D_2, \ldots, D_{m-1}$. For $1 \le i \le m-1$, we denote by f_i the face of $\mathcal{G}$ inside of D_i that is bounded by C_i and, possibly, some other circles nested inside of C_i. The unbounded face is given by f_m and its "disk" is denoted by D_m. Lemma 2 again implies $|S| = m$. However, since the circles may be nested, it is no longer good enough to choose s_m within $\bigcap_{1 \le i \le m-1} \mathfrak{H}(C_i, D_i)$.

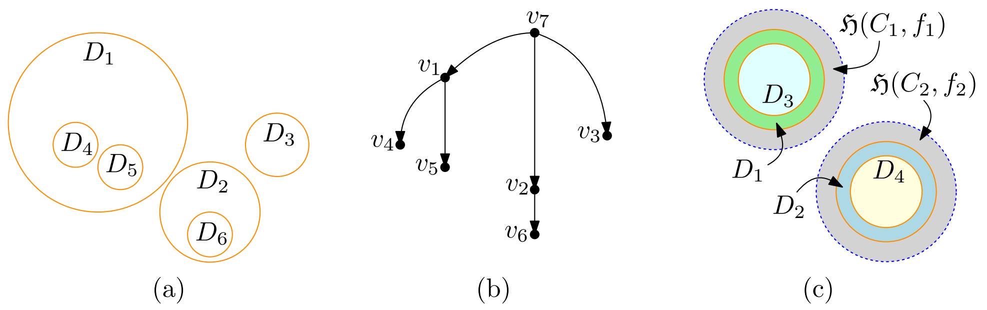


Figure 4: (a) An example illustration of $\mathcal{G}$ containing nested non-intersecting circles; (b) The dual graph $\mathcal{D}$ of $\mathcal{G}$ in (a); (c) An example where there is no solution as $\mathfrak{H}(C_1, f_1) \cap \mathfrak{H}(C_2, f_2) = \emptyset$.

We construct a dual graph $\mathcal{D}$ of $\mathcal{G}$ in the following way: For every face f_i we create a node v_i in $\mathcal{D}$. Two nodes in $\mathcal{D}$ are connected by an edge if their faces in $\mathcal{G}$ are adjacent. Since the circles in $\mathcal{G}$ are non-intersecting, $\mathcal{D}$ can not contain a cycle but forms a tree. We turn $\mathcal{D}$ into a rooted tree by rooting it at the node that corresponds to the unbounded face. In Figures 4a and 4b we illustrate such a setup where v_7 is the root node that corresponds to the unbounded face.

Next we define a *solution set*, $ss(f)$, for each face f of $\mathcal{G}$ as the loci of points that are feasible for a point site inside of f.

$$ss(f_i) := \begin{cases} D_i & \text{if } v_i \text{ is a leaf of } \mathcal{D}, \\ D_i \cap \left\{ \bigcap_{v_j \text{ is a child of } v_i} \mathfrak{H}(C_j, ss(f_j)) \right\} & \text{otherwise.} \end{cases}$$

In particular, the solution set $ss(f_m)$ for the unbounded face f_m of $\mathcal{G}$ is obtained by starting at the root node of $\mathcal{D}$ and following all branches until the leaves of $\mathcal{D}$ are reached. Then the respective point sets are mapped back to the unbounded face.

Let s_i lie within f_i. Then an inductive proof immediately implies that it is necessary for s_i to lie within $ss(f_i)$ for all $1 \leq i \leq m$. Hence, if $ss(f_i)$ is empty for some $1 \leq i \leq m$ then there exists no set S such that $\mathcal{VD}_\sigma(S)$ matches $\mathcal{D}$; cf. Figure 4c. Otherwise, we can choose an arbitrary point s_m within $ss(f_m)$. We may also choose its weight arbitrarily. The positions of all other sites $s_1, \ldots, s_{m-1}$ (and appropriate weights) are obtained by recursively computing circular inversions of s_m, as implied by $\mathcal{D}$.

It remains to argue that this construction is sufficient to ensure that $\mathcal{VD}_\sigma(S)$ matches $\mathcal{D}$. Due to Lemma 1, we know that C_i forms the bisector circle $b(i, j)$ if s_j lies outside of D_i and if s_i is obtained by a circular inversion of s_j about C_i. So suppose that there exist $1 \leq i < j \leq m$ such that $\mathcal{VD}_\sigma(S)$ contains a point on the bisector $b(i, j)$ which does not lie on a circle of $\mathcal{G}$. The arguments used for non-nested circles imply that this could only happen if the node v_j is an ancestor of the node v_i (or vice versa). Since, by construction, $\mathcal{G}$ contains the bisector of s_i and s_j if v_i is a child of v_j, we know that there is at least one node v_k that is a child of v_j and an ancestor of v_i. Hence, $\mathcal{R}_\sigma(s_i, S) \subseteq D_i \subset D_k \subset D_j$ and D_k equals $b(k, j)$. However, then every point of $\mathcal{R}_\sigma(s_i, S)$ is closer to s_k than to s_j, making it impossible for s_i and s_j to share a point that belongs to $\mathcal{VD}_\sigma(S)$.

