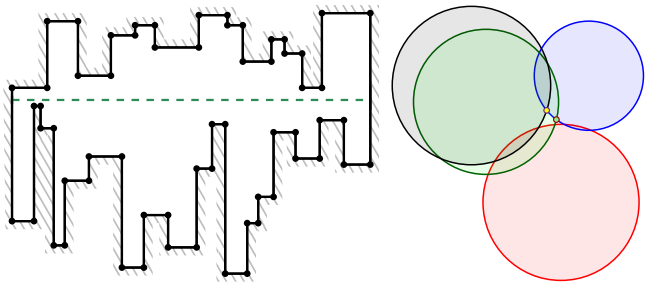
Routing and Stabbing



Dissertation zur Erlangung des akademischen Grades

eines Doktors der Naturwissenschaften

vorgelegt am

Fachbereich Mathematik und Informatik

der Freien Universität Berlin

von

Max Willert

Berlin, 2020

Erstgutachter: Prof. Dr. Wolfgang Mulzer
Zweitgutachter: Prof. Dr. Matias Korman

Tag der Disputation: 24. März 2021

Selbständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbständig und ohne Benutzung anderer als der anegegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Dissertation wurde in gleicher oder ähnlicher Form noch in keinem früheren Promotionsverfahren eingereicht. Mit einer Prüfung meiner Arbeit durch ein Plagiatsprüfungsprogramm erkläre ich mich einverstanden.

_____

Max Willert

(Berlin, 27. November 2020)

Abstract

Routing. Let G be a simple, connected, undirected graph. We consider *routing with preprocessing* in G. In a preprocessing step, each vertex of G receives a *label* and a *routing table*. Then, we must be able to route a packet between any two vertices s and t of G, where each step may use only the label of the target node t, the routing table of the current node and the packet header. This problem has been studied extensively for general graphs, where efficient routing schemes with polylogarithmic routing tables have turned out to be impossible. Thus, special graph classes are of interest.

Let P be an x-monotone orthogonal polygon with n vertices. We call P a *simple histogram* if its upper boundary is a single edge; and a *double histogram* if it has a horizontal chord from the left boundary to the right boundary. Two points p and q in P are *co-visible* if and only if the (axis-parallel) rectangle spanned by p and q completely lies in P. In the r-*visibility graph* $\mathrm{Vis}(P)$ of P, we connect two vertices of P with a unit weighted edge if and only if they are co-visible. We present a routing scheme for visibility graphs of simple and of double histograms that have label size $\lceil \log n \rceil$ and table size $O(\log n \deg(v))$ for each vertex v of P, where $\deg(v)$ is the degree of v in $\mathrm{Vis}(P)$. In simple histograms we can route along a shortest path and need no additional header, whereas in double histograms we need headers of size $\lceil \log n \rceil$ and we can route on a path that has twice the length of an optimal path. The preprocessing time is in both cases $O(m)$, where m is the number of edges in $\mathrm{Vis}(P)$.

Let $V \subset \mathbb{R}^2$ be a set of n sites in the plane. The *unit disk graph* $\mathrm{DG}(V)$ of V is the graph with vertex set V where two sites v and w are adjacent if and only if their Euclidean distance is at most 1. The edge weights correspond to the Euclidean distance of its endpoints. Moreover, we use D to denote the diameter of $\mathrm{DG}(V)$. We show that for any given $\varepsilon > 0$, we can construct a routing scheme for $\mathrm{DG}(V)$ that achieves stretch $1 + \varepsilon$, has label size $O\left(\varepsilon^{-1} \log D \log^3 n / \log\log n\right)$, table size $\varepsilon^{-O(\varepsilon^{-2})} \log^3 n (1 + \log D / \log\log n)$ and the header needs at most $O(\log^2 n / \log\log n)$ bits. The preprocessing time is $O(\varepsilon^{-1} n^2 \log^2 n)$.

Stabbing. Suppose we are given a set $\mathcal{D}$ of n pairwise intersecting disks in the plane. A planar point set P *stabs* $\mathcal{D}$ if and only if each disk in $\mathcal{D}$ contains at least one point from P. We present a deterministic algorithm that takes $O(n)$ time to find five points that stab $\mathcal{D}$. This provides a simple—albeit slightly weaker—algorithmic version of a classical result by Danzer that such a set $\mathcal{D}$ can always be stabbed by four points. Furthermore, we give a simple example of 13 pairwise intersecting disks that cannot be stabbed by three points.

Acknowledgements

After ten years of learning, teaching, and living at Freie Universität Berlin, this thesis closes an important period of my life. There are many people who helped me to conclude this thesis and who became important persons in my life.

First of all, I would like to thank Wolfgang Mulzer, who was a great supervisor. His door was always open for all my questions – including the stupid ones. It was inspiring to talk to him about computational geometry, algorithms, mathematics and it was no problem to chat about off-topics – in Berlin, but on trips to other cities and countries as well. Moreover, I am very thankful for his confidence in my teaching skills.

Furthermore, thank you to Matias Korman and his Sendai crew for giving me and my colleagues the opportunity to visit the wonderful country Japan, to eat a lot of new and delicious food, and of course to work in a group of brilliant researchers. I have fond memories of these visits.

Additionally, I would like to thank the group of the theoretical computer scientists for their great support in all facets. Especially Katharina and Jonas offered great help during summer term 2018 but also their comments on my thesis. It was a pleasure to me to study and work with them. Moreover, thank you to Bahar for her technical support regarding my thesis. I will also never forget our joint time in Japan. Finally, thank you to Frank and Klaus. Their tuition is the basement of my mathematical knowledge. Frank, we all miss you!

I would also like to thank all my coauthors: the research group from Sendai, I already mentioned, but also the group from Israel around Micha Sharir, Liam Roditty, and Haim Kaplan. Both groups presented nice research topics I really enjoyed to think about. They encouraged me with their feedback to present shorter and more intuitive proofs.

Thanks to Nils and his quite concentrated work on his bachelor thesis, I could focus on the last meters of my own work.

I am very grateful for all my students and teaching assistants I could supervise in the last decade. Most of all, I loved our discussions about mathematics, computer science, and good teaching. I am so proud of you all and hope that I could give you the core ideas about student centered teaching. In this context, I would like to thank the professors Ralf Romeike, Christian Haase, and Jan Vahrenhold for their ideas and trust.

Last but not least, I would like to thank my family for their emotional support and of course their interest for my research.

Contents

CHAPTER 1

Introduction

Graph theory is one of the main research fields in computer science [CLRS09]. It provides a mathematical model that can be used to analyze networks, like wireless networks, Internet, traffic, but also social connections between people. In this thesis we consider two different problems connected to graph theory and examine the influence of geometry on these problems.

1.1 Routing

Suppose we want to watch our favorite movie using our favorite streaming provider. The video file lies on a big server – we do not know where. However, the file is split into small data packets and each packet has to find its way in the Internet to our home desktop – it has to be *routed* through a network.

The *routing problem* is a classic question in distributed graph algorithms [Gav01, GS04, PU89]. We would like to preprocess a graph G for the following task: given a data packet located at a *source* vertex s, route the packet to a *target* vertex t, identified by its *label*. We strive for three main properties: *locality* (to find the next step of the packet, the scheme should use only information at the current vertex or in the packet header), *efficiency* (the packet should choose a path that is not much longer than a shortest path between s and t), and *compactness* (the space for labels, routing tables, and packet headers should be as small as possible). The ratio between the length of the *routing path* and a shortest path is called *stretch factor*.

Obviously, we could store at each vertex v of G the complete shortest path tree of v. This routing scheme is local and perfectly efficient: we can send the packet along a shortest path. However, the scheme lacks compactness. Thus, the general challenge is to balance the (potentially) conflicting goals of compactness and efficiency, while maintaining locality.

There are many routing schemes for general graphs (e.g., [ABNLP90, AG11, Che13, Cow01, EGP03, RT15, RT16, SK85] and the references therein). For example, the scheme by Roditty and Tov [RT16] stores a poly-logarithmic number of bits in the packet header and it routes a

packet from s to t on a path of length $O\big(k\Delta + m^{1/k}\big)$, where $k > 2$ is any fixed integer, Δ is the shortest path distance between s and t, and m is the number of edges in the graph. The routing tables use $mn^{O(1/\sqrt{\log n})}$ total space, where n is the number of vertices.

In the late 1980's, Peleg and Upfal [PU89] proved that in general graphs, any routing scheme with constant stretch factor must store $\Omega(n^c)$ bits per vertex, for some constant $c > 0$. Later, Gavoille and Gengler could prove that there exists an n-vertex graph on which every routing scheme that achieves stretch factor strictly smaller than 3 requires at least $\Omega(n^2)$ bits of total routing information [GG01].

This provides ample motivation to focus on special graph classes to obtain better routing schemes. For instance, trees admit routing schemes that always follow the shortest path and that store $O(\log^2 n/\log\log n)$ bits at each node [FG01, TZ01]. Moreover, in planar graphs, for any fixed $\varepsilon > 0$, there is a routing scheme with a poly-logarithmic number of bits in each routing table that always finds a path that is within a factor of $1+\varepsilon$ from optimal [Tho04]. Furthermore, in graphs with bounded doubling dimension we can find a huge number of routing schemes, e.g. [AGGM06, CGMZ16, Sli05, Tal04], but the most recent result is by Konjevod, Richa, and Xia [KRX16]. For any fixed $\varepsilon > 0$, there is a routing algorithm for graphs of doubling dimension α with $O(\log n)$ bit labels, $O(\log^2 n/\log\log n)$ header size, and routing tables of size $\varepsilon^{-O(\alpha)}\log^3 n$. Observe, that the table size does not depend on the normalized diameter of the input graph.

Another approach is *geometric routing*: the graph lies in a geometric space, and the routing algorithm must find the next vertex for the packet based on the coordinates of the source and the target vertex, the current vertex, and its neighborhood; see, e.g., [BFvRV15, BFvRV17] and the references therein. In contrast to compact routing schemes, there are no routing tables, and the routing is purely based on the local geometry (and possibly the packet header). For example, the routing algorithm for triangulations by Bose and Morin [BM04] uses the line segment between the source and the target for its routing decisions. In a recent result, Bose, Fagerberg, van Renssen, Verdonschot [BFvRV17] showed that if vertices do not store any routing tables, no geometric routing scheme can achieve stretch factor $o(\sqrt{n})$. This holds irrespective of the header size.

We consider a particularly interesting and practically relevant class of geometric graphs, namely visibility graphs of polygons. Banyassady et al. [BCK+17] presented a routing scheme for polygonal domains with n vertices and h holes that uses $O(\log n)$ bits for the label, $O((\varepsilon^{-1}+h)\log n)$ bits for the routing tables, and achieves a stretch of $1+\varepsilon$, for any fixed $\varepsilon > 0$. However, their approach is only efficient if the edges of the visibility graph are weighted with their Euclidean lengths. Banyassady et al. ask whether there is an efficient routing scheme for visibility graphs with unit weights (the *hop-distance*). This setting seems to be more relevant in practice, and similar results have already been obtained in unit disk graphs for routing schemes [KMRS18, YXD12] and for spanners [Bin20, JV09].

We focus on r-visibility graphs of simple and double histograms. A *simple histogram* is a monotone orthogonal polygon whose upper boundary consists of a single edge; a *double histogram* is a monotone orthogonal polygon that has a horizontal chord that touches the boundary of P only at the left and the right boundary. Let P be a (simple or double) histogram with n vertices. Two vertices v and w in P are connected in the r-visibility graph $\mathrm{Vis}(P)$

by an unweighted edge if and only if the axis-parallel rectangle spanned by v and w is contained in the (closed) region P. We say that v and w are *co-visible*. At first, the focus on r-visibility graphs of (double) histograms may seem a strong restriction. However, even this case turns out to be quite challenging and reveals the whole richness of the compact routing problem in unweighted, geometrically defined graphs: on the one hand, the problem is still highly nontrivial, while on the other hand, much better results than in general graphs are possible. Histograms constitute a natural starting point, as they are often crucial building blocks in visibility problems [Bär11, BGM+14, BGP17, BGR17, BS14, HKS+18]. Moreover, r-visibility is a popular concept in orthogonal polygons that enjoys many useful structural properties; see, e.g., [Hof90, HKS+18, MRS90, O'R87, WK07] and the references therein for more background on histograms and r-visibility.

Another graph class of interest comes from the study of mobile and wireless networks. These networks are usually modeled as *unit disk graphs* [CCJ90]. Nodes in this network are points in the plane and two nodes are connected if their distance is at most one. This is equivalent to a disk intersection graph in which all disks have diameter one. For unit disk graphs there are several known routing schemes. The first routing scheme is by Kaplan, Mulzer, Roditty, and Seiferth [KMRS18]. They present a routing scheme with stretch $1+\varepsilon$ and routing table size $O(\log^2 n \log^2 D)$, where D is the diameter of the given unit disk graph. Their routing is recursive and needs an additional header of size $O(\log n \log D)$. The second routing scheme is due to Yan, Xiang, and Dragan [YXD12]. They present a routing scheme with label size $O(\log^2 n)$ and show that a data packet routes along a path of length at most $5\Delta + 13$, where Δ is the length of the optimal path. The two results use slightly different notions of routing schemes; see Chapter 2 for further details.

Thorup and Zwick introduced the notion of a *distance oracle* [TZ05]. Given a graph G, the goal is to construct a compact data structure to quickly answer *distance queries* for any two nodes in G. A routing scheme can be seen as a distributed implementation of a distance oracle [RT16]. We will use the recently developed distance oracle from Chan and Skrepetos [CS19] and turn it into an efficient routing scheme for unit disk graphs.

1.2 Stabbing

Another well-known problem from graph theory is the clique problem. It is defined as follows.

> **Clique Problem:**
>
> **Given:** simple, undirected graph G, integer value $k \in \mathbb{N}$
>
> **Question:** Does G contain the complete graph with k vertices?

In 1972, Richard Karp proved that the clique problem is NP-hard [Kar72]. Furthermore, Zuckerman [Zuc06] showed that the optimization version of the clique problem is hard to approximate, unless $P = NP$.

However, if the input graph is the intersection graph of disks we can find efficient (approximation) algorithms: for unit disk graphs the clique problem can be solved in polynomial time [CCJ90]. Moreover, Bonamy et al. presented a randomized EPTAS for the clique problem in

disk graphs [BBB+18]. Finally, Ambühl and Wagner [AW05] presented a polynomial time algorithm that computes a $\tau/2$-approximation, where τ is the so called *stabbing number*. One goal of this thesis will be to understand this stabbing number. Therefore, we want to give a short definition and later we present some bounds of this number.

Let $\mathcal{D}$ be a set of n disks in the plane. If every *three* disks in $\mathcal{D}$ intersect, then Helly's theorem shows that the whole intersection $\bigcap \mathcal{D}$ of $\mathcal{D}$ is nonempty [Hel23, Hel30, Rad21]. In other words, there is a single point p that lies in all disks of $\mathcal{D}$, i.e., p *stabs* $\mathcal{D}$. More generally, when we know only that every *pair* of disks in $\mathcal{D}$ intersect, there must be a point set P of constant size such that each disk in $\mathcal{D}$ contains at least one point in P – the minimum cardinality of P is the *stabbing number* of $\mathcal{D}$. It is not surprising that $\mathcal{D}$ can be stabbed by a constant number of points, but for some time, the exact bound remained elusive. Eventually, in July 1956 at an Oberwolfach seminar, Danzer presented the answer: four points are always sufficient and sometimes necessary to stab any finite set of pairwise intersecting disks in the plane. Danzer was not satisfied with his original argument, so he never formally published it. In 1986, he presented a new proof [Dan86]. Previously, in 1981, Stachó had already given an alternative proof [Sta84], building on a previous construction of five stabbing points [Sta65]. This line of work was motivated by a result of Hadwiger and Debrunner, who showed that three points suffice to stab any finite set of pairwise intersecting *unit* disks [HD55]. In later work, these results were significantly generalized and extended, culminating in the celebrated (p, q)-theorem that was proven by Alon and Kleitman in 1992:

> **(p, q)-Theorem.** Let $\mathcal{D}$ be a family of compact, convex sets in $\mathbb{R}^d$ and let $p \geqslant q \geqslant d + 1$ be natural numbers. If among any p members of the family some q have a nonempty intersection than the family $\mathcal{D}$ can be stabbed by a constant number[1] of points. [AK92]

See also a recent paper by Dumitrescu and Jiang that studies generalizations of the stabbing problem for translates and homothets of a convex body [DJ11].