The solution set $ss(f_m)$ is described by $m - 1$ circles (or straight lines) together with sidedness information that tells us on which side of a circle (or line) the feasible points lie. If $ss(f_m)$ is not empty then it forms a face in the arrangement of the $m - 1$ circles (and lines). Note that this approach works also if the circles of $\mathcal{D}$ are allowed to touch each other.

We now focus on the actual computation of $ss(f_m)$. We start with interpreting $\mathbb{R}^2$ as the complex plane $\mathbb{C}$. The theory of Möbius transformations tells us that every circular inversion in $\mathbb{C}$ can be modeled as a (potentially conjugated) Möbius transformation. Every such Möbius transformation maps circles and lines to circles and lines. Furthermore, the composition of two Möbius transformations yields yet another Möbius transformation, which can be computed by multiplying two matrices of $GL_2(\mathbb{C})$, the so-called general linear group of invertible 2×2 matrices over $\mathbb{C}$.

Hence, we can proceed as follows: For every non-leaf node of $\mathcal{D}$ we set up the appropriate Möbius transformation. (The identity transformation is used for the root of $\mathcal{D}$.) Then we apply an in-order traversal to $\mathcal{D}$ and, for each non-leaf node ν other than the root of $\mathcal{D}$, we compute the composition of the Möbius transformation stored at ν with the Möbius transformation stored at the parent of ν. This composed Möbius transformation replaces the old transformation stored at ν. Hence, in $\mathcal{O}(m)$ time we can obtain appropriate transformations of the $m-1$ disks associated with all nodes of $\mathcal{D}$ except for its root node.

In order to obtain $ss(f_m)$ it remains to compute the intersection of these disks. The intersection of $m-1$ disks or the disks' complements can be constructed in $\mathcal{O}(m \log m)$ time. Brown describes this representation in detail in his thesis [17]. Aurenhammer and Edelsbrunner [1] use an extended version to construct the weighted Voronoi diagram. We follow their description and describe in the following how we apply it to our setting. For every disk we embed its defining circle in the xy-plane in $\mathbb{R}^3$. We choose an arbitrary *point of inversion* p_i in $\mathbb{R}^3$ that does not lie on the xy-plane, e.g., $(0,0,1)$ the point above the origin at $z=1$. Note that by using a single circle and a point not on the circle we can uniquely define a sphere such that both circle and point lie on the sphere's boundary. Hence, for each circle we create a unique sphere in combination with p_i. Then, p_i lies on all $m-1$ spheres. Using p_i as point of inversion we apply a spherical inversion that creates a half-space from every sphere. For each disk computed for $ss(f_m)$ we know whether the disk or its complement is to be considered. If the disk's complement is required then we form the complement of the respective half-space. We can form the intersection of these $m-1$ half-spaces in $\mathcal{O}(m \log m)$ time. The result is a convex polyhedron $\mathfrak{P}$ in $\mathbb{R}^3$. To obtain a representation of $\mathbb{R}^2$ we invert the xy-plane using p_i as well. The result is a sphere S_{xy} that contains p_i. Then we intersect $\mathfrak{P}$ with S_{xy}. Since $\mathfrak{P}$ is formed from $m-1$ half-spaces it has a combinatorial complexity of $\mathcal{O}(m)$. Hence, we can traverse each facet f of $\mathfrak{P}$ and intersect it with S_{xy}. As each facet is convex as well we find the intersection for every f in $\mathcal{O}(|f|)$ time. We keep the portion of each facet that lies outside of S_{xy}. Hence, $ss(f_m)' := \mathfrak{P} \cap \overline{S_{xy}}$ is constructed in $\mathcal{O}(m)$ time, where $\overline{S_{xy}}$ denotes the complement of S_{xy}. Let e' denote an edge of $\mathfrak{P}$ that is shortened by the intersection process. Let e denote e' transformed back to the xy-plane. The two planes that are locally incident at e' imply two specific disks with respect to e in the xy-plane. The shortened endpoint of e', and e respectively, is the point where the boundaries of the two disks meet.

Transforming $ss(f_m)'$ back into the xy-plane yields $ss(f_m)$ and is accomplished in linear time in the size of the intersection. Therefore we obtain $ss(f_m)$ in overall $\mathcal{O}(m \log m)$ time.

Recall that our circular inversions may create half-planes as well: In case a transformed circle intersects the center of an inversion circle it is transformed into a line, i.e., half-plane. Let ℓ denote such a line in $\mathbb{R}^2$. Instead of a sphere we form a plane in $\mathbb{R}^3$ that intersects ℓ and the inversion point p_i. We take advantage of the inversion property of the plane that intersects the inversion point, that is, the plane inverts into the same plane. The half-plane defining ℓ defines the half-space for the plane and we can apply our half-space intersection as described above.