Danzer's published proof [Dan86] is fairly involved. It uses a compactness argument that does not seem to be constructive, and one part of the argument relies on an underspecified verification by computer. Therefore, it is quite challenging to check the correctness of the argument, let alone to derive any intuition from it. There seems to be no obvious way to turn it into an efficient algorithm for finding a stabbing set of size four. The two constructions of Stachó [Sta84, Sta65] are simpler, but they are obtained through a lengthy case analysis that requires a very disciplined and focused reader. Here, we present a new argument that yields five stabbing points. Our proof is constructive, and it lets us find the stabbing set in deterministic linear time. Following an open question in Har-Peled et al. [HKM+18], Carmi, Katz, and Morin posted a draft on the arXiv in which they claim the existence of an algorithm that can find four stabbing points in linear time [CKM18].

As for lower bounds, Grünbaum gave an example of 21 pairwise intersecting disks that cannot be stabbed by three points [Grü59]. Later, Danzer reduced the number of disks to ten [Dan86]. This example is close to optimal, because every set of eight disks can be stabbed by three points, as mentioned by Stachó [Sta65]. We prove his claim. However, it is hard to

[1] To be more precise: the number depends only on p, q and d, but is independent of $|\mathcal{D}|$.

verify Danzer's lower bound example—even with dynamic geometry software, the positions of the disks cannot be visualized easily. In this thesis, we present a simple construction that needs 13 disks and can easily be verified, almost by inspection.

1.3 Contributions

We briefly discuss the major contributions of this thesis.

Routing in orthogonal polygons. Let P be a simple or double histogram with n vertices. The visibility graph $\mathrm{Vis}(P)$ of P has vertex set $V(P)$ and there is an edge between two vertices $v, w \in V(P)$ if and only if they are r-visible, i.e., the axis-spanned rectangle of v and w is contained in P. The edge weights equal to 1. We use m to denote the number of edges of $\mathrm{Vis}(P)$, and $\deg(v)$ is the degree of v in $\mathrm{Vis}(P)$. We provide routing schemes for simple and double histograms. For simple histograms we present a routing scheme with stretch 1, label size $\lceil \log n \rceil$, no headers, table size $O(\log n \deg(v))$ for $v \in V$, and preprocessing time $O(m)$. For double histograms we present a routing scheme with stretch 2, label and header size $\lceil \log n \rceil$, table size $O(\log n \deg(v))$ for $v \in V$, and preprocessing time $O(m)$.

Routing in unit disk graphs. Let V be a set of n points in the Euclidean plane. The unit disk graph $\mathrm{DG}(V)$ of V has vertex set V and there is an edge between two vertices $v, w \in V$ if and only if their Euclidean distance is at most 1. The edge weight equals the Euclidean distance of its endpoints. Let D be the diameter of $\mathrm{DG}(V)$. For each $\varepsilon > 0$, we present a routing scheme for unit disk graphs with stretch $1+\varepsilon$, label size $O\left(\varepsilon^{-1} \log D \log^3 n / \log\log n\right)$, table size $\varepsilon^{-O(\varepsilon^{-2})} \log^3 n (1 + \log D / \log\log n)$, header size $O(\log^2 n / \log\log n)$, and preprocessing time $O(\varepsilon^{-1} n^2 \log^2 n)$. This routing scheme is based on the well-known shortest-path separator decomposition.

Stabbing pairwise intersecting disks. Let $\mathcal{D}$ be a set of n pairwise intersecting disks in the Euclidean plane. We present a new proof that shows that the stabbing number of $\mathcal{D}$ is upper bounded by 5. Moreover, we obtain a linear time algorithm that can find these 5 stabbing points assuming the real ram model. Finally, we present a simple construction of 13 pairwise intersecting disks that cannot be stabbed by 3 points.

1.4 Thesis Outline

We now describe the structure of this thesis. First of all, we present geometric and graph-theoretic background in the preliminaries and define the concept of routing. The remainder of the thesis is then split into two parts. In the first part, we present the already mentioned routing schemes for visibility graphs of simple and double histograms as well as for unit disk graphs. In the second part we present some lower bounds on the stabbing number, give the

construction of five stabbing points for pairwise intersecting disks and show how these five points can be computed in linear time. We conclude with open problems.

1.5 Publications

The results that are covered by this thesis appeared in the following publications.

[HKM+18] Sariel Har-Peled, Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, Micha Sharir, and Max Willert. Stabbing pairwise intersecting disks by five points. *In Proceedings of the 29th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 50:1–50:12, 2018.

[CCK+20] Man-Kwun Chiu, Jonas Cleve, Katharina Klost, Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, and Max Willert. Routing in histograms. *In Proceedings of the 14th International Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 43–54, 2020.

[MW20] Wolfgang Mulzer and Max Willert. Compact routing in unit disk graphs. *In Proceedings of the 31st Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 16:1–16:14, 2020.

CHAPTER 2

Preliminaries

In this chapter, we want to present the notational aspects of geometry, graphs, and routing schemes. Throughout the thesis we use Σ to denote the binary alphabet, i.e., $\Sigma = \{0, 1\}$. Moreover, Σ^* denotes the set of all binary strings (including the empty string). Additionally, let $[m] = \{0, 1, \ldots, m\}$, for $m \in \mathbb{N}$.[1] For a complete description of geometric and graph theoretic objects, see [DBVKOS97] and the references therein.

2.1 Geometry

2.1.1 Points, Lines and Disks

Throughout the whole thesis we work in the *Euclidean plane*. A *point* p is a tuple of two real values, i.e., $p = (p_x, p_y) \in \mathbb{R}^2$. Let p and q be two points in $\mathbb{R}^2$. The *Euclidean distance* between p and q is defined as

$$|pq| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

Moreover, we use $\overline{pq}$ to denote the *line segment* between the points p and q, that is

$$\overline{pq} = \{\lambda \cdot p + (1 - \lambda) \cdot q \mid \lambda \in [0, 1]\}.$$

We refer to p and q as the *endpoints* of the line segment $\overline{pq}$. The *length* $|\overline{pq}|$ of the line segment equals the Euclidean distance $|pq|$ of its endpoints. Next, for a point $c \in \mathbb{R}^2$ and a real value $r \geqslant 0$, we use $D(c, r)$ to denote the *closed Euclidean disk* with *center* c and *radius* r, i.e.,

$$D(c, r) = \{p \in \mathbb{R}^2 \mid |cp| \leqslant r\}.$$

Later, we will define the notion of balls. This notion does not differ too much from the notion of Euclidean disks. However, whenever we talk about disks we mean Euclidean disks.

[1] Remark that it is more common to exclude 0 from $[m]$. Nevertheless, we include 0 for technical reasons.

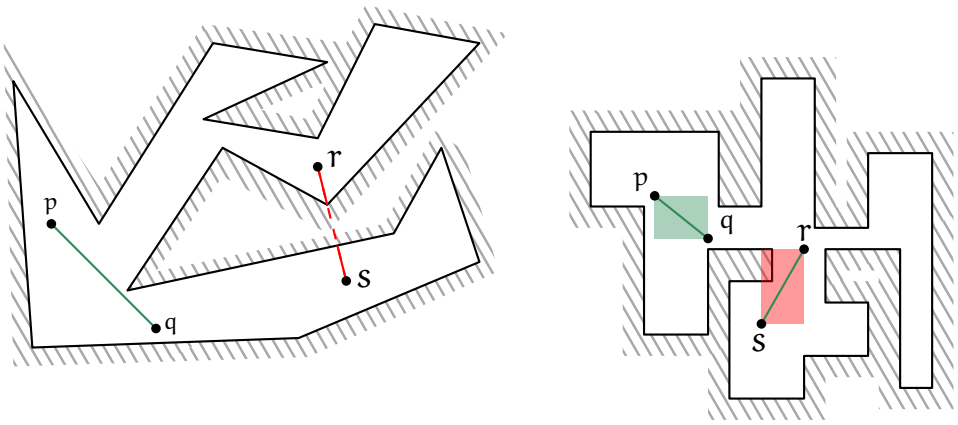


Figure 2.1: Left: ℓ-visibility. The points p and q can see each other, the points r and s cannot. Right: r-visibility. The points p and q are co-visible. The points r and s are not co-visible but they see each other with respect to ℓ-visibility.

2.1.2 Simple Polygons

A *closed polygonal chain* $\pi = \overline{p_0p_1}, \overline{p_1p_2}, \ldots, \overline{p_{n-2}p_{n-1}}, \overline{p_{n-1}p_0}$ is a sequence of n line segments, where every 3 consecutive endpoints $p_{i-1}, p_{i \bmod n}, p_{(i+1) \bmod n}$ are non-collinear. It suffices to identify the polygonal chain with its *vertices* p_i, i.e. $\pi = p_0p_1 \ldots p_{n-1}$. Furthermore, the polygonal chain is *simple* if and only if adjacent line segments intersect only in one of their endpoints and any two non-adjacent line segments do not intersect at all. A simple, closed polygonal chain π in the Euclidean plane always separates the plane into two disjoint sets: the bounded interior of π – noted by $\operatorname{int}\pi$ – and the unbounded exterior.

Definition 2.1 (Polygon)**.** *Let* $\pi = p_0p_1 \ldots p_{n-1}$ *be a simple, closed polygonal chain of at least* 3 *vertices. A* polygon $P(\pi)$ *is the set of all points of the Euclidean plane contained in* $\operatorname{int}\pi$ *or in* π*, i.e.,* $P(\pi) = \pi \cup \operatorname{int}\pi$*. The polygonal curve* π *is the* boundary of $P(\pi)$*. The set* $\operatorname{int}\mu$ *is the* interior of $P(\pi)$*. We use the notation* $V(P) = \{p_0, \ldots, p_{n-1}\}$ *for the set of* vertices*.*

Visibility. When we talk about polygons it is natural to look at the concept of *visibility*. There are at least two different concepts. First of all, there is ℓ*-visibility* (or line-visibility), which is the standard type and also the most natural one. Here, two points $p, q \in P$ can see each other if and only if the line segment $\overline{pq}$ is fully contained in P. Note, that the line segment is allowed to touch the boundary since by definition the boundary is part of the polygon. Secondly, there is the so called r*-visibility* (or rectangular-visibility). In this concept, two points $p, q \in P$ can see each other if and only if the axis-aligned rectangle $\square(p, q)$ spanned by p and q is fully contained in the polygon P. Both concepts are shown in Figure 2.1. Throughout this thesis, we only use r-visibility. Hence, we say that two points p and q are *co-visible* or *see each other* if and only if $\square(p, q) \subseteq P$. However, r-visibility only makes sense in polygons where each boundary segment is parallel to the x- or the y-axis.

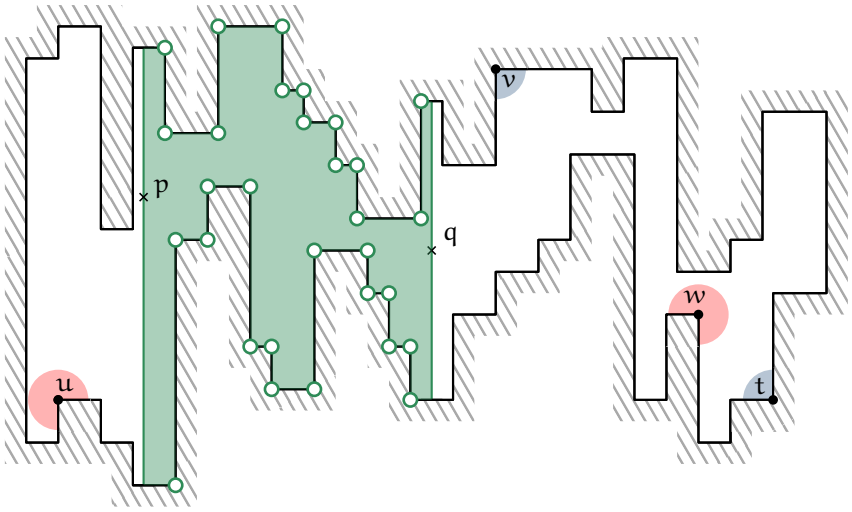


Figure 2.2: This figure shows an orthogonal x-monotone polygon. The interval $[p, q]$ is the set of green vertices. The vertices u and w are reflex, the vertices v and t are convex. On the other hand, u and v are left vertices, while w and t are right vertices.

Orthogonal polygons. A polygon P is said to be *orthogonal* if every edge is either horizontal or vertical, i.e., every interior angle is either $\pi/2$ or $3\pi/2$.

We can classify the vertices of P as follows. A vertex v in P is incident to exactly one horizontal edge h. We call v a *left* vertex if it is the left endpoint of h; otherwise, v is a *right* vertex. Furthermore, v is *convex* if the interior angle at v is $\pi/2$; otherwise, v is *reflex*. Accordingly, every vertex of P is either *ℓ-convex*, *r-convex*, *ℓ-reflex*, or *r-reflex*. The different types of vertices are shown in Figure 2.2.

Monotone polygons. An orthogonal polygon P is said to be *x-monotone* if and only if for every vertical line ℓ the intersection of ℓ with P is either empty or one line segment; refer to Figure 2.2. Next, let p and q be two points (not necessarily vertices) in P. We say that p is *(strictly) to the left of* q, if $p_x \leqslant q_x$ (or $p_x < q_x$). The term *(strictly) to the right of* is defined analogously. The *interval* $[p, q]$ of p and q is the set of vertices in P between p and q, i.e.,

$$[p, q] = \{v \in V(P) \mid p_x \leqslant v_x \leqslant q_x\}.$$

The interval of two points is depicted in Figure 2.2.

2.1.3 Range Spaces

We define the notion of *hypergraphs* or *range spaces*, which are synonyms. The following definitions and properties can be found in [HP11].

Definition 2.2. *A* range space $\mathcal{S}$ *is a pair* $(X, \mathcal{R})$*, consisting of a set* X *and a set of* ranges $\mathcal{R}$*, which is a family of subsets of* X*.*

In computational geometry, as also in our work, the *Vapnik-Chervonenkis dimension* [VC15] plays a crucial role. For this, let Y be a subset of X. Then, $\mathcal{S}_Y = (Y, \mathcal{R}_Y = \{R \cap Y \mid R \in \mathcal{R}\})$ is the *range space induced by* Y. We say that Y *is shattered by* $\mathcal{S}$ if $\mathcal{R}_Y$ is the power set of Y, i.e., it contains every subset of Y and therefore, $|\mathcal{R}_Y| = 2^{|Y|}$ if Y is finite.

Definition 2.3. *Let* $\mathcal{S} = (X, \mathcal{R})$ *be a range space. The* VC-dimension *of* $\mathcal{S}$ *is the maximum cardinality of a shattered subset of* X*. If there are arbitrarily large shattered subsets, i.e., the maximum does not exist, the VC-dimension is* ∞*.*

Next, we define another important measure. Let $\mathcal{S} = (X, \mathcal{R})$ be a range space. The *shatter function* is defined as

$$\pi_{\mathcal{S}}(n) = \max_{Y \subseteq X, |Y| = n} |\mathcal{R}_Y|.$$

Finally, the *shattering dimension* of $\mathcal{S}$ is the smallest number δ such that $\pi_{\mathcal{S}}(n) \in O(n^{\delta})$ for all $n \in \mathbb{N}$. We will need the following important property that relates the shattering dimension to the VC-dimension. A proof can be found in [HP11].

Lemma 2.1. *Let* $\mathcal{S}$ *be a range space,* d *its VC-dimension and* δ *its shattering dimension. Then we have* $\pi_{\mathcal{S}}(n) \in O(n^d)$ *and* $d \in O(\delta \log \delta)$*.*

Remark. The first statement of the lemma implies $\delta \leqslant d$ which is known as Sauer-Shelah lemma [Sau72, She72].

2.2 Graphs

A *graph* G is a pair of *vertices* V_G and *edges* E_G. We use V instead of V_G and E instead of E_G if the context is clear. Throughout this thesis we consider *simple, undirected* graphs, which means that E does not contain any loops or multi-edges, i.e., $E \subseteq \binom{V}{2}$. Moreover, we only consider *finite* graphs: V is a finite set and we use n to denote its cardinality. Hence, a graph G can be seen as range space in which every range has cardinality 2. For a vertex $v \in V$, we use $N(v)$ to denote the *neighborhood* of v, i.e., the set of adjacent vertices of v in G. The *degree* of v in G is defined as $\deg(v) = |N(v)|$. Moreover, the *closed neighborhood of* v, denoted by $N[v]$, contains v and all its neighbors. We extend $N(\cdot)$ to sets as usual: for $W \subseteq V$, we let $N(W)$ be the set of all vertices of G, that have a neighbor in W. Finally, we let $N[W] = N(W) \cup W$.

Paths and connectedness. A sequence $\pi : \langle v_0, v_1, \ldots, v_k \rangle$ of vertices is called a *path* if and only if two consecutive vertices of π share an edge, i.e., $\{v_{i-1}, v_i\} \in E$, for $i = 1, 2, \ldots, k$. Moreover, we call v_0 and v_k the *endpoints of* π. The path is *simple* if and only if $v_i \neq v_j$, for $i \neq j$. Moreover, let $\mathrm{Path}_G(s, t)$ be the set of all paths in G whose endpoints are s and t. A graph G is *connected* if and only if for every pair of vertices $s, t \in V$ the set $\mathrm{Path}_G(s, t)$ is non-empty.

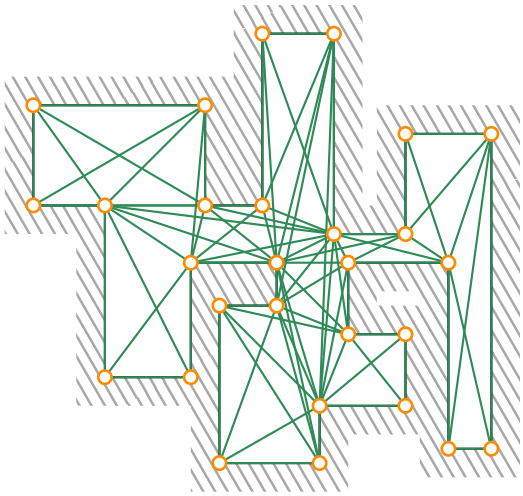


Figure 2.3: This figure shows the r-visibility graph of the orthogonal polygon from Figure 2.1.

Shortest path metric. We consider also *weighted* graphs. That means our graph G is a triple consisting of the set of vertices V, the set of edges E, and a weight function $w\colon E \to \mathbb{R}^+$ that assigns to every edge $e \in E$ a positive value $w(e)$. Next, we want to define the distance of two vertices in the graph. First of all, let $\pi : \langle v_0, v_1, \ldots, v_k \rangle$ be a path. We use $|\pi|$ to denote the *length of* π, that is

$$|\pi| = \sum_{i=1}^{k} w\big(\{v_{i-1}, v_i\}\big).$$

We define the *shortest path distance of* s *and* t *in* G as

$$d_G(s,t) = \min\{|\pi| \mid \pi \in \mathrm{Path}_G(s,t)\}.$$

We omit the subscript $_G$ if the context is clear. Moreover, since the weights of the edges are positive, we know that $d_G(s,t)$ is well-defined and every shortest path between s and t is simple. The *diameter of* G, denoted by $\mathrm{diam}(G)$, is the length of the longest shortest path, i.e.,

$$\mathrm{diam}(G) = \max_{s,t \in V(G)} d_G(s,t).$$

Next, for $v \in V$ and $r > 0$, we define $B(v,r) = \{w \in V \mid d_G(v,w) \leqslant r\}$ as the *ball of* v *with radius* r *in* G. Moreover, it is important to define the *hop distance* and to distinguish it from the shortest path distance. For the weighted graph $G = (V, E, w)$ we define $H(G) = (V, E, w_1)$ where $w_1(e) = 1$, for each $e \in E$. In other words, we forget the original weights of the edges and assign to every edge the unit cost 1. Then, the *hop distance of* s *and* t *in* G, denoted by $h_G(s,t)$, is the shortest path distance of s and t in $H(G)$. More formally, it is $h_G(s,t) = d_{H(G)}(s,t)$. Again, we omit the subscript $_G$ if the context is clear.

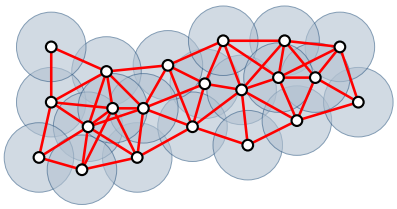


Figure 2.4: The disks in the unit disk graph have diameter 1 and there is an edge between two centers if and only if their corresponding disks intersect.

Visibility graphs. Let P be an orthogonal polygon. The *visibility graph* $\mathrm{Vis}(P)$ of P is a simple, undirected, weighted graph with vertex set $V(P)$ and an edge between every pair of vertices $v, w \in V(P)$ if and only if v and w are co-visible, that is $\square(v, w) \subseteq P$; see for example Figure 2.1. The weight of the edge $\{v, w\}$ equals the unit weight 1. Hence, in our visibility graph we focus on the hop distance. Since P is simple, $\mathrm{Vis}(P)$ is connected. Recall, that for a vertex $v \in V(P)$ the set $N(v)$ contains exactly the vertices that share an edge with v. In other words, $N(v)$ is the set of vertices visible from v.

Unit Disk Graphs. Let $V \subset \mathbb{R}^2$ be a set of n sites in the Euclidean plane. The *unit disk graph* $\mathrm{DG}(V)$ of V is a simple, undirected, weighted graph with vertex set V and an edge between every pair of vertices $v, w \in V$ if and only if $|vw| \leqslant 1$; see Figure 2.4. The weight of the edge $\{v, w\}$ equals the Euclidean distance $|vw|$. Throughout, we will assume that $\mathrm{DG}(V)$ is connected. Clearly, we have $\mathrm{diam}(G) \leqslant n - 1$. As depicted in Figure 2.4, a unit disk graph can also be seen as a *disk intersection graph*: there is an edge between two vertices v and w if and only if the (Euclidean) disks $D(v, 0.5)$ and $D(w, 0.5)$ intersect. Thus, for $v \in V$, we have $N(v) = D(v, 1) \cap V = B(v, 1)$. Recall, that $D(\cdot, \cdot)$ is the Euclidean disk, whereas $B(\cdot, \cdot)$ is the graph theoretic ball.

2.3 Routing Schemes

In this section, we discuss all the different routing models and provide a precise definition of our notion of a routing scheme. For this, let $G = (V, E, w)$ be a simple, undirected, weighted, and connected graph with n vertices.

The ports. The first main ingredient for a precise definition of a routing scheme is the port model. Let $v \in V$ be a vertex. This vertex can access an adjacent vertex $w \in N(v)$ through a *port number* p, i.e., the vertex v can send information to w by using the port p. We write $w = \mathrm{node}(v, p)$. Different port numbers lead to different neighbors. In terms of routing the literature distinguishes at least two different models [FG01, FG02, TZ01]. In the *fixed-port model* we treat the port numbers as part of the input and hence, we are not allowed to change them, they are assigned in arbitrary manner. On the other hand, there is the *designer-port*

model. Here, it is allowed to assign port numbers during the preprocessing steps. This gives much more power to the designer of the routing scheme [FG01, FG02, TZ01, YXD12]. However, a routing scheme that uses the designer-port model cannot easily be used as a building block for more complicated routing schemes, since additional lookup tables become necessary in order to store the assignments of the port numbers. The routing schemes in this thesis use the fixed-port model. We assume that a port number is represented as a binary string of length exactly $\lceil \log n \rceil$.

The labels. The next main idea for a routing scheme is the concept *label*. Each vertex $v \in V$ obtains a label $\text{lab}(v) \in \Sigma^*$ which is a binary string. The literature distinguishes at least two models. In the *name-independent model* the label of a vertex corresponds to an identifier of the vertex which is given as input, see for instances [AGM04a, AGM+04b, ACL+06, KRX16]. Hence, in this model the designer is not allowed to assign any labels different from the identifier. The routing scheme is built on top of the names. In contrast to that, the *labeled model* allows the designer to store additional information in the label. Hence, the label is used to identify the node in the network but might also contain more information about the topology of the graph. This model gives more power to the designer. Throughout this thesis, we use the labeled model.

The tables. Every node $v \in V$ in the network obtains a *routing table* $\text{tab}(v) \in \Sigma^*$ which is – like the label – a binary string. This routing table contains information about the topology of the graph as well as the identity of the corresponding vertex v. A lot of routing schemes do not distinguish between labels and tables, because we can easily concatenate both strings. The designers of these routing schemes talk about labels rather than tables. However, the concatenation only makes sense if the string lengths are asymptotically equal. This is the reason, why we want to distinguish between labels and tables.

The header. A data packet consists of three parts: the message, the static header and the dynamic header. The *message* is the information that has to be sent to the destination node, like an email or a video. Its content is independent of the routing scheme – we ignore it. Let us assume that the data packet is located at some node (perhaps the destination). The node has to know the destination of the data packet to decide on the hop to the next node (or to read the message). This part of information is stored in the *static header*. It is created during some initialization step at the source node and does not change again. The *dynamic header* is a bit string that contains additional information and might change during the routing of the data packet through the network. Like other routing schemes, we want to use the term "target label" instead of "static header" and "header" rather than "dynamic header". It is not a problem to use this notion, since in this thesis the static header will always contain the target label only. As we can see next, in some cases the static header might not contain the target label. However, this does not occur in this thesis.

The initialization of the header. When a data packet is created at some source node s and we are given some target label $\text{lab}(t)$, the static part of the header has to be initialized.

There are different techniques how this can be done. First, the most obvious idea is to put $\mathrm{lab}(t)$ into the static header. The second idea is to use $\mathrm{lab}(t)$ as well as $\mathrm{tab}(s)$ to compute a static header which might differ from $\mathrm{lab}(t)$. Both ideas are simple and can be done locally. We briefly sketch a third idea that is called *handshaking*, refer to [TZ01]. For this, we use a data packet whose message is empty and send it through the network in a deterministic fashion. While this packet travels through the network it collects information about the graph. Eventually, this data packet – together with the collected information – returns to the start node s. Finally, we use $\mathrm{lab}(t)$, $\mathrm{tab}(s)$, and the collected information to compute the static header. Obviously, handshaking is more general than the second idea which is more general than the first idea. Moreover, handshaking is not restricted to the initialization of the static header. It can also be used during the routing process to update the dynamic header. However, we will not make use of this concept – neither for the initialization nor the routing process. We will use the first idea for our definition of a routing scheme to keep the presentation as slim as possible.

The routing function. Last but not least we have to discuss the process of routing. For this, we assume that a data packet, together with its static header h_s and its dynamic header h_d, is located at a node $v \in V$. The *routing function* uses the two parts of the header as well as the routing table $\mathrm{tab}(v)$ of the current node v and computes a port p as well as a new dynamic header h'_d. The data packet is then sent to the next vertex $\mathrm{node}(v, p)$ while the dynamic header of the data packet is updated to h'_d. Hence, the routing function can be seen as a transition function like we have in finite automatons – it models exactly one step of the routing process. In the literature we can find different answers on the question of what a routing function is allowed to access. For instance, the concept of *geometric routing* allows the routing function to have direct access to geometrically induced coordinates [BFvRV15, BFvRV17, BKvRV17a, BKvRV17b]. These coordinates might be irrational which yields an impractical routing scheme. Moreover, we can find routing schemes where the routing function has direct access to the vertices in the neighborhood of the current one [KMRS18]. Since the neighborhood test problem is fundamental in the context of routing we will not make use of these tricks.

The mathematical definition. Let $\mathcal{G}$ be a graph class. The routing model is depicted in Figure 2.5. A *routing scheme* $\mathcal{R}$ for $\mathcal{G}$ consists of a family of *label functions* $\mathrm{lab}_G : V_G \to \Sigma^*$ and *table functions* $\mathrm{tab}_G : V_G \to \Sigma^*$, for each $G \in \mathcal{G}$. As before, we omit the subscript $_G$ if the context is clear. Furthermore, $\mathcal{R}$ has a *routing function* $\sigma : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^* \times \Sigma^*$. The routing function σ describes the behavior of the routing scheme, as follows: assume a data packet with (dynamic) header h is located at a vertex $s \in V_G$ and must be routed to a destination $t \in V_G$. Then, $\sigma(\mathrm{tab}(s), \mathrm{lab}(t), h)$ computes a port p so that the next hop of the data packet is from s to $\mathrm{node}(s, p)$, as well as a string h' which is then updated in the header of the data packet. Now, let $v_0 = s$, $h_0 = \varepsilon$, $\sigma(\mathrm{tab}(v_i), \mathrm{lab}(t), h_i) = (p_{i+1}, h_{i+1})$ and $v_{i+1} = \mathrm{node}(v_i, p_{i+1})$, for $i \geqslant 0$. The sequence $(v_i, h_i)_{i \in \mathbb{N}}$ is called *routing sequence*. The routing scheme $\mathcal{R}$ is *correct* for $G \in \mathcal{G}$, if and only if for all distinct vertices $s, t \in V_G$, there is a number $m(s, t) \in \mathbb{N}$ such that $v_i = t$ and $h_i = \varepsilon$, for all $i \geqslant m(s, t)$, and $v_i \neq t$,

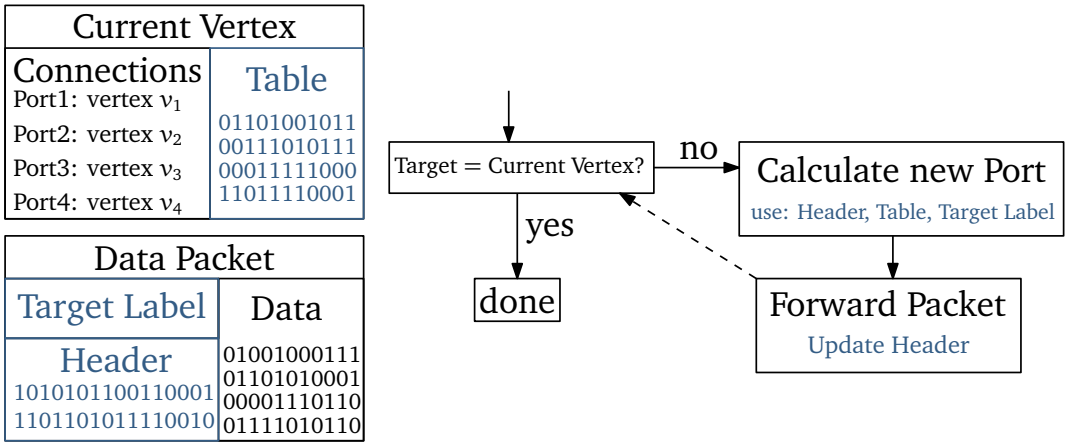


Figure 2.5: The routing model.

for all $i \in [m(s,t)-1]$. If $\mathcal{R}$ is correct for G, then $\delta_G(s,t) = \sum_{i=1}^{m(s,t)} w(v_{i-1}, v_i)$ is called the *routing length* between s and t in G.

The measures. Let $\mathcal{G}(n)$ be the set of all graphs of $\mathcal{G}$ with exactly n vertices. Let $\mathcal{R}$ be a correct routing scheme for all $G \in \mathcal{G}$. We want to present the main measures that are used to compare different routing schemes.