3.3. Nested Circles and Lines in $\mathcal{G}$

Let $\mathcal{G}$ be a collection of $m-1$ circles and lines which do not intersect pairwise. The circles are allowed to be nested, though. We follow the notation of the previous section. Let k denote the number of lines $\ell_1, \ldots, \ell_k$ in $\mathcal{G}$. Clearly for $k > 1$ the lines have to be parallel to be non-intersecting. Since a line is not finite it can only partition the unbounded face. Hence, the lines partition the unbounded face into $k+1$ unbounded regions. For two sites s_i, s_j to form a line $\ell \in \mathcal{G}$ as their bisector the sites must have equal weight and therefore equal distance to ℓ, cf. Section 2. We modify our inversion function $\mathfrak{H}(.,.)$ such that the inverse of a point about a line is simply its mirrored image. To obtain a solution we construct the dual graph $\mathcal{D}$ for each unbounded face separately. Then, we compute the solution set for each tree root $v^1, \ldots, v^{k+1}$. Finally, starting at an unbounded face that is incident to at most one line ℓ, we map its solution arrangement via ℓ to the next consecutive face and form the intersection with its solution space. We repeat this process until we reach the last unbounded face and thereby obtain a full characterization of the solution.

4. Recognizing $\mathcal{G}$ as Bisector Graph

Given a planar circular-arc graph $\mathcal{G}$, we ask whether $\mathcal{G}$ is a weighted bisector graph. If it is a bisector graph, then we seek a set S of suitable sites and a corresponding weight function σ. That is, we want to find a solution (S, σ) such that every edge (of the embedding) of $\mathcal{G}$ lies on a bisector defined by two sites of S. Contrary to Section 3 we now assume that $\mathcal{G}$ contains at least one node. For a start, we also assume that all nodes of $\mathcal{G}$ are of degree exactly three.

We begin by studying the structure of a weighted bisector graph by establishing the following lemmas:

Lemma 3. *Let s_i, s_j, s_k denote three sites, with $\sigma(s_i) < \sigma(s_j) < \sigma(s_k)$. Then there exists a line ℓ which contains the centers of all three bisectors $b(i,j)$, $b(j,k)$, and $b(i,k)$.*

Proof. Consider sites s_i, s_j with $\sigma(s_i) < \sigma(s_j)$ and their common bisector arc $b(i,j)$. This circular arc lies on a circle centered at $c(i,j)$; cf. Figure 5.

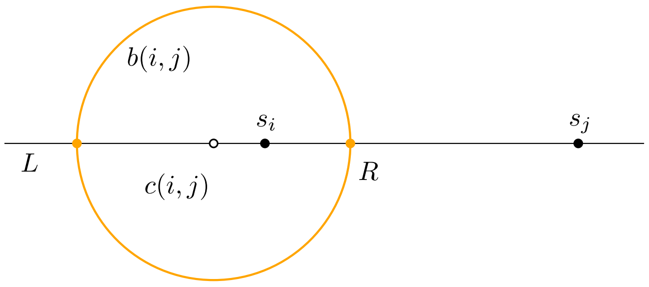


Figure 5: Two weighted sites and their bisector.

The points L and R are two points on the bisector, equidistant (in weighted terms) to both s_i, s_j. Point L is of maximal distance and the point R of minimal distance. Now it holds that

$$R = s_i + (s_j - s_i) \cdot \frac{\sigma(s_i)}{\sigma(s_i) + \sigma(s_j)} = \frac{s_i\sigma(s_i) + s_i\sigma(s_j) + s_j\sigma(s_i) - s_i\sigma(s_i)}{\sigma(s_i) + \sigma(s_j)}$$
$$= \frac{s_i\sigma(s_j) + s_j\sigma(s_i)}{\sigma(s_i) + \sigma(s_j)},$$

and likewise it follows from $^{(s_i-L)}/_{(s_j-L)} = {}^{\sigma(s_i)}/_{\sigma(s_j)}$ that

$$L = \frac{s_j\sigma(s_i) - s_i\sigma(s_j)}{\sigma(s_i) - \sigma(s_j)}.$$

Since $c(i,j) = {}^1/_2(L+R)$, we immediately see that

$$c(i,j) = \frac{s_j\sigma(s_i)^2 - s_i\sigma(s_j)^2}{\sigma(s_i)^2 - \sigma(s_j)^2}.$$

Now consider three sites, s_i, s_j, s_k with $\sigma(s_i) < \sigma(s_j) < \sigma(s_k)$. We want to show that the centers of their bisector circles, i.e., $c(i,j)$, $c(i,k)$, and $c(j,k)$ are collinear. Three points are collinear if and only if the area of the triangle defined by them is zero. Thus, the three points in question are on the same supporting line if the following determinant vanishes:

$$D := \begin{vmatrix} x(c(i,j)) & y(c(i,j)) & 1 \\ x(c(i,k)) & y(c(i,k)) & 1 \\ x(c(j,k)) & y(c(j,k)) & 1 \end{vmatrix}$$

where $x(p)$ and $y(p)$ denote the x- and y-coordinates of a point p.

After expanding the determinant we get

$$\begin{aligned} D &= x(c(i,j)) \cdot (y(c(i,k)) - y(c(j,k))) \\ &\quad + x(c(i,k)) \cdot (y(c(j,k)) - y(c(i,j))) \\ &\quad + x(c(j,k)) \cdot (y(c(i,j)) - y(c(i,k))) \\ &= \frac{(x(s_j)\sigma(s_i)^2 - x(s_i)\sigma(s_j)^2)\left(\frac{y(s_k)\sigma(s_i)^2 - y(s_i)\sigma(s_k)^2}{\sigma(s_i)^2-\sigma(s_k)^2} - \frac{y(s_k)\sigma(s_j)^2 - y(s_j)\sigma(s_k)^2}{\sigma(s_j)^2-\sigma(s_k)^2}\right)}{\sigma(s_i)^2 - \sigma(s_j)^2} \\ &\quad + \frac{(x(s_k)\sigma(s_i)^2 - x(s_i)\sigma(s_k)^2)\left(\frac{y(s_k)\sigma(s_j)^2 - y(s_j)\sigma(s_k)^2}{\sigma(s_j)^2-\sigma(s_k)^2} - \frac{y(s_j)\sigma(s_i)^2 - y(s_i)\sigma(s_j)^2}{\sigma(s_i)^2-\sigma(s_j)^2}\right)}{\sigma(s_i)^2 - \sigma(s_k)^2} \\ &\quad + \frac{(x(s_k)\sigma(s_j)^2 - x(s_j)\sigma(s_k)^2)\left(\frac{y(s_j)\sigma(s_i)^2 - y(s_i)\sigma(s_j)^2}{\sigma(s_i)^2-\sigma(s_j)^2} - \frac{y(s_k)\sigma(s_i)^2 - y(s_i)\sigma(s_k)^2}{\sigma(s_i)^2-\sigma(s_k)^2}\right)}{\sigma(s_j)^2 - \sigma(s_k)^2} \\ &= 0, \end{aligned}$$

thus having proved the claim. □

Lemma 4. *Let s_i, s_j, s_k denote three distinct sites. If two of the three bisectors $b(i,j)$, $b(j,k)$, and $b(i,k)$ intersect in a point p, then all three bisectors intersect in p.*

Proof. Assume that $b(i,j)$ and $b(j,k)$ intersect in the point p. Since $b(i,j)$ is the set of points of equal (weighted) distance to s_i and s_j, and $b(j,k)$ is the set of points of equal distance to s_j and s_k, it follows that p has equal distance to s_i, s_j, and s_k. Thus, p also lies on $b(j,k)$. □

Corollary 1. *Let s_i, s_j, s_k denote three distinct sites. Then $b(i,j)$, $b(j,k)$ and $b(i,k)$ intersect pairwise either in exactly two points, exactly one point, or not at all.*

4.1. Finding Sites from Bisectors

Let v denote a node of degree three of the graph $\mathcal{G}$. This node is the intersection of three circular arcs, i.e., of the bisector circles of three sites. We seek the locations of these sites given the bisector arcs. Let i, j, k be the indices of these three sites, and denote the centers of their bisector circles by $c(i,j)$, $c(j,k)$, $c(i,k)$; see Figure 6.

Lemma 3 tells us that the centers $c(i,j)$, $c(j,k)$, $c(i,k)$ lie on a line ℓ. Thus, we can, w.l.o.g., assume the centers to lie on the x-axis and v to be at coordinates $(0,1)$. (The general case is reduced to this setting by rotation, scaling, and translation.)

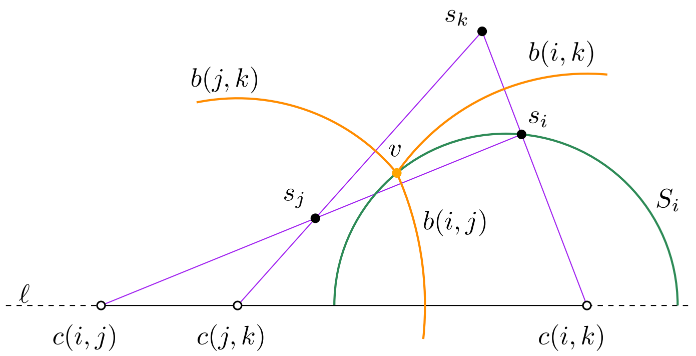


Figure 6: Reconstructing the three sites that traced out the orange bisector arcs incident at node v. A site s_i necessarily lies on an arbitrary location on the green solution circle S_i. Sites s_j and s_k then follow by inversions of s_i across the corresponding bisector.

Next, we set up the conjugated Möbius transforms $\mathfrak{H}_{i,j}$, $\mathfrak{H}_{j,k}$, and $\mathfrak{H}_{i,k}$, across bisectors $b(i,j)$, $b(j,k)$, and $b(i,k)$, respectively, Furthermore, we concatenate all three transforms into a single transform $\mathfrak{H} := \mathfrak{H}_{i,j} \circ \mathfrak{H}_{j,k} \circ \mathfrak{H}_{i,k}$.