First of all the sizes of the labels and the tables have to be small, since we want to use a small amount of storage for the tables in the nodes and for the labels in the static header. The *routing label size* $L(n)$ and the *routing table size* $T(n)$ are formally defined as

$$L(n) = \max_{G \in \mathcal{G}(n)} \max_{v \in V_G} |\mathrm{lab}_G(v)|,$$
$$T(n) = \max_{G \in \mathcal{G}(n)} \max_{v \in V_G} |\mathrm{tab}_G(v)|.$$

We measure the size of the labels and tables, resp., by maximizing over all $v \in V_G$. Instead it is also possible to measure the size by summing over all $v \in V_G$. This measure is usually used for routing schemes, where a lot of nodes have small labels, resp. tables, and only a few vertices store huge amounts. Moreover, this measure is often used for lower bound results.

The *stretch* relates the length of the routing path to the length of the shortest path. It is formally defined as

$$S(n) = \max_{G \in \mathcal{G}(n)} \max_{s \neq t \in V_G} \frac{\delta_G(s,t)}{d_G(s,t)}.$$

Next, the size of the header should be as small as possible. It can be measured as follows. Let $h_i(s,t)$ be the header of the i-th routing step from s to t. The *header size* is defined as

$$H(n) = \max_{G \in \mathcal{G}(n)} \max_{s,t \in V_G} \max_{i \in [m(s,t)-1]} |h_i(s,t)|.$$

The objective is, to design routing schemes that minimize all four values at the same time. However, the computation of labels and tables occurs in a preprocessing step. This *preprocessing time* should be as small as possible.

Routing in trees. We give a short overview for routing in trees. Routing in trees will play an important role in the routing schemes discussed in this thesis. There are many different such schemes, based on similar ideas. The following two lemmas are due to Fraigniaud and Gavoille [FG01] as well as Thorup and Zwick [TZ01]. Both lemmas appear in both articles.

Lemma 2.2. *Let T be an n-vertex tree with arbitrary edge weights. In the fixed-port model, there is a routing scheme for T with label and table size $O(\log^2 n / \log\log n)$ whose routing function σ_{tree} sends a data packet along a shortest path, for any pair of vertices. The preprocessing time of the labels, resp. tables, is $O(n \log n)$ and the header is not used.*

Lemma 2.3. *Let T be an n-vertex tree with arbitrary edge weights. In the designer-port model, there is a routing scheme for T with label and table size $O(\log n)$ whose routing function sends a data packet along a shortest path, for any pair of vertices. The preprocessing time of the labels, resp. tables, is $O(n \log n)$ and the header is not used.*

Both routing schemes are asymptotically optimal. More precisely, for each tree routing scheme in the fixed-port model, that achieves stretch 1, there exists an n-vertex tree T and a vertex of T that needs $\Omega(\log^2 n / \log\log n)$ bits in the label, resp. table. This was proven by Fraigniaud and Gavoille [FG02]. A similar statement holds for the designer-port model. However, the designer-port routing scheme for trees is not useful as a building block for more complex routing schemes, especially if we need to be able to route in several subtrees of the input graph: If a graph is covered by a set of trees and a vertex is contained in different trees, we need a lookup table that maps the virtual ports of the different trees to the physical port numbers. In fact, we only use Lemma 2.2.

I

Routing

Part Outline

In this part of the thesis we give the details of routing schemes for three different graph classes: visibility graphs of simple histograms, visibility graphs of double histograms and intersection graphs of unit disks.

Definition 2.4 (Histogram)**.** *A* simple histogram P *is an* x*-monotone orthogonal polygon where the upper boundary consists of exactly one horizontal edge. The upper horizontal edge is called* base edge*, its endpoints are called* base vertices*; see Figure 2.6, left. A* double histogram P *is an* x*-monotone orthogonal polygon with a* base line*, a horizontal line segment whose relative interior lies in the interior of* P *and whose left and right endpoints are on the left and right boundary edge of* P*; see Figure 2.6, right.*

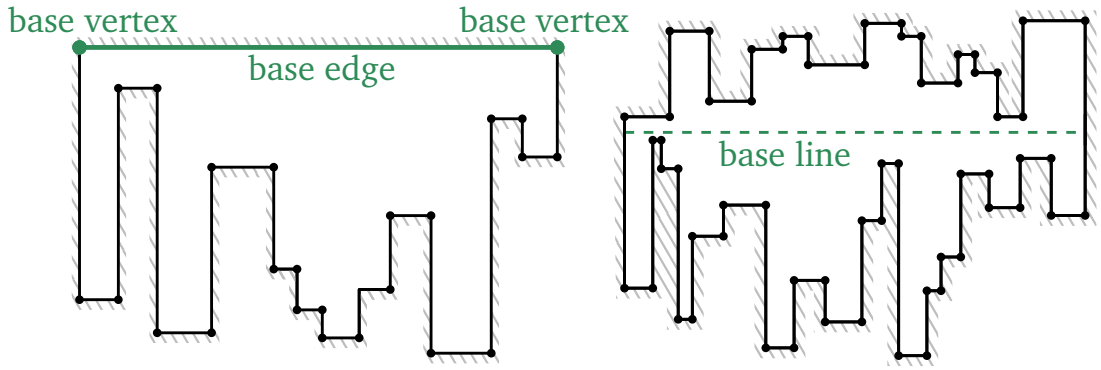


Figure 2.6: Left: Simple Histogram. Right: Double Histogram.

Let P be a simple or double histogram with n vertices. The *visibility graph* $\mathrm{Vis}(P)$ has vertex set $V(P)$ and there is an edge between two vertices $v, w \in V(P)$ if and only if $\square(v, w) \subseteq P$. The visibility graph has unit weights. Let V be a finite set of points in the Euclidean plane. The *unit disk graph* $\mathrm{DG}(V)$ has vertex set V and there is an edge between $v, w \in V$ if and only if $|vw| \leqslant 1$. The unit disk graph has Euclidean weights.

First, we present a routing scheme for visibility graphs of simple histograms and subsequently a routing scheme for visibility graphs of double histograms. Both routing schemes do not use any known routing schemes as building blocks. Hence, the insight of these two chapters lies in the combinatorial structure of the shortest paths in these types of polygons.

Secondly, we present a routing scheme for unit disk graphs on the edges. In contrast to the histograms, we use a lot of known techniques and data structures and combine them to an efficient routing scheme for unit disk graphs. The main building block of this scheme is the *shortest-path separator decomposition by* Mikkel Thorup [Tho04]. There is a complete chapter for the description of this decomposition, but it also includes a brief outline of covers and spanners.

CHAPTER 3

Simple Histograms

In this chapter we present an efficient routing scheme for visibility graphs of *simple histograms* with r-visibility assuming hop distance.

Let P be an n-vertex simple histogram and $\mathrm{Vis}(P)$ the visibility graph of its vertices assuming r-visibility. The edges have unit weights. We use V to denote the vertex set $V(P)$ of P. Hence, $|V| = n$. To make the discussion easier, we assume[1] that each vertex $v \in V$ has a unique *identifier* $v_{\mathrm{id}} \in [n-1]$, where the left base vertex has index 0 and the indices increase counterclockwise along the boundary. Thus, the right base vertex has index $n-1$. We assume that no three vertices are on a horizontal or vertical line. The chapter is organized as follows: first of all we present some notions relating to simple histograms. After that, we analyze the structure of shortest paths in $\mathrm{Vis}(P)$. Last but not least, we present the routing scheme.

3.1 Landmarks

We want to understand how shortest paths in P behave. Therefore, we associate with each $v \in V$ three landmark vertices in P; see Figure 3.1. Whenever we compute the minimum or maximum of a set of tuples, we use lexicographic order.

First, the *corresponding vertex* of v, $\mathrm{cv}(v)$, is the unique vertex within the same horizontal edge as v. Next, the *right vertex* of v, $r(v)$, is the lexicographically largest vertex of $N(v)$, i.e.,

$$r(v) = \operatorname{argmax}\{(w_x, w_y) \mid w \in N(v)\}.$$

The geometrical interpretation of $r(v)$ is as follows: we shoot a rightward ray r from v. Let e be the vertical edge where r first hits the boundary of P. The vertex $r(v)$ is the endpoint of e closer to the base edge. In a similar way, we obtain the *left vertex* $\ell(v)$ of v, by shooting the horizontal ray to the left, i.e.,

$$\ell(v) = \operatorname{argmin}\{(w_x, -w_y) \mid w \in N(v)\}.$$

[1] We can drop the assumption by performing a simple scan of the graph in time $O(\text{number of edges})$.

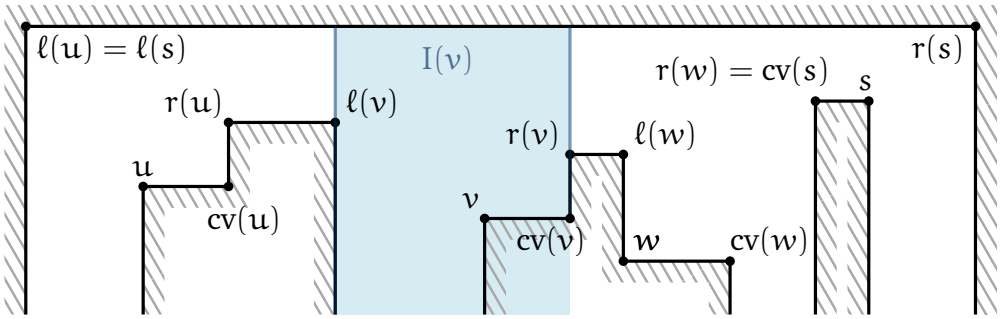


Figure 3.1: Left, right, and corresponding vertices. The interval $I(v)$ of v is the *set of vertices* between $\ell(v)$ and $r(v)$.

It is easy to see that $r(v) = \operatorname{argmax}\{w_{id} \mid w \in N(v)\}$ and $\ell(v) = \operatorname{argmin}\{w_{id} \mid w \in N(v)\}$. The following observation shows the relationship between the indices and the x-coordinates of the vertices.

Observation 3.1. *If* p *is either an* r*-reflex vertex or the left base vertex and* q *is either* ℓ*-reflex or the right base vertex, then* $[p, q] = \{v \in V \mid p_{id} \leqslant v_{id} \leqslant q_{id}\}$.

Additionally, the *interval of a vertex* v, $I(v)$, is the interval of the left and right vertex of v, that is $I(v) = [\ell(v), r(v)]$. Every vertex visible from v is in $I(v)$, i.e., $N(v) \subseteq I(v)$. This interval will be crucial in our routing schemes and gives a very useful characterization of visibility in simple histograms.

The previous definitions give notions for different landmarks relative to one vertex. To characterize shortest paths we have to take a look at some important landmarks that depend on both, the start and the target vertex; see Figure 3.2. Let s and t be two vertices. The following definitions can only be applied for the case that $t \in I(s) \setminus N(s)$. We assume that t lies strictly to the right of s, the other case is symmetric. The *near dominator* $\operatorname{nd}(s, t)$ of t with respect to s is the rightmost vertex in $N(s)$ to the left of t. If there is more than one such vertex, $\operatorname{nd}(s, t)$ is the vertex closest to the base edge. To be precise:

$$\operatorname{nd}(s, t) = \operatorname{argmax}\{(w_x, w_y) \mid w \in N(s), w_x \leqslant t_x\}.$$

In contrast to that, the *far dominator* $\operatorname{fd}(s, t)$ of t with respect to s is the leftmost vertex in $N(s)$ but to the right of t, i.e.,

$$\operatorname{fd}(s, t) = \operatorname{argmin}\{(w_x, -w_y) \mid w \in N(s), t_x \leqslant w_x\}.$$

The interval $I(s, t) = [\operatorname{nd}(s, t), \operatorname{fd}(s, t)]$ contains all the vertices between (and including) the near and far dominator. Specifically, it is $t \in I(s, t)$ by definition.

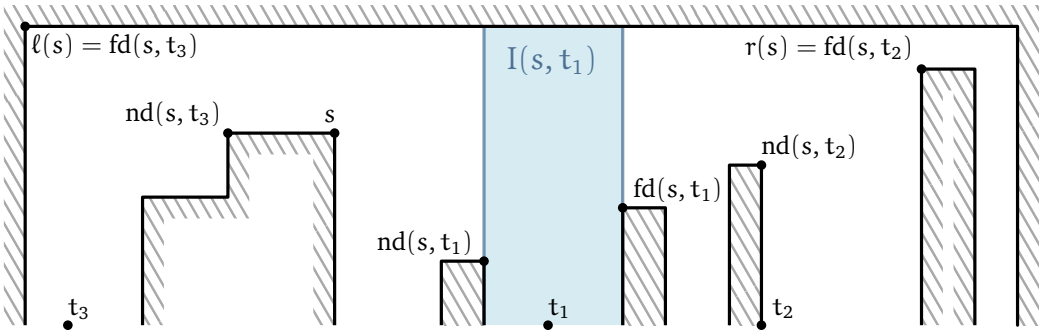


Figure 3.2: The near and the far dominators, as well as the corresponding interval $I(s, t_1)$.

3.2 Structural Insights

Next, we give several characterizations of visibility in simple histograms and analyze how the shortest paths between vertices behave.

Observation 3.2. *Let $v \in V$ be r-reflex (resp., ℓ-reflex) or the left (resp., right) base vertex, and let $A = [v, r(v)]$ (resp., $A = [\ell(v), v]$). Then, for all vertices $u \in A \setminus \{v, r(v), \ell(v)\}$ we have $I(u) \subseteq A$.*

Proof. Let v be r-reflex or the left base vertex. Assume for the sake of contradiction, that $\ell(u)$ or $r(u)$ is outside of A. Then, u must have a larger y-coordinate than v. It follows that v cannot see $r(v)$. This contradicts the definition of $r(v)$. Hence, we have $\ell(u), r(u) \in I$ and therefore, $I(u) \subseteq A$. If v is ℓ-reflex or the right base vertex, the proof is symmetric. □

Observation 3.3. *Let $v \in V$ be a left (resp., right) vertex distinct from the base vertex. Then, v can see exactly two vertices to its right (resp., left), namely $\mathrm{cv}(v)$ and $r(v)$ (resp., $\ell(v)$).*

Proof. Suppose that v is a left vertex; the other case is symmetric. Any vertex visible from v to the right of v lies in in $[\mathrm{cv}(v), r(v)]$. If $\mathrm{cv}(v)$ is convex, the observation is immediate, since then $[\mathrm{cv}(v), r(v)] = \{\mathrm{cv}(v), r(v)\}$. Otherwise, $\mathrm{cv}(v)$ is r-reflex and $r(v) = r(\mathrm{cv}(v))$. By Observation 3.2, we get that for all $u \in [\mathrm{cv}(v), r(v)] \setminus \{\mathrm{cv}(v), r(v)\}$, we have $I(u) \subseteq [\mathrm{cv}(v), r(v)]$. Thus, $v \notin I(u)$ for any such u, and since $N(u) \subseteq I(u)$, v cannot see u. □

We analyze the (shortest) paths in a simple histogram. The following lemma identifies certain "bottleneck" vertices that appear on any path; see Figure 3.3.

Lemma 3.4. *Let $v, w \in V$ be co-visible vertices such that v is either r-reflex or the left base vertex and w is either ℓ-reflex or the right base vertex. Let s and t be two vertices with $s \in [v, w]$ and $t \notin [v, w]$. Then, any path between s and t includes v or w.*

Proof. Let $A = [v, w]$. Since $t \notin A$, not both v, w are base vertices. Thus, suppose without loss of generality that $v_y < w_y$. Then, $\mathrm{cv}(v)$ is a left vertex and can see w. Hence, Observation 3.3 implies that $r(v) = r(\mathrm{cv}(v)) = w$. By Observation 3.2, we get $N(u) \subseteq I(u) \subseteq A$, for any

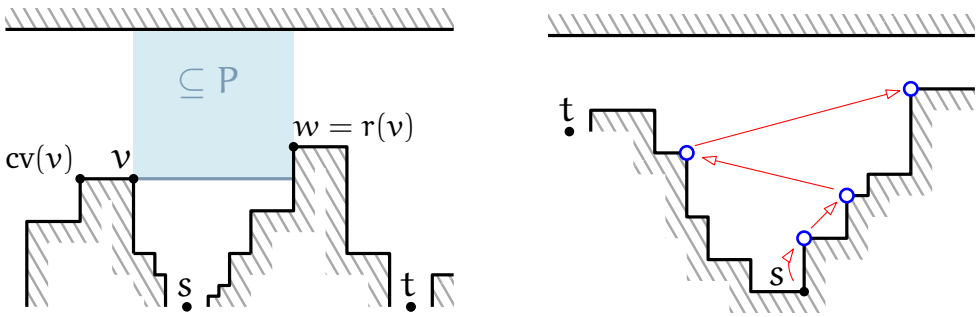


Figure 3.3: Left: Any path from s to t includes v or w, since the blue rectangle contains only v and w as vertices. Right: A shortest path from s to t using the highest vertex.