Let the x-coordinates of $c(i,j)$, $c(i,k)$, and $c(j,k)$ be given by x_1, x_2, and x_3. Then, the Möbius transform $\mathfrak{H}$ is given by the matrix product of three individual transforms. Simple math yields

$$\mathfrak{H} = \begin{pmatrix} (x_1 - x_2 + x_3 + x_1x_2x_3) & (1 + x_1x_2 - x_1x_3 + x_2x_3) \\ (1 + x_1x_2 - x_1x_3 + x_2x_3) & (-x_1 + x_2 - x_3 - x_1x_2x_3) \end{pmatrix}.$$

Now it must hold that $S_i = \mathfrak{H}_{i,j}(S_j)$ and $S_j = \mathfrak{H}_{j,k}(S_k)$ and $S_k = \mathfrak{H}_{i,k}(S_i)$, where S_i is a solution set for site s_i, and S_j and S_k are defined likewise. In particular, $S_i = \mathfrak{H}_{i,j}(\mathfrak{H}_{j,k}(\mathfrak{H}_{i,k}(S_i)))$, or equivalently, $S_i = \mathfrak{H}(S_i)$.

Solving this equation for $S_i =: (x, y)$ yields

$$y^2 + x^2 - \frac{2(x_1 - x_2 + x_3 + x_1x_2x_3)}{1 + x_1x_2 - x_1x_3 + x_2x_3}x - 1 = 0,$$

which is a circle with center at coordinates

$$c(s_i) = \left(\frac{x_1 - x_2 + x_3 + x_1x_2x_3}{1 + x_1x_2 - x_1x_3 + x_2x_3}, 0\right)$$

and which goes through the intersection point of all bisectors at $(0, 1)$. Likewise, the solution set for S_j and S_k are circles, namely the corresponding inverses or Möbius transforms of S_i.

Therefore, given three intersecting bisectors, we can find sites s_i, s_j, s_k that generate these bisectors. In general the sites are not uniquely defined, and instead each site lies on a "solution circle". This establishes Lemma 5.

Lemma 5. *Let v denote a node of degree three of $\mathcal{G}$. Let $b(i, j)$, $b(i, k)$, and $b(j, k)$ define the three arcs that meet at v. Then the sites s_i, s_j, and s_k lie on solution circles S_i, S_j, and S_k, which can be constructed. Picking a specific locus for s_i on S_i automatically fixes the other sites, and vice versa.*

4.2. Bisector Graph Recognition

In the following, we describe how to obtain a solution (S, σ) such that every arc of $\mathcal{G}$ lies on one bisector defined by (S, σ), if such a solution exists, and thereby detect whether $\mathcal{G}$ is a bisector graph. Recall that every node of $\mathcal{G}$ has degree three.

Let $f_1, \ldots, f_m$ denote the faces of $\mathcal{G}$, where $m \geq n$ for $n := |S|$. Let f_i denote a face of $\mathcal{G}$ with a maximal number of boundary nodes. The nodes on the boundary of f_i are given by $v_1, \ldots, v_k$. We have $k > 0$ because we had assumed to have at least one node in $\mathcal{G}$. As each node is given by the intersection of distinct bisectors, we necessarily have $k > 1$. In case $k = 2$, we apply Lemma 5 to obtain a family of solutions for each face with degree two and combine these local solutions using the process outlined in Section 3.2. Note that we are able to handle nested circles in $\mathcal{O}(m)$ time whenever $\mathcal{G}$ includes degree-three vertices, as applying the conjugated Möbius transform on $\mathcal{G}$ already yields all feasible sites for the subgraphs of $\mathcal{G}$ that contain vertices. Thus, it is not necessary to compute the entire solution set for the nested circles in $\mathcal{G}$.

Otherwise, the face f_i has $k \geq 3$ nodes. Note that it is still possible for each bounded face to have at most two nodes. However, in that case two or more nodes will be incident to the unbounded face. Applying Lemma 5 to one node of f_i will yield a family of solutions for the site s_i for f_i. However, we can even restrict this solution to a constant number of points as follows. We traverse the boundary of f_i, node by node, and apply Lemma 5. Thereby we produce k solution circles for s_i, denoted by $S_i^1, \ldots, S_i^k$. Necessarily, s_i is in the intersection of these solution circles: $s_i = \bigcap_{j=1}^{k} S_i^j$.

Let S_i^1 and S_i^2 denote two circles such that $S_i^1 \neq S_i^2$. Note that a solution circle S_i^j computed for v_j intersects v_j. Thus, for every S_i^j we can find a different node (when $k \geq 3$) on f_i which is not on S_i^j and so its solution circle will be different. Therefore, two such circles exist. Hence, the intersection $s_i = \bigcap_{j=1}^{k} S_i^j$ contains at most two points. If $s_i = \emptyset$ then $\mathcal{G}$ is not a bisector graph. Otherwise, these points are loci for the site, thus establishing Lemma 6.

Lemma 6. *For a face f of $\mathcal{G}$ with at least three boundary nodes the set of solutions for the site of f consists of at most two points.*

Every other solution circle intersects either both solution points or reduces the solution to a single point p. We now assume that the solution consists of a single point p. (Otherwise we apply the following process to both points.)

We choose p as locus for our site s_i with arbitrary weight. We use the identified site s_i and apply inversions $\mathfrak{H}(.,.)$ via every arc defining f_i, obtaining all neighboring sites. Then we repeat the process for all neighboring sites and their neighbors in turn, traversing the entire graph $\mathcal{G}$, breadth first. Thereby we obtain the solution (S, σ) after $\mathcal{O}(m)$ inversions. In every face processed we verify that the solution circles defined by the boundary nodes contain the alleged site. If a solution circle does not contain the alleged site then $\mathcal{G}$ does not constitute a bisector graph for this starting point p. (Recall that we may have up to two starting locations for p.)

4.3. Complexity

We find a face f_i with $k \geq 3$ nodes in $\mathcal{O}(m)$ time. We find the solution circles in $\mathcal{O}(1)$ time per node and intersect them in $\mathcal{O}(k)$ time. Using the resulting intersection point p, we apply the breadth-first traversal, which takes again $\mathcal{O}(m)$ time, as each face f is processed in time linear in the combinatorial size of the face. Therefore we derive Lemma 7.

Lemma 7. *Given a planar circular-arc graph $\mathcal{G}$ with m faces, in time $\mathcal{O}(m)$ we can detect whether $\mathcal{G}$ constitutes a bisector graph and, if yes, find suitable (S, σ).*

Note that S may contain more than one site on the same locus. This is admissible for a bisector graph but not for a Voronoi diagram.

4.4. Handling Nodes of Higher Degree

At the start of this section we had assumed that all vertices of $\mathcal{G}$ are of degree three. This restriction can be waived: Consider a vertex of $\mathcal{G}$ with degree larger than three and its incident bisector arc circles. We can distinguish two fundamentally distinct cases: a) The centers of all the circles are collinear, or b) not all of them are collinear.

If all centers lie on the same supporting line then we can construct a Möbius transform $\mathfrak{H}$ that represents the inversions across all the incident bisector arcs in the same way as described in the proof of Lemma 5. Once we have that, we can once more solve $s_i = \mathfrak{H}(s_i)$ and thus obtain all valid locations for a site. Note that, for instance with degree four vertices and the bisector circles in specific configurations, it may be that the entire plane is a valid solution. Other degree-four configurations may yield only the trivial solution of the intersection point of all bisectors and thus will never appear in bisector graphs.

If not all centers lie on the same supporting line then we can reduce the problem to subproblems of smaller size. For instance, if we have a degree-four vertex, we consider appropriate pairs of centers of the bisector arcs. For each pair, we construct its supporting line, and its intersection is the center for another bisector, one that did not have arcs represented in $\mathcal{G}$. Thus, for each supporting line we now have three bisector arcs, and the procedure from Lemma 5 yields a solution circle for a site. By intersecting the solution circles for the different supporting lines we obtain the location of a site. Figure 7 demonstrates this procedure.

5. Recognizing $\mathcal{G}$ as Voronoi Diagram

Consider a planar circular-arc graph $\mathcal{G}$. Does there exist a solution set (S, σ) such that $\mathcal{VD}_\sigma(S)$ equals $\mathcal{G}$? Since $\mathcal{VD}_\sigma(S)$ is a bisector graph, we start by applying the bisector-graph detection presented in the previous section.

If $\mathcal{G}$ is not recognized as a bisector graph, then it is not a Voronoi diagram. However, even if we can find a suitable (S, σ), then $\mathcal{G}$ still need not be a