$u \in A \setminus \{v, w\}$. Thus, a neighbor of u cannot be outside of A. Therefore, the only two vertices of A that can see a vertex not in A are v and w. Hence, any path between s and t must include v or w. □

An immediate consequence of Lemma 3.4 is that if $t \notin I(s)$, then any path from s to t uses $\ell(s)$ or $r(s)$. The next lemma shows that if $t \notin I(s)$, there is a shortest path from s to t that uses the higher vertex of $\ell(s)$ and $r(s)$, see Figure 3.3.

Lemma 3.5. *Let s and t be two vertices with $t \notin I(s)$. If $\ell(s)_y > r(s)_y$ (resp., $\ell(s)_y < r(s)_y$), then there is a shortest path from s to t using $\ell(s)$ (resp., $r(s)$).*

Proof. Assume $\ell(s)_y > r(s)_y$, the other case is symmetric. Let $\pi : \langle s = p_0, \ldots, p_k = t \rangle$ be a shortest path from s to t. If π contains $\ell(s)$, we are done. Otherwise, by Lemma 3.4, there is a $0 < j < k$ with $p_j = r(s)$ and $p_i \neq r(s)$, for $i > j$. Thus, $p_{j+1} \notin I(s)$. Since we assumed $\ell(s)_y > r(s)_y$, it follows that $\ell(p_j) = \ell(r(s)) = \ell(s)$, so p_{j+1} must be to the right of p_j. Therefore, by Observation 3.3, we can conclude that $p_{j+1} \in \{cv(p_j), r(p_j)\}$. Now, since $\ell(s)$ is higher than $r(s)$, it can also see $cv(p_j)$ and $r(p_j)$, in particular, it can see p_{j+1}. Hence, $\langle s, \ell(s), p_{j+1}, \ldots, p_k \rangle$ is a valid path of length at most $|\pi|$, so there exists a shortest path from s to t through $\ell(s)$. □

The next lemma considers the case where t is in $I(s)$. Then, the near and far dominator are the potential vertices that lie on a shortest path from s to t (see also Fig. 3.5).

Lemma 3.6. *Let s and t be two vertices with $t \in I(s) \setminus N(s)$. Then, $nd(s,t)$ is reflex. Moreover, if t is to the left of s then $fd(s,t) = \ell(nd(s,t))$, and if t is to the right of s then $fd(s,t) = r(nd(s,t))$.*

Proof. Without loss of generality, t lies strictly to the right of s. First, assume that $nd(s,t)$ is ℓ-convex. Since s can see $nd(s,t)$ and since $nd(s,t)$ is to the right of s, it follows that s and $nd(s,t)$ share the same vertical edge. Then, $cv(nd(s,t))$ is also visible from s and its horizontal distance to t is smaller. This contradicts the definition of $nd(s,t)$.

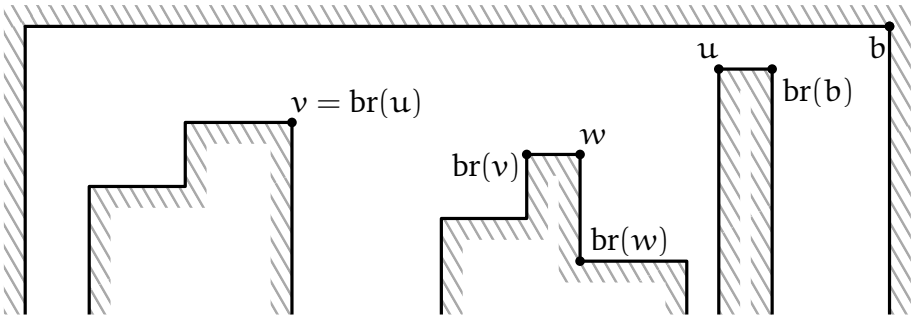


Figure 3.4: The breakpoints of some vertices.

Next, assume that $\mathrm{nd}(s,t)$ is r-convex. Let v be the reflex vertex sharing a vertical edge with $\mathrm{nd}(s,t)$. Then, $N(\mathrm{nd}(s,t)) \subseteq N(v)$ and $v \in N(s)$. Furthermore, since t is strictly to the right of v but still inside $I(s)$, the vertices v and $r(s)$ must be distinct. Thus, $v_y < s_y$, so that $\mathrm{cv}(v)$ is also visible from s. Moreover, the horizontal distance of $\mathrm{cv}(v)$ and t is smaller than the horizontal distance of $\mathrm{nd}(s,t)$ and t. This again contradicts the definition of $\mathrm{nd}(s,t)$. The first part of the lemma follows.

It remains to show that $\mathrm{fd}(s,t) = r(\mathrm{nd}(s,t))$. First of all, $\mathrm{fd}(s,t)$ is higher than $\mathrm{nd}(s,t)$, since otherwise $\mathrm{fd}(s,t)$ would not be visible from s. Moreover, if $\mathrm{nd}(s,t)$ and $\mathrm{fd}(s,t)$ are not co-visible, there must be a vertex v strictly between $\mathrm{nd}(s,t)$ and $\mathrm{fd}(s,t)$ that is visible from s and higher than $\mathrm{nd}(s,t)$. Now, either $t \in [\mathrm{nd}(s,t), v]$ or $t \in [v, \mathrm{fd}(s,t)]$. In the first case, the horizontal distance between v and t is smaller than between t and $\mathrm{fd}(s,t)$, and in the second case, the horizontal distance between v and t is smaller than between t and $\mathrm{nd}(s,t)$. Either case leads to a contradiction. Therefore, $\mathrm{fd}(s,t)$ is higher than $\mathrm{nd}(s,t)$, strictly to the right of $\mathrm{nd}(s,t)$ and visible from $\mathrm{nd}(s,t)$. Thus, Observation 3.3 gives $\mathrm{fd}(s,t) = r(\mathrm{nd}(s,t))$. □

3.3 The Routing Scheme

We now describe our routing scheme and prove that it gives a shortest path. The idea for our routing scheme is as follows: as long as the target vertex t is not in the interval $I(s)$ of the current vertex s, i.e., as long as there is a higher vertex that blocks visibility between s and t, we have to leave the interval of s as fast as possible. Once $t \in I(s)$, we have to find the interval containing t.

The labels and routing tables. Let $v \in V$. It is $\mathrm{lab}(v) = v_{\mathrm{id}}$. For the table of v, let $w \in N(v)$ be one of its neighbors. If w is convex, we store w_{id} and the port number p_w that points from v to w in $\mathrm{tab}(v)$. Otherwise, suppose that w is an r-reflex vertex or the left base vertex. The *breakpoint of* w, $\mathrm{br}(w)$, is defined as the left endpoint of the horizontal edge with the highest y-coordinate to the right of and below w that is visible from w; analogous definitions apply to ℓ-reflex vertices and the right base vertex; see Figure 3.4. We store $(w_{\mathrm{id}}, \mathrm{br}(w)_{\mathrm{id}}, p_w)$ in

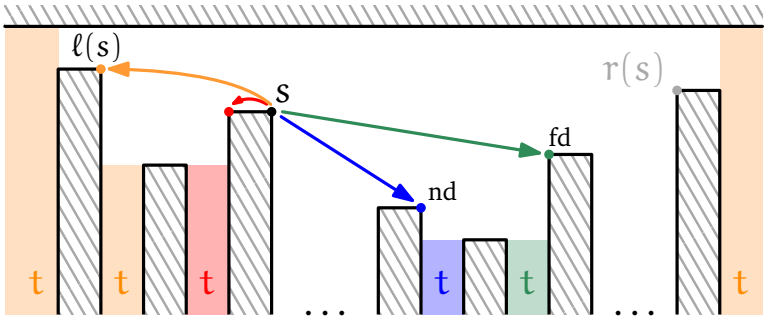


Figure 3.5: The cases where the vertex t lies and the corresponding vertices where the data packet is sent to. If $t \in [\ell(s), s]$ we have $\mathrm{nd}(s,t) = \mathrm{cv}(s)$ and $\mathrm{fd}(s,t) = \ell(s)$.

the table of v. Again, p_w is the port number pointing from v to w. Finally, we store $\mathrm{lab}(v)$ in $\mathrm{tab}(v)$. Therefore, we need exactly $\lceil \log n \rceil$ bits for the label and at most $3 \cdot \log n \cdot (\deg(v)+1)$ bits for the table of v.

The routing function. We are given the table $\mathrm{tab}(s)$ of the current vertex s and the label $\mathrm{lab}(t)$ of the target vertex t. The routing function does not need a header, i.e., $H(n) = 0$; refer to Algorithm 3.1. If $t = s$, i.e., if $\mathrm{lab}(t) = \mathrm{lab}(s)$, we are done. Next, if t is visible from s, i.e., if t_{id} is in $\mathrm{tab}(s)$, we directly go from s to t on a shortest path by using the port number stored in $\mathrm{tab}(s)$. Thus, assume that t is not visible from s. First, we check if $t \in I(s)$. This is done as follows: we determine the smallest and largest id in the table $\mathrm{tab}(s)$ of s (ignoring the id's of the breakpoints). The corresponding vertices are $\ell(s)$ and $r(s)$. Then, we can check if $t_{\mathrm{id}} \in [\ell(s)_{\mathrm{id}}, r(s)_{\mathrm{id}}]$, which is the case if and only if $t \in I(s)$. Now, there are two cases, illustrated in Figure 3.5. First, suppose $t \notin I(s)$. We have to determine which one of $\ell(s)$ and $r(s)$ is higher. For this we look at the breakpoints. If one of the two has no breakpoint entry in $\mathrm{tab}(s)$, it has to be a convex vertex. This means the corresponding vertex is a base vertex. We take a hop to that base vertex, since no other vertex can be higher. Hence, assume that both vertices are stored together with the indices of their breakpoints. If the breakpoint of $r(s)$ is to the right of $\ell(s)$, i.e., $\ell(s)_{\mathrm{id}} < \mathrm{br}(r(s))_{\mathrm{id}}$, the vertex $\ell(s)$ is higher. We take the hop to that one. Finally, if otherwise $\mathrm{br}(\ell(s))_{\mathrm{id}} < r(s)_{\mathrm{id}}$, the vertex $r(s)$ is higher. We take the hop to $r(s)$. However, in each case we took the hop to the highest vertex in $\{\ell(s), r(s)\}$. By Lemma 3.5, this hop lies on a shortest path from s to t.

Second, suppose that $t \in I(s) \setminus N(s)$. This case is slightly more involved. We use the table $\mathrm{tab}(s)$ of s and the label $\mathrm{lab}(t)$ of t to determine $\mathrm{fd}(s,t)$ and $\mathrm{nd}(s,t)$. Again, we can do this by comparing the id's. Lemma 3.6 states that either $\mathrm{fd}(s,t) = \ell(\mathrm{nd}(s,t))$ or $\mathrm{fd}(s,t) = r(\mathrm{nd}(s,t))$. We discuss the case that $\mathrm{fd}(s,t) = r(\mathrm{nd}(s,t))$, the other case is symmetric. By Lemma 3.4, any shortest path from s to t includes $\mathrm{fd}(s,t)$ or $\mathrm{nd}(s,t)$. Moreover, due to Lemma 3.6, $\mathrm{nd}(s,t)$ is reflex, and we can use $\mathrm{tab}(s)$ to access $b_{\mathrm{id}} = \mathrm{br}(\mathrm{nd}(s,t))_{\mathrm{id}}$. The vertex b splits $I(s,t) = [\mathrm{nd}(s,t), \mathrm{fd}(s,t)]$ into two disjoint subintervals $[\mathrm{nd}(s,t), b]$ and $[\mathrm{cv}(b), \mathrm{fd}(s,t)]$. Also, b and $\mathrm{cv}(b)$ are not visible from s, as they are located strictly between the far and the

Algorithm 3.1 The routing function for simple histograms with stretch 1.

```
1:  procedure RoutingFunction(tab(s) ∈ Σ*, lab(t) ∈ Σ*)
2:      Outputs: port p ∈ Σ*
3:      if t = s then return ε
4:      if t ∈ N(s) then return p_t
5:      if t ∉ [ℓ(s), r(s)] then
6:          if ℓ(s)_id = 0 then return p_ℓ(s)          ▷ ℓ(s) is left base vertex.
7:          if r(s)_id = n − 1 then return p_r(s)      ▷ r(s) is right base vertex.
8:          if ℓ(s)_id < br(r(s))_id then return p_ℓ(s)   ▷ ℓ(s) is higher than r(s).
9:          else return p_r(s)                         ▷ r(s) is higher than ℓ(s).
10:     b = br(nd(s, t))
11:     if t ∈ I(s) and t_id < s_id then
12:         if t ∈ [b, nd(s, t)] then return p_nd(s,t)
13:         else return p_fd(s,t)
14:     if t ∈ I(s) and s_id < t_id then
15:         if t ∈ [nd(s, t), b] then return p_nd(s,t)
16:         else return p_fd(s,t)
```

near dominator. Based on b_{id}, we can now decide on the next hop.

If $t \in [nd(s,t), b]$, we take the hop to $nd(s,t)$. If $t = b$, our packet uses a shortest path of length 2. Thus, assume that t lies between $nd(s,t)$ and b. This is only possible if b is ℓ-reflex, and we can apply Lemma 3.4 to see that any shortest path from s to t includes $nd(s,t)$ or b. But since $d(s,b) = 2$, our data packet routes along a shortest path.

If $t \in [cv(b), fd(s,t)]$, we take the hop to $fd(s,t)$. If $t = cv(b)$, our packet uses a shortest path of length 2. Thus, assume that t lies between $cv(b)$ and $fd(s,t)$. This is only possible if $cv(b)$ is r-reflex, so we can apply Lemma 3.4 to see that any shortest path from s to t uses $fd(s,t)$ or $cv(b)$. Since $d(s, cv(b)) = 2$, our packet routes along a shortest path. Thus, we can conclude that our routing function routes the data packet along a shortest path.

The preprocessing time. Let m be the number of edges of $Vis(P)$. If we want to compute the labels and tables, we have to compute for every vertex $v \in V$ the two vertices $\ell(v)$ and $r(v)$. Moreover, for every reflex vertex v we need to compute its breakpoint $br(v)$. Since the output consists of $\Theta(m \log n)$ bits[2], the overall preprocessing time has to be at least $\Omega(m)$. First, we provide an algorithm that computes $\ell(v)$, $r(v)$, and $br(v)$, for all $v \in V$, in $O(n)$ time. The routing tables and labels can then be computed in $O(m)$ time.

First, we present an algorithm that computes $r(v)$ for every vertex v; see Algorithm 3.2 for the details and Figure 3.6 for an illustration. The algorithm works as follows: we start at the left base vertex, i.e., the vertex with $id = 0$, and push it onto an empty stack. Then we walk counterclockwise along the boundary. Whenever we encounter a right vertex (a vertex with

[2]We assume that $\Theta(\log n)$ bits can be written in constant time.