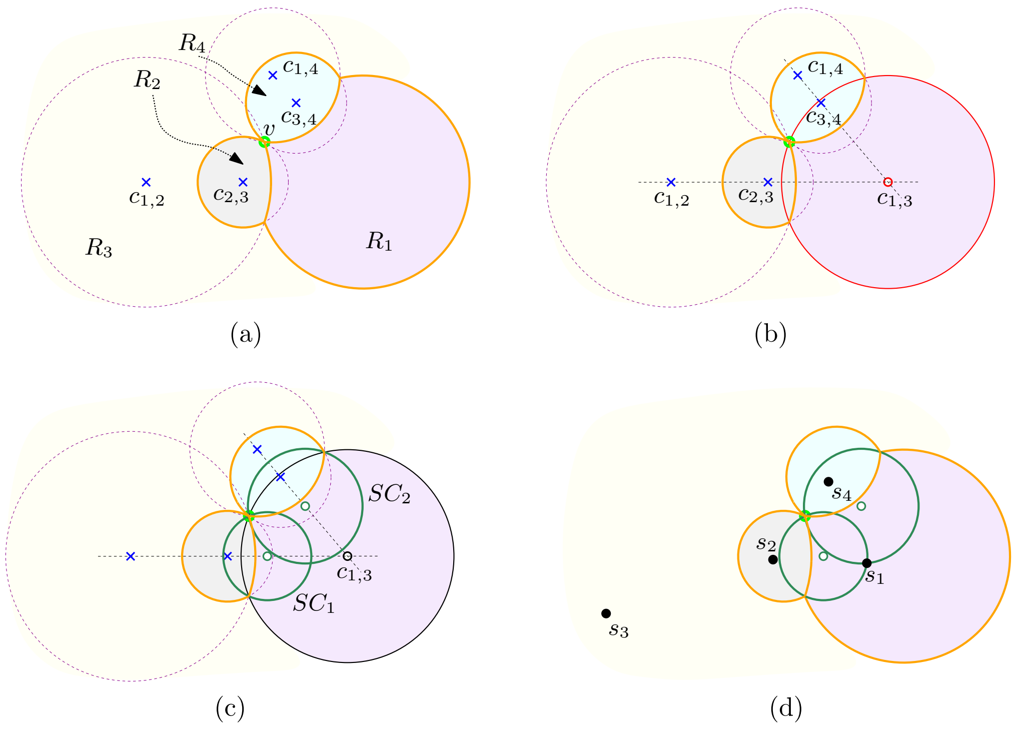


Figure 7: In a bisector graph with nodes of higher degree, we can partition the incident arcs, construct individual solution sets, and intersect those. (a) Graph $\mathcal{G}$ with centers of the bisector arcs incident at v. For instance, $c_{1,2}$ is the center of the bisector between sites s_1 and s_2, which had traced out regions R_1 and R_2. (b) Even if we do not have the bisector given, we can construct the bisector (red) between s_1 and s_3. (c) Now we have two systems of three bisectors each: The s_1, s_2, s_3-system, and the s_1, s_3, s_4- system. For each of them, we can construct solution circles for s_1 (green): SC_1 and SC_2. (d) The intersection of these solution circles yields a position for s_1. The other sites are obtained by standard spherical inversions across the appropriate bisector arcs.

Voronoi diagram. In Figure 8, we illustrate two examples which both are bisector graphs for the same pair (S, σ) but only Figure 8b is a Voronoi diagram. Observe that Figure 8a does not contain the face f in the center of Figure 8b.

A canonical way to verify whether $\mathcal{G}$ is a Voronoi diagram is to compute $\mathcal{VD}_\sigma(S)$ using the approach by Aurenhammer and Edelsbrunner [1]. Their algorithm is worst-case optimal and runs in $\mathcal{O}(n^2)$ time and space. Of course, this is a waste of time if the combinatorial complexity of $\mathcal{G}$ is sub-quadratic.

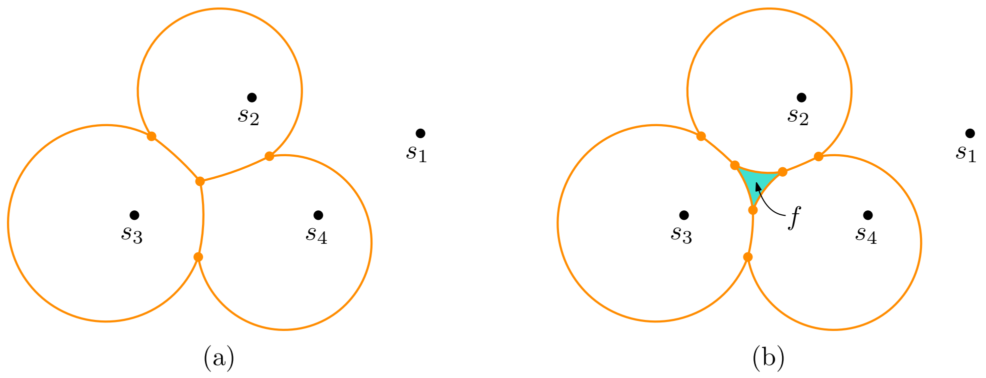


Figure 8: Given sites $S := \{s_1, \ldots, s_4\}$ and appropriate weights, then (a) shows a bisector graph of S and (b) $\mathcal{VD}_\sigma(S)$.

Alternatively, one may use the strategies presented by Har-Peled and Raichel [2] or Held and de Lorenzo [3], which allow to compute $\mathcal{VD}_\sigma(S)$ in expected $\mathcal{O}(n \log^3 n)$ or $\mathcal{O}(n \log^4 n)$ time, respectively, under the assumption that the corresponding weights are sampled from some random distribution. A final comparison between the diagram computed and $\mathcal{G}$ yields the decision sought: We find a common node in $\mathcal{G}$ and $\mathcal{VD}_\sigma(S)$ and apply a breath-first traversal to compare all nodes and arcs. Therefore, the comparison can be carried out in $\mathcal{O}(m)$ time.

6. Discussion

We present a novel approach for recognizing whether a given planar circular-arc graph $\mathcal{G}$ is a weighted bisector graph, and, provided that this is the case, for reconstructing the respective input sites. The Möbius transformation is central to our approach, as it allows us to generate a solution set, or confirm that $\mathcal{G}$ is no weighted bisector graph whenever no such set exists.

Whenever $\mathcal{G}$ does not contain nodes, we are even able to determine whether $\mathcal{G}$ is a multiplicatively weighted Voronoi diagram $\mathcal{VD}_\sigma(S)$, and to reconstruct S. If $\mathcal{G}$ has been recognized as a bisector graph of (S, σ) then the main difficulty of verifying that a general bisector graph $\mathcal{G}$ matches $\mathcal{VD}_\sigma(S)$ is given by deciding whether all disconnected faces of $\mathcal{VD}_\sigma(S)$ that do not contain their defining sites are also present in $\mathcal{G}$. It remains a question for future research to confirm that a bisector graph of (S, σ) matches $\mathcal{VD}_\sigma(S)$ without explicitly computing $\mathcal{VD}_\sigma(S)$.

Acknowledgements

This work was supported by the Austrian Science Fund (FWF): Grant P31013.

References

[1] F. Aurenhammer, H. Edelsbrunner, An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane, Pattern Recogn. 17 (2) (1984) 251 – 257. doi:10.1016/0031-3203(84)90064-5.

[2] S. Har-Peled, B. Raichel, On the Complexity of Randomly Weighted Multiplicative Voronoi Diagrams, Discrete Comput. Geom. 53 (3) (2015) 547–568. doi:10.1007/s00454-015-9675-0.

[3] M. Held, S. de Lorenzo, An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams, in: Proc. 28th Annu. Europ. Symp. Alg. (ESA'20), 2020, pp. 9:1–9:15. doi:10.4230/LIPIcs.ESA.2020.9.

[4] B. N. Boots, Weighting Thiessen Polygons, Economic Geography 56 (3) (1980) 248–259. doi:10.2307/142716.

[5] G. Eder, M. Held, Weighted Voronoi Diagrams in the Maximum Norm, Internat. J. Comput. Geom. Appl. 29 (03) (2019) 239–250. doi:10.1142/S0218195919500079.

[6] J. N. Portela, M. S. Alencar, Cellular Coverage Map as a Voronoi Diagram, J. Communication and Information Systems 23 (1) (2008) 22–31. doi:10.14209/jcis.2008.3.

[7] M. Parter, D. Peleg, On the Relations Between SINR Diagrams and Voronoi Diagrams, in: Proc. 14th Int. Conf. Ad-hoc, Mobile, and Wireless Networks, 2015, pp. 225–237. doi:10.1007/978-3-319-19662-6_16.

[8] A. R. David González G., Harri Hakula, J. Hämäläinen, Spatial Mappings for Planning and Optimization of Cellular Networks, IEEE/ACM Trans. Networking 26 (1) (2018) 175–188. doi:10.1109/TNET.2017.2768561.

[9] P. F. Ash, E. D. Bolker, Recognizing Dirichlet Tessellations, Geometriae Dedicata 19 (2) (1985) 175–206. doi:10.1007/BF00181470.

[10] D. Hartvigsen, Recognizing Voronoi Diagrams with Linear Programming, ORSA J. on Comp. 4 (4) (1992) 369–374. doi:10.1287/ijoc.4.4.369.

[11] F. Aurenhammer, Recognizing Polytopical Cell Complexes and Constructing Projection Polyhedra, J. Symb. Comp. 3 (3) (1987) 249–255. doi:10.1016/S0747-7171(87)80003-2.

[12] T. Biedl, M. Held, S. Huber, Recognizing Straight Skeletons and Voronoi Diagrams and Reconstructing Their Input, in: Proc. 10th Int. Sympos. Voronoi Diagrams in Sci. & Eng. (ISVD'13), 2013, pp. 37–46. doi:10.1109/ISVD.2013.11.

[13] O. Aichholzer, T. Biedl, T. Hackl, M. Held, S. Huber, P. Palfrader, B. Vogtenhuber, Representing Directed Trees as Straight Skeletons, in: Proc. 23rd Int. Symp. Graph Drawing & Network Visualization (GD 2015), 2015, pp. 335–347. doi:10.1007/978-3-319-27261-0_28.

[14] O. Aichholzer, H. Cheng, S. L. Devadoss, T. Hackl, S. Huber, B. Li, A. Risteski, What Makes a Tree a Straight Skeleton?, in: Proc. 24th Canad. Conf. Comp. Geom (CCCG'12), 2012, pp. 267–272.

[15] G. Eder, M. Held, P. Palfrader, Recognizing Geometric Trees as Positively Weighted Straight Skeletons and Reconstructing Their Input, Internat. J. Comput. Geom. Appl. 29 (03) (2019) 251–267. doi:10.1142/S0218195919500080.

[16] N. Altshiller-Court, College Geometry: An Introduction to the Modern Geometry of the Triangle and the Circle, Dover Publications, 2007.

[17] K. Q. Brown, Geometric Transforms for Fast Geometric Algorithms, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1979).