Algorithm 3.2 Computes $r(v)$ for all v.

```
1:  procedure ComputeRightVertices(simple histogram P)
2:      S = new empty stack
3:      id = 0
4:      while id < n do
5:          S.push(v_id)                    ▷ v_id is a right vertex or the left base vertex
6:          id = id + 1
7:          w = S.top()
8:          while w.y ⩽ v_id.y do           ▷ v_id is a left vertex or the right base vertex
9:              w.r = cv(w).r = v_id        ▷ v_id is the right vertex of w and cv(w)
10:             S.pop()
11:             if S = ∅ then return
12:             w = S.top()
13:         id = id + 1
```

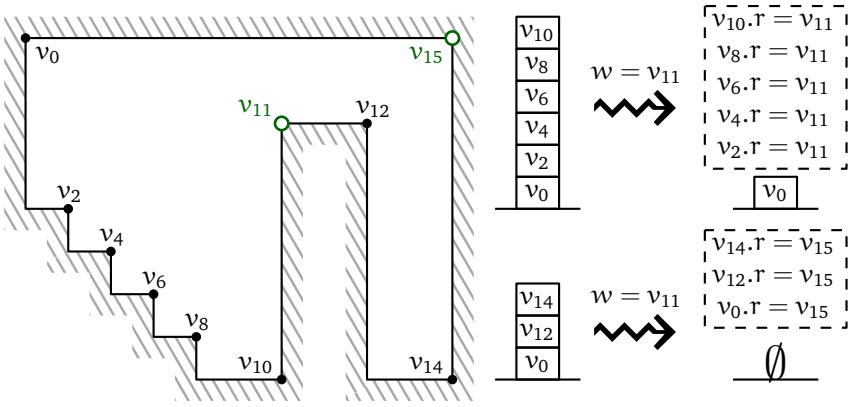


Figure 3.6: This figure shows the process how the right vertices are computed.

even id) we push that vertex onto the stack. Once we encounter a vertex v with odd index – v is either a left vertex or the right base vertex – we pop a vertex w from the stack as long as v is higher than the popped vertex. In this case, we set $w.r$ and $cv(w).r$ to v. This process continues until we encounter the right base vertex. We claim the following lemma.

Lemma 3.7. *Algorithm 3.2 is correct. More precisely, upon the termination of the algorithm we have $v.r = r(v)$ for every vertex v.*

Proof. First of all, it is obvious that the algorithm proceeds correctly for the left and right base vertex. Next, we claim that for every right vertex w on the lower part of the boundary the algorithm satisfies $r(w) = w.r$. Once we have proved the claim, we can immediately conclude the correctness of the lemma, since for every left vertex w on the lower part of the boundary we have $r(w) = r(cv(w)) = cv(w).r = w.r$ by construction.

For the proof of the claim let w be a right vertex on the lower part of the boundary and

let v be the reason why w is removed from the stack. Assume for the sake of contradiction that $v \neq r(w)$. Since $r(w)$ is the leftmost vertex on the boundary that is higher than w (but still to its right), v must be strictly to the right of $r(w)$. Since w has not been removed from the stack when $r(w)$ has been processed, there must be a vertex w' on the stack satisfying $w'_y \geqslant r(w)_y > w_y$. Since this vertex appears in the stack higher than w it must lie on the boundary strictly between w and $r(w)$. This contradicts the definition of $r(w)$. Hence, $v = r(w)$. This finishes the proof. □

Next, we have to analyze the running time. There are $n/2$ vertices that are pushed onto the stack. Hence, the inner `while`-loop can only make a linear number of runs in total. Every basic operation is constant. Thus, the total running time is in $O(n)$. Moreover, we can compute $\ell(v)$ for every v using the same procedure but walking counterclockwise around the boundary starting at the right base vertex. This costs linear time as well.

Last but not least, we have to compute the break points for all reflex vertices. We focus attention on an r-reflex vertex w and observe that its breakpoint $br(w)$ is the rightmost left vertex v satisfying $\ell(v) = w$. Hence, Algorithm 3.3 computes the breakpoints in linear time. Once, ℓ, r and br have been computed, the routing table of a vertex v can be computed in $O(\deg(v))$ time. Therefore, we can conclude with our first theorem.

Algorithm 3.3 Computes $br(v)$ for all v.

```
1: procedure ComputeBreakPoints(simple histogram P)
2:     for id = 0, ..., n − 1 do
3:         if v_id is a left vertex then
4:             br(ℓ(v_id)) = v_id
5:     for id = n − 1, ..., 0 do
6:         if v_id is a right vertex then
7:             br(r(v_id)) = v_id
```

Theorem 3.8. *Let P be a simple histogram with n vertices and m edges in $\mathrm{Vis}(P)$. There is a routing scheme for $\mathrm{Vis}(P)$ without header, label size $\lceil \log n \rceil$, and each vertex v obtains a routing table of size $\Theta(\log n \deg(v))$. The labels and tables can be computed in time $O(m)$ and we can route between any two vertices on a shortest path.*

Remark. In a slightly different routing model, we distinguish two parts in the table of a vertex v. The first one is called *link table*. It saves the labels and ports of all neighbors $N(v)$. Technically, some version of this table always exists, since otherwise we were not able to distinguish the physical ports pointing to the neighbors. The second part in this context is then called *routing table*. It contains information of the remaining topology of the graph. Turning our routing scheme for simple histograms into this model gives us a link table of size $\Theta(\log n \deg(v))$ but the routing table of v would only contain the label of v; see [CCK+20].

CHAPTER 4

Double Histograms

In this chapter we present an efficient routing scheme for visibility graphs of *simple histograms* with r-visibility assuming hop distance.

Let P be an n-vertex double histogram and $\mathrm{Vis}(P)$ the visibility graph of its vertices assuming r-visibility. The edges have unit weights. We use V to denote the vertex set $V(P)$ of P. Hence, $|V| = n$. We assume general position, i.e., no three vertices are on a vertical or horizontal line. The chapter is organized as follows: first of all we present some notions used to address properties of double histograms. After that we analyze the structure of shortest paths in the visibility graph. Last but not least, we present the routing scheme.

4.1 Landmarks

In this section we assume that the baseline lies on the x-axis. This does not influence any properties but makes the presentation easier. Two vertices v, w in V are *on the same side* if both are below or above the base line, i.e., if $v_y \cdot w_y > 0$. Next, for a vertex v of P, we want to define the vertices $\ell(v)$, $r(v)$, and $\mathrm{cv}(v)$ as well as the interval $I(v)$. The definition of $\mathrm{cv}(v)$ is transferable from simple to double histograms. Hence, $\mathrm{cv}(v)$ is the unique vertex that shares the same horizontal edge with v and is again called *corresponding vertex* of v. If we define $\ell(v)$ and $r(v)$ as before, we run into the problem that in some cases $\ell(v)$ and $r(v)$ might not be on the same side of the base line. We solve this problem by relaxing the definition in the following way: we do not require $\ell(v)$ and $r(v)$ to be vertices. This leads to the following definition of the *left point* $\ell(v)$ of v: we shoot a leftward horizontal ray r from v. Let e be the vertical edge where r first hits the boundary of P, the intersection of r and e is the leftward projection $\mathrm{pr}_\ell(v)$ of v. If e is the left boundary of P; then $\ell(v) = \mathrm{pr}_\ell(v)$. If e is not the left boundary of P, we let $\ell(v)$ be the endpoint of e closer to the base line. The formal definition of $\ell(v)$ is as follows:

$$\ell(v) = \operatorname{argmin}\left\{(w_x, |w_y|) \mid w \in N(v) \cup \{\mathrm{pr}_\ell(v)\}, v_y \cdot w_y > 0\right\}.$$

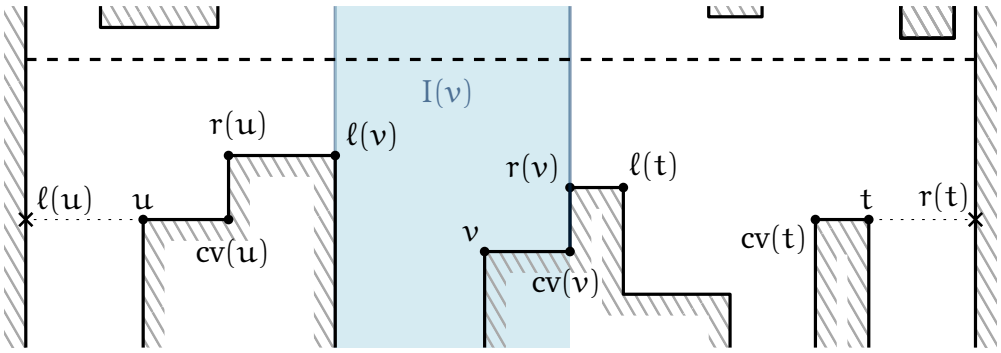


Figure 4.1: Left and right points, the corresponding vertex, and the near and far dominators. The interval $I(v)$ of v is the *set of vertices* between $\ell(v)$ and $r(v)$. The dashed line is the base line.

The *right vertex* $r(v)$ of v is defined analogously, by shooting the ray r rightwards until it hits the boundary in the point $\mathrm{pr}_r(v)$. The formal definition is

$$r(v) = \operatorname{argmax}\left\{(w_x, -|w_y|) \mid w \in N(v) \cup \{\mathrm{pr}_r(v)\}, v_y \cdot w_y > 0\right\}.$$

The three landmark vertices are illustrated in Figure 4.1. Finally, the *interval of a vertex* v, $I(v)$, is the interval of the left and right point of v, $I(v) = [\ell(v), r(v)]$. Moreover, we get the property we wanted, i.e., every vertex visible from v is in $I(v)$, that is, $N(v) \subseteq I(v)$.

Last but not least, we want to define the far and the near dominator like we did in simple histograms. We should recap the intuition of the near and the far dominator. Let s and t be two vertices with $t \in I(s) \setminus N(s)$ and assume t is to the right of s. The definitions for the case that t is to the left of s are symmetric. In simple histograms the near and the far dominator are the two vertices visible from s whose x-coordinate are closest to t_x. The near dominator is located before t and the far dominator is located behind t, relative to s. If we try to define the far dominator as before, like

$$\mathrm{fd}(s,t) = \operatorname{argmin}\left\{(w_x, -w_y) \mid w \in N(s), t_x \leqslant w_x\right\},$$

we might get the problem, that there is no vertex w in $N(s)$ satisfying $t_x \leqslant w_x$; as we can see in Figure 4.2. As before we solve this problem by not forcing the far dominator to be a vertex. In contrast to the left and right points we do not care about the side of the dominators: they might be on the same side or on different sides. Summarized, the two definitions of the dominators are as follows. The *near dominator* $\mathrm{nd}(s,t)$ of t with respect to s is the rightmost vertex in $N(s)$ to the left of t. If there is more than one such vertex, $\mathrm{nd}(s,t)$ is the vertex closest to the base line (the y-coordinate is closer to 0). To be precise we present the formal definition:

$$\mathrm{nd}(s,t) = \operatorname{argmax}\left\{(w_x, -|w_y|) \mid w \in N(s), w_x \leqslant t_x\right\}.$$

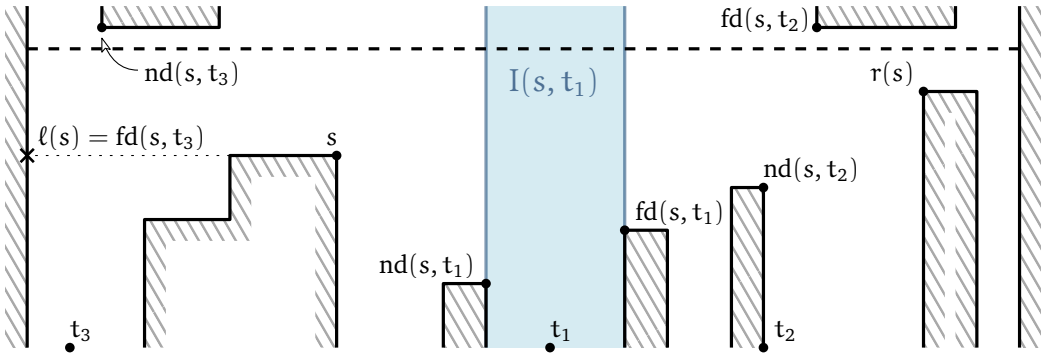


Figure 4.2: The near and the far dominators. Observe that $\mathrm{fd}(s, t_3)$ is not a vertex.

We have to argue why the near dominator is well-defined, i.e., why there always is a vertex w visible from s that is to the left of t. If t is not visible from s then there has to be some object between s and t blocking the visibility. But then there has to be at least one vertex between s and t visible from s. Hence, the near dominator is well-defined. For the far dominator we remember that $r(s)$ might not be a vertex but a projection of s onto the right boundary of P. The *far dominator* $\mathrm{fd}(s, t)$ of t with respect to s is the leftmost point in $N(s) \cup \{r(s)\}$ to the right of t. If there is more than one such point, $\mathrm{fd}(s, t)$ is the vertex closest to the base line. As formula we have

$$\mathrm{fd}(s, t) = \operatorname{argmin}\big\{(w_x, |w_y|) \mid w \in N(s) \cup \{r(s)\}, t_x \leqslant w_x\big\}.$$

Again, we argue that the far dominator is well-defined. We assumed t to be in $I(s)$. Hence, $t_x \leqslant r(s)_x$ and therefore, the definition makes sense.

Like in simple histograms, the interval $I(s, t) = \big[\mathrm{nd}(s, t), \mathrm{fd}(s, t)\big]$ contains all vertices between the near and far dominator, especially the vertex t. Figure 4.2 shows the dominators and the interval.

4.2 Structural Insights

Let P be a double histogram. Again, we assume that the base line lies on the x-axis. Let s and t be two vertices. To understand shortest paths in double histograms between s and t, we distinguish three cases, depending on where t lies relative to s. First, if t is close, i.e., if $t \in I(s)$, we focus on the near and far dominators. Second, if $t \notin I(s)$ but there is a vertex v visible from s with $t \in I(v)$, then we can find a vertex on a shortest path from s to t. Third, if there is no visible vertex v from s such that $t \in I(v)$, we can apply our intuition from simple histograms: go as fast as possible towards the base line.

4.2.1 Visibility in Double Histograms

The structure of the shortest paths in double histograms can be much more involved than in simple histograms; in particular, Lemma 3.4 does not hold anymore. However, the following observations provide some structural insight that can be used for an efficient routing scheme.

Observation 4.1. *Two vertices v, w are co-visible if and only if $v \in I(w)$ and $w \in I(v)$.*

Proof. The forward direction is immediate, as co-visibility implies $v \in N(w) \subseteq I(w)$ and $w \in N(v) \subseteq I(v)$. For the backward direction, consider the rectangle $\square(v, w)$ spanned by v and w. Since $v \in I(w)$ and $w \in I(v)$, the upper and lower boundary of $\square(v, w)$ do not contain a point outside P. As P is a double histogram, this implies that the left and right boundary of $\square(v, w)$ also do not contain any point outside P. Since P has no holes, we can conclude that $\square(v, w) \subseteq P$. Hence, v and w are co-visible, as claimed. □

Observation 4.2. *Let a, b, c, and d be vertices in P with $a_x \leqslant b_x \leqslant c_x \leqslant d_x$. If $a \in I(c)$ and $d \in I(b)$, then b and c are co-visible.*

Proof. This follows immediately from Observation 4.1 and the fact that intervals are monotone. □

Observation 4.3. *The intervals form a laminar family, i.e., for any two vertices v and w on the same side of the base line, we have (i) $I(v) \cap I(w) = \emptyset$, (ii) $I(v) \subseteq I(w)$, or (iii) $I(w) \subseteq I(v)$.*

Proof. Suppose there are two vertices v and w on the same side of P with $\ell(v)_x < \ell(w)_x \leqslant r(v)_x < r(w)_x$. By Observation 4.2, $\ell(w)$ and $r(v)$ are co-visible. Since $\ell(w)$ and $r(v)$ are on the same side of P, either $r(v)$ cannot see any vertex to the left of $\ell(w)$ or $\ell(w)$ cannot see any vertex to the right of $r(v)$. This contradicts the fact that $\ell(v)$ and $r(v)$ as well as $\ell(w)$ and $r(w)$ must be co-visible. □

4.2.2 The target is close

Let s, t be two vertices with $t \in I(s) \setminus N(s)$. In contrast to simple histograms, $\mathrm{fd}(s, t)$ now might not be a vertex. Furthermore, $\mathrm{fd}(s, t)$ and $\mathrm{nd}(s, t)$ might be on different sides of the base line. In this case, Lemma 3.6 no longer holds. However, the next lemma establishes a visibility relation between them; see Figure 4.3.

Lemma 4.4. *The vertices $\mathrm{nd}(s, t)$ and $\mathrm{fd}(s, t)$ are co-visible.*

Proof. Without loss of generality, t is strictly to the right of s. Suppose for a contradiction that $r(\mathrm{nd}(s, t))$ is strictly left of $\mathrm{fd}(s, t)$. Then, we get $r(\mathrm{nd}(s, t)) \in I(s)$. Also, $s \in I(\mathrm{nd}(s, t)) \subseteq I(r(\mathrm{nd}(s, t)))$. Hence, by Observation 4.1, s can see $r(\mathrm{nd}(s, t))$. But then $r(\mathrm{nd}(s, t))$ is a vertex strictly between the near and far dominator visible from s, contradicting the choice of the dominators. Thus, $s_x \leqslant \mathrm{nd}(s, t)_x \leqslant \mathrm{fd}(s, t)_x \leqslant r(\mathrm{nd}(s, t))_x$, and Observation 4.2 gives the result. □

Lemma 4.5. *One of $\mathrm{nd}(s, t)$ or $\mathrm{fd}(s, t)$ is on a shortest path from s to t. If $\mathrm{fd}(s, t)$ is not a vertex, then $\mathrm{nd}(s, t)$ is on a shortest path from s to t.*

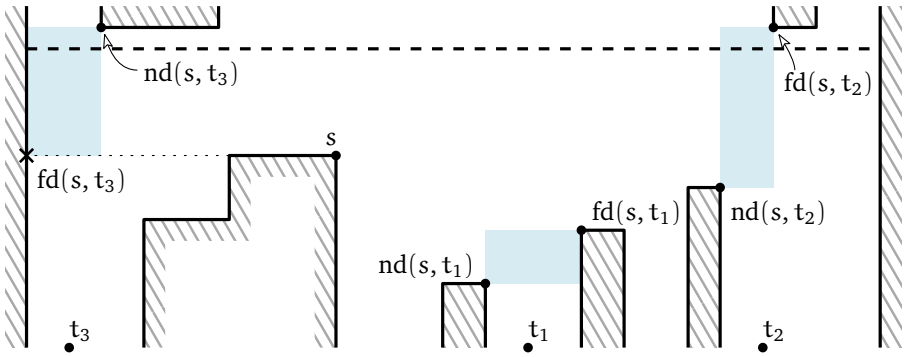


Figure 4.3: The far and the near dominator can see each other.

Proof. Without loss of generality, t is to the right of s. Let $\pi : \langle s = p_0, \ldots, p_k = t \rangle$ be a shortest path from s to t, and let p_j be the last vertex outside of $I(s,t)$. If $j = 0$, then p_{j+1} must be one of the dominators, since by definition they are the only vertices in $I(s,t)$ visible from s. Now, assume $j \geqslant 1$. If p_j is to the left of $\mathrm{nd}(s,t)$, we apply Lemma 4.4 and Observation 4.2 on the four points p_j, $\mathrm{nd}(s,t)$, p_{j+1}, and $\mathrm{fd}(s,t)$ to conclude that $\mathrm{nd}(s,t)$ can see p_{j+1}. Symmetrically, if p_j is to the right of $\mathrm{fd}(s,t)$, the same argument shows that the far dominator can see p_{j+1}. Thus, depending on the position of p_j we can exchange the subpath $p_1, \ldots, p_j$ in π by $\mathrm{nd}(s,t)$ or $\mathrm{fd}(s,t)$ and get a valid path of length $k - j + 1 \leqslant k$. The second part of the lemma follows because p_j cannot be to the right of $\mathrm{fd}(s,t)$, if $\mathrm{fd}(s,t)$ is not a vertex but a point on the right boundary. □

Next, we consider the case where $\mathrm{fd}(s,t)$ is a vertex but not on a shortest path from s to t. Then, $\mathrm{fd}(s,t)$ cannot see t, and we define $\mathrm{fd}^2(s,t) = \mathrm{fd}(\mathrm{fd}(s,t),t)$. By Lemma 4.4, $\mathrm{nd}(s,t)$ and $\mathrm{fd}(s,t)$ are co-visible, so $\mathrm{fd}^2(s,t)$ has to be in the interval $[\mathrm{nd}(s,t), t]$, and therefore it is a vertex. The following lemma states that $\mathrm{fd}^2(s,t)$ is strictly closer to t than s; see Figure 4.4.

Lemma 4.6. *If* $\mathrm{fd}(s,t)$ *is a vertex but not on a shortest path from* s *to* t*, then we have* $d(\mathrm{fd}^2(s,t),t) = d(s,t) - 1$.

Proof. Without loss of generality, t is to the right of s; see Figure 4.4. By Lemma 4.5, $\mathrm{nd}(s,t)$ lies on a shortest path from s to t. Let $\langle s = p_0, \mathrm{nd}(s,t) = p_1, p_2, \ldots, p_k = t \rangle$ be such a shortest path. We claim that $\mathrm{fd}^2(s,t)$ can see p_2. Then, $\langle \mathrm{fd}^2(s,t), p_2, \ldots, p_k = t \rangle$ is a valid path of length $k - 1 = d(s,t) - 1$. To prove that $\mathrm{fd}^2(s,t)$ can indeed see p_2, we show that $p_2 \in I\big(\mathrm{fd}^2(s,t)\big)$ and $\mathrm{fd}^2(s,t) \in I(p_2)$ and then apply Observation 4.1.

First, we show $p_2 \in I(\mathrm{fd}(s,t),t)$ by contradiction. Thus, suppose that $p_2 \notin I(\mathrm{fd}(s,t),t)$. Since $t \in I(\mathrm{fd}(s,t),t)$, there is a $j \geqslant 2$ with $p_{j+1} \in I(\mathrm{fd}(s,t),t)$ and $p_j \notin I(\mathrm{fd}(s,t),t)$. First, if $p_{j,x} < \mathrm{fd}^2(s,t)_x$, then $p_{j,x} < \mathrm{fd}^2(s,t)_x \leqslant p_{j+1,x} \leqslant \mathrm{nd}(\mathrm{fd}(s,t),t)_x$. By Lemma 4.4, $\mathrm{fd}^2(s,t)$ and $\mathrm{nd}(\mathrm{fd}(s,t),t)$ are co-visible, so Observation 4.2 implies that $\mathrm{fd}^2(s,t)$ and p_{j+1} are co-visible. Then it follows that $\langle s, \mathrm{fd}(s,t), \mathrm{fd}^2(s,t), p_{j+1}, \ldots, p_k = t \rangle$ is a valid path of length $k - j + 2 \leqslant k$, contradicting the assumption that $\mathrm{fd}(s,t)$ is not on a shortest path. If $\mathrm{nd}(\mathrm{fd}(s,t),t)_x < p_{j,x}$, it follows with the same reasoning that the vertices $\mathrm{nd}(\mathrm{fd}(s,t),t)$ and

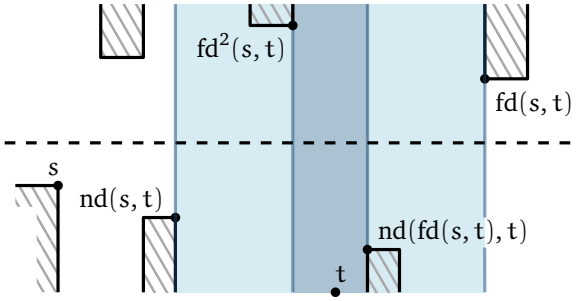


Figure 4.4: $\mathrm{fd}^2(s,t)$ lies between $\mathrm{nd}(s,t)$ and $\mathrm{fd}(s,t)$ and is closer to t than s. The darker region is $I(\mathrm{fd}(s,t),t)$ and a subset of $I(s,t)$, the brighter region.

p_{j+1} are co-visible then $\mathrm{fd}(s,t)$ is on the path $\langle s, \mathrm{fd}(s,t), \mathrm{nd}(\mathrm{fd}(s,t),t), p_{j+1}, \ldots, p_k = t\rangle$ which is a valid path of length $k-j+2 \leqslant k$. This is again a contradiction. Now, since $I(\mathrm{fd}(s,t),t) = \left[\mathrm{fd}^2(s,t), \mathrm{nd}(\mathrm{fd}(s,t),t)\right] \subseteq I(\mathrm{fd}^2(s,t))$, we get $p_2 \in I(\mathrm{fd}^2(s,t))$. Since p_2 sees $\mathrm{nd}(s,t)$ which is to the left of $\mathrm{fd}^2(s,t)$ and since p_2 is in $I(\mathrm{fd}(s,t),t)$, and thus to the right of $\mathrm{fd}^2(s,t)$, it follows that $\mathrm{fd}^2(s,t) \in I(p_2)$. □

4.2.3 The target can be made close in one step.

Let s,t be two vertices so that $t \notin I(s)$ but there is a vertex $v \in N(s)$ with $t \in I(v)$. For clarity of presentation, we will always assume that s is below the base line. The crux of this case is this: there might be many vertices visible from s that have t in their interval. However, we can find a best vertex as follows: once t is in the interval of a vertex, the goal is to shrink the interval (i.e., reduce it to include fewer vertices) as fast as possible. Therefore, we must find a vertex $v \in N(s)$ whose left or right interval boundary is closest to t among all vertices in $N(s)$. This leads to the following inductive definition of two sequences $a^i(s)$ and $b^i(s)$ of vertices in $N(s)$; see Figure 4.5. We omit (s) if the context is clear and instead write a^i and b^i, respectively. For $i = 0$, we let $a^0 = b^0 = s$. Otherwise, let $i > 0$. We let $A^i(s)$, and write A^i instead, to be the set of all vertices that can see further to the left than a^{i-1} but are still visible from s, i.e.,

$$A^i = \left\{v \in N(s) \mid \ell(v)_x < \ell(a^{i-1})_x\right\}.$$

The vertex a^i is the leftmost vertex in A^i, that can see further to the right than a^{i-1}. The formal definition of a^i also captures the case that A^i is empty and breaks unambiguity:

$$a^i = \begin{cases} a^{i-1} & \text{if } A^i = \emptyset \\ \operatorname{argmin}\left\{(v_x, |v_y|) \mid v \in A^i\right\} & \text{otherwise.} \end{cases}$$

The definition of b^i is quite similar but focusing the attention to the right instead of to the left. It is

$$B^i = \left\{v \in N(s) \mid r(b^{i-1})_x < r(v)_x\right\}$$

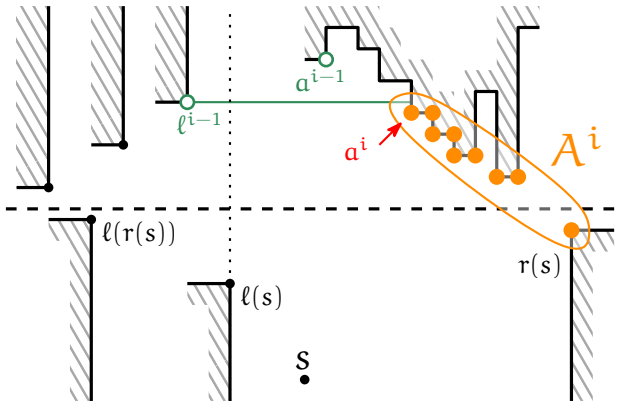


Figure 4.5: All the vertices in A^i can see further to the right than a^{i-1}. The leftmost vertex in A^i is a^i.

and

$$b^i = \begin{cases} b^{i-1} & \text{if } B^i = \emptyset \\ \operatorname{argmax}\{(v_x, -|v_y|) \mid v \in B^i\} & \text{otherwise.} \end{cases}$$

Finally, let a^* be the vertex with $a^* = a^i = a^{i-1}$, for an $i > 0$, and b^* the vertex with $b^* = b^i = b^{i-1}$, for an $i > 0$, respectively. Let us try to understand this definition. For $i \geqslant 0$, we write ℓ^i for $\ell(a^i)$; and we write ℓ^* for $\ell(a^*)$. Then, we have $a^0 = s$ and $\ell^0 = \ell(s)$. Now, if $\ell(s)$ is not a vertex, then $a^* = s$, because there is no vertex whose left point is strictly to the left of the left boundary of P. On the other hand, if ℓ^0 is a vertex in P, we have $a^1 = \ell^0 = \ell(s)$, and $[\ell^1, a^1]$ is an interval between points on the lower side of P. Then comes a (possibly empty) sequence of intervals $[\ell^2, a^2], [\ell^3, a^3], \ldots, [\ell^k, a^k]$ between points on the upper side of P; possibly followed by the interval $[\ell(r(s)), r(s)]$. There are four possibilities for a^*: it could be s, $\ell(s)$, a vertex a^i on the upper side of P, or $r(s)$. If $a^* \neq s$, then the intervals $[\ell^1, a^1] \subset [\ell^2, a^2] \subset \cdots \subset [\ell^*, a^*]$ are strictly increasing: ℓ^i is strictly to the left of ℓ^{i-1} and a^i is strictly to the right of a^{i-1}; see Figure 4.6. Symmetric observations apply for the b^i; we write r^i for $r(b^i)$ and r^* for $r(b^*)$.

Lemma 4.7. *For $i \geqslant 1$, the vertices ℓ^{i-1}, a^i as well as r^{i-1}, b^i are co-visible.*

Proof. We focus on ℓ^{i-1} and a^i. We show that $\ell^{i-1} \in I(a^i)$ and $a^i \in I(\ell^{i-1})$; the lemma follows from Observation 4.1. The claim $\ell^{i-1} \in I(a^i)$ is due to the facts that $[\ell^i, a^i] \subseteq I(a^i)$ and $\ell^{i-1} \in [\ell^i, a^i]$ (this holds also for $i = 1$, as then $\ell^{i-1} = a^i$). Next, since $a^{i-1} \in I(\ell^{i-1})$, the vertex a^{i-1} is to the left of $r(\ell^{i-1})$; and since $\ell^{i-1} \in I(r(\ell^{i-1}))$, the point $\ell(r(\ell^{i-1}))$ is to the left of ℓ^{i-1}. Thus, if $r(\ell^{i-1})$ is visible from s, we have $a^i = r(\ell^{i-1})$, by the definition of a^i. On the other hand, if $r(\ell^{i-1})$ is not visible from s, the visibility must be blocked by $r(s)$, and then $a^i = r(s)$. In either case, we have $a^i \in [a^{i-1}, r(\ell^{i-1})] \subseteq I(\ell^{i-1})$, as desired. □

Finally, the next lemma tells us the following: if $t \in [\ell^*, r^*]$ we find a vertex $v \in N(s)$ with $t \in I(v)$.

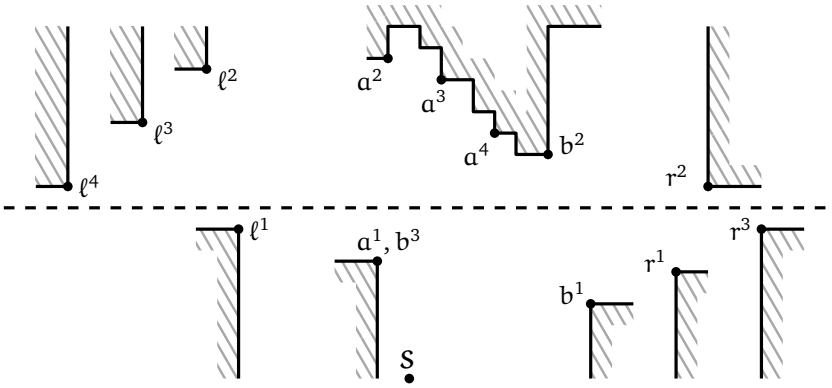


Figure 4.6: Illustration of the a^i and b^i. Observe that $\ell(s) = a^1 = b^3$ and $r(s) = b^1$.

Lemma 4.8. *If* $t \in [\ell^i, \ell^{i-1}]$*, for some* $i \geqslant 1$*, then* a^i *is on a shortest path from* s *to* t*. If* $t \in [r^{i-1}, r^i]$*, for some* $i \geqslant 1$*, then* b^i *is on a shortest path from* s *to* t*.*

Proof. We focus on the first statement; see Figure 4.7. Let $\pi : \langle s = p_0, \ldots, p_k = t \rangle$ be a shortest path from s to t, and let p_j be the last vertex on π outside of $[\ell^i, \ell^{i-1}]$. If $j = 0$, then p_{j+1} must be $\ell^0 = \ell(s)$, because this is the only vertex ℓ^{i-1} visible from s. Then, $i = 1$ and $a^i = \ell(s)$ is on π. From now on, we assume that $j \geqslant 1$.

First, suppose that p_{j+1} and a^i are co-visible. Then $\langle s, a^i, p_{j+1}, \ldots, p_k \rangle$ is a path from s to t that uses a^i and has length $k - j + 1 \leqslant k$. Second, suppose that p_{j+1} and a^i are not co-visible. Then, the contrapositive of Observation 4.2 applied to the four points ℓ^i, p_{j+1}, a^i, and p_j shows that p_j is strictly to the left of a^i. There are two subcases, depending on whether p_j is strictly to the left of ℓ^i or strictly to the right of ℓ^{i-1}.

If p_j is strictly to the left of ℓ^i, then $j \geqslant 2$, since ℓ^i is to the left of $\ell^0 = \ell(s)$ and we need at least two hops to reach a point strictly to the left of $\ell(s)$ from s. We apply Observation 4.2 on the four points p_j, ℓ^i, p_{j+1}, and a^i, and get that ℓ^i and p_{j+1} are co-visible. Hence, $\langle s, a^i, \ell^i, p_{j+1}, \ldots, p_k \rangle$ is a path that uses a^i and has length $k - j + 2 \leqslant k$.

Finally, assume that p_j is strictly to the right of ℓ^{i-1}. By Lemma 4.7, a^i can see ℓ^{i-1}. Thus, $p_{j+1} \neq \ell^{i-1}$ and there is no vertex strictly between ℓ^{i-1} and a^i on the same side as ℓ^{i-1} that can see a vertex strictly to the left of ℓ^{i-1}. Thus, p_j and a^{i-1} are on different sides of the base line. Let b be the rightmost vertex that (i) lies on the same side of P as p_j; (ii) is strictly between ℓ^{i-1} and a^i; (iii) is closest to the base line. The vertex b exists (since p^j is a candidate), is not visible from s (because b is strictly left of a^i and can see strictly left of ℓ^{i-1}); and thus strictly left of $\ell(s)$. The vertex p_j cannot be strictly to the right of b, as otherwise b would obstruct visiblity between p_j and p_{j+1}. We conclude that $j \geqslant 2$, since we need at least two hops to reach a point strictly to the left of $\ell(s)$ from s. If $p_j \in \{b, \mathrm{cv}(b)\}$, a^i can see p_j and thus, $\langle s, a^i, p_j, p_{j+1}, \ldots, p_k \rangle$ is a path of length $k - j + 2 \leqslant k$ using a^i. If $p_j \notin \{b, \mathrm{cv}(b)\}$, then b is strictly closer to the base line than p_j. Then, we have $j \geqslant 3$, because we need two hops to cross the vertical line through $\ell(s)$ and one more hop to cross the horizontal line through b. We apply Observation 4.2 on the four points p_{j+1}, ℓ^{i-1}, p_j,

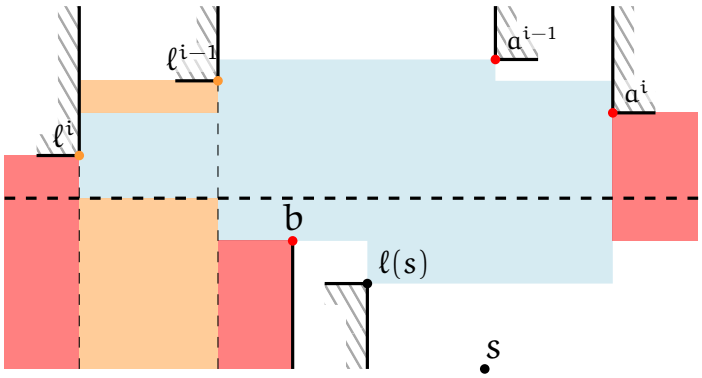


Figure 4.7: The vertex a^i is on a shortest path. The vertex p_j can lie in the red regions, the vertex p_{j+1} can lie in the orange region, and the blue region cannot contain any point outside of P.

and b to conclude that ℓ^{i-1} can see p_j. Hence, $\langle s, a^i, \ell^{i-1}, p_j, p_{j+1}, \ldots, p_k\rangle$ is a path of length $k - j + 3 \leqslant k$ using a^i. □

4.2.4 The target is far away

Finally, we consider the case that there is no vertex $v \in N(s)$ with $t \in I(v)$, i.e., $t \notin [\ell^*, r^*]$. The intuition now is as follows: to widen the interval, we should go to a vertex that is visible from s, but closest to the base line. In simple histograms, there was only one such vertex, but in double histograms there might be a second one on the other side. These two vertices are the *dominators of* s. They might have their own dominators, and so on. This leads to the following inductive definition.

For $k \geqslant 0$, we define the k-th *bottom dominator* $bd^k(s)$, the k-th *top dominator* $td^k(s)$, and the k-th *interval* $I^k(s)$ of s. For any set $Q \subset V(P)$, we write Q^- (resp. Q^+) for all points in Q below (resp. above) the base line. We set $bd^0(s) = td^0(s) = s$ and $I^0(s) = \{s\}$. For $k > 0$, we set $I^k(s) = I(bd^{k-1}(s)) \cup I(td^{k-1}(s))$. If $I^k(s)^-$ is nonempty, we let $bd^k(s)$ be the leftmost vertex inside $I^k(s)^-$ that minimizes the distance to the base line. If $I^k(s)^+$ is nonempty, we let $td^k(s)$ be the leftmost vertex inside $I^k(s)^+$ that minimizes the distance to the base line; see Figure 4.8. If one of the two sets is empty, the other one has to be nonempty, since $s \in I^k(s)$. In this case, we let $td^k(s) = bd^k(s)$. We write $bd(s)$ for $bd^1(s)$ and $td(s)$ for $td^1(s)$.

Observe, that $I^1(s) = I(s)$ and $I^2(s) = [\ell^*, r^*]$. If $I(bd^{k-1}(s)) = V(P)$, we have $bd^k(s) = bd^{k-1}(s)$. The same holds for the top dominator. We provide a few technical properties concerning the k-th interval as well as the k-th dominators.

Lemma 4.9. *For any* $s \in V$ *and* $k \geqslant 0$*, we have* $I^k(s) \subseteq I(bd^k(s)) \cap I(td^k(s))$ *and* $bd^k(s)$, $td^k(s)$ *are co-visible.*

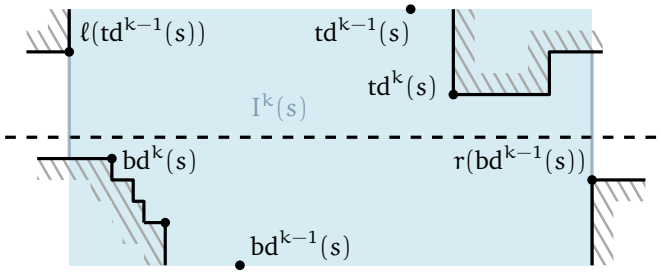


Figure 4.8: The $k-1$-th and k-th dominators and the k-th interval.

Proof. We have $I^k(s) \subseteq I(\mathrm{bd}^k(s))$, since by definition, the interval $I^k(s)$ contains no vertex that is on the same side as $\mathrm{bd}^k(s)$ and strictly closer to the base line, so no vertex can obstruct horizontal visibility of $\mathrm{bd}^k(s)$ in $I^k(s)$. Analogously, $I^k(s) \subseteq I(\mathrm{td}^k(s))$, as desired.

By definition and the first part, $\mathrm{td}^k(s) \in I^k(s) \subseteq I(\mathrm{bd}^k(s))$ and $\mathrm{bd}^k(s) \in I^k(s) \subseteq I(\mathrm{td}^k(s))$. The claim now follows from Observation 4.1. □

The following lemma seems rather specific, but will be needed later to deal with short paths.

Lemma 4.10. *For any $s \in V$, we have $I^3(s) = I^2\big(\mathrm{bd}(s)\big) \cup I^2\big(\mathrm{td}(s)\big)$.*

Proof. We begin by showing that

$$I\big(\mathrm{bd}^2(s)\big) = I\big(\mathrm{bd}(\mathrm{td}(s))\big) \cup I\big(\mathrm{bd}(\mathrm{bd}(s))\big). \tag{4.1}$$

If $\mathrm{bd}(s)$ is above the base line, then $\mathrm{bd}(s) = \mathrm{td}(s)$ and $I^2(s) = I(\mathrm{bd}(s)) \cup I(\mathrm{td}(s)) = I(\mathrm{td}(s))$. The definition of $\mathrm{bd}^2(s)$ then gives $\mathrm{bd}^2(s) = \mathrm{bd}(\mathrm{td}(s))$, and Equation (4.1) follows.

If $\mathrm{bd}(s)$ is below the base line, the vertex $b_1 = \mathrm{bd}(\mathrm{bd}(s))$ is below the base line. Let $b_2 = \mathrm{bd}(\mathrm{td}(s))$. By Lemma 4.9, $\mathrm{bd}(s)$ and $\mathrm{td}(s)$ are co-visible, so $\mathrm{bd}(s) \in I(\mathrm{td}(s))^-$. Therefore, b_2 is below the base line. Since $I(b_1)$ and $I(b_2)$ are not disjoint (both contain s) and since b_1 and b_2 are on the same side of the base line, Observation 4.3 gives $I(b_1) \subseteq I(b_2)$ or $I(b_2) \subseteq I(b_1)$. Because $\mathrm{bd}^2(s)$ is the highest vertex in $\big(I(\mathrm{bd}(s)) \cup I(\mathrm{td}(s))\big)^-$, we get that $\mathrm{bd}^2(s)$ is b_1 or b_2, and Equation (4.1) follows also in this case. Symmetrically, we have

$$I(\mathrm{td}^2(s)) = I(\mathrm{td}(\mathrm{td}(s))) \cup I(\mathrm{td}(\mathrm{bd}(s))). \tag{4.2}$$

We use the definitions and Equations (4.1) and (4.2) to get

$$\begin{aligned} I^3(s) &= I\big(\mathrm{bd}^2(s)\big) \cup I\big(\mathrm{td}^2(s)\big) \\ &= I\big(\mathrm{bd}(\mathrm{td}(s))\big) \cup I\big(\mathrm{bd}(\mathrm{bd}(s))\big) \cup I\big(\mathrm{td}(\mathrm{td}(s))\big) \cup I\big(\mathrm{td}(\mathrm{bd}(s))\big) \\ &= I^2\big(\mathrm{bd}(s)\big) \cup I^2\big(\mathrm{td}(s)\big), \end{aligned}$$

as desired. □

It is possible to prove the following property: let ℓ be the leftmost and r be the rightmost vertex with hop distance exactly k from s, then, $I^k(s) = [\ell, r]$. We do not need this property, so we leave it as an exercise for the reader to find a proof. Instead, we prove the following weaker statement. For this, recall that due to its definition, $\mathrm{bd}^k(s)$ might not be on the lower side of the histogram (and $\mathrm{td}^k(s)$ might not be on the upper side).

Lemma 4.11. *Let* $k \geqslant 0$ *and let* $s, t \in V$ *with* $d(s,t) \leqslant k$. *Then,* $t \in I^k(s)$.

Proof. We show that for any $j \geqslant 0$ and any vertex $v \in I^j(s)$, we have $N(v) \subseteq I^{j+1}(s)$. The lemma then follows by induction. If $v \in I^j(s)^-$, then $\mathrm{bd}^j(s)$ is on the lower side, and by definition, $I(v) \subseteq I(\mathrm{bd}^j(s))$. If $v \in I^j(s)^+$, by a similar argument $I(v) \subseteq I(\mathrm{td}^j(s))$. Thus, $N(v) \subseteq I(v) \subseteq I(\mathrm{bd}^j(s)) \cup I(\mathrm{td}^j(s)) = I^{j+1}(s)$, as desired. □

Let $k \geqslant 0$ and $s \in V$. For $i = 1, \ldots, k$, by Lemma 4.9, $\mathrm{bd}^{i-1}(s), \mathrm{td}^{i-1}(s) \in I(\mathrm{bd}^i(s)) \cap I(\mathrm{td}^i(s))$. Moreover, by definition, $\mathrm{bd}^i(s), \mathrm{td}^i(s) \in I(\mathrm{bd}^{i-1}(s)) \cup I(\mathrm{td}^{i-1}(s))$. As we show in the full version, now both $\mathrm{bd}^i(s)$ and $\mathrm{td}^i(s)$ can see at least one of $\mathrm{bd}^{i-1}(s)$ or $\mathrm{td}^{i-1}(s)$. Therefore, there is a path $\pi_b(s,k) : \langle s = p_0, \ldots, p_k = \mathrm{bd}^k(s)\rangle$ from s to $\mathrm{bd}^k(s)$ and a path $\pi_t(s,k) : \langle s = q_0, \ldots, q_k = \mathrm{td}^k(s)\rangle$ from s to $\mathrm{td}^k(s)$ with $p_i, q_i \in \{\mathrm{bd}^i(s), \mathrm{td}^i(s)\}$, for $i = 0, \ldots, k$. We call $\pi_b(s,k)$ and $\pi_t(s,k)$ the *canonical path* from s to $\mathrm{bd}^k(s)$ and from s to $\mathrm{td}^k(s)$, respectively. The following two lemmas show that for every $t \notin I^{k+1}(s)$ one of the canonical paths is the prefix of a shortest path from s to t. To show Lemma 4.13 we need Lemmas 4.11 and 4.12.

Lemma 4.12. *Let* $k \geqslant 1$ *and* $s \in V$. *If* $I(\mathrm{bd}^{k-1}(s)) \neq V$, *we have that* $d(s, \mathrm{bd}^k(s)) = k$. *If* $I(\mathrm{td}^{k-1}(s)) \neq V$ *we have* $d(s, \mathrm{td}^k(s)) = k$.

Proof. On the one hand, $d(s, \mathrm{bd}^k(s)) \leqslant |\pi_b(s,k)| = k$ and $d(s, \mathrm{td}^k(s)) \leqslant |\pi_t(s,k)| = k$. On the other hand, we show that $\mathrm{bd}^k(s) \notin I^{k-1}(s)$ and $\mathrm{td}^k(s) \notin I^{k-1}(s)$. The claim then follows by the contrapositive of Lemma 4.11.
Case 1: First, assume that $\mathrm{bd}^{k-1}(s) \in I^{k-1}(s)^-$. Since $I(\mathrm{bd}^{k-1}(s)) \neq P$, at least one of its bounding points is a vertex v contained in $I^k(s)$. Then, v is strictly closer to the base line than $\mathrm{bd}^{k-1}(s)$, and since v is a candidate for $\mathrm{bd}^k(s)$, the same applies to $\mathrm{bd}^k(s)$. It follows that $\mathrm{bd}^k(s) \notin I^{k-1}(s)$. Similarly, we get that if $\mathrm{td}^{k-1}(s) \in I^{k-1}(s)^+$, the vertex $\mathrm{td}^k(s)$ is not in $I^{k-1}(s)$.
Case 2: Second, assume that $\mathrm{bd}^{k-1}(s) \in I^{k-1}(s)^+$. Then, we have that $\mathrm{bd}^k(s) \notin I^{k-1}(s)^-$, since this set is empty. Thus, suppose for a contradiction that $\mathrm{bd}^k(s) \in I^{k-1}(s)^+$. This can only be the case if $\mathrm{bd}^{k-1}(s) = \mathrm{td}^{k-1}(s)$ and $\mathrm{bd}^k(s) = \mathrm{td}^k(s)$. However, in Case 1 we showed that $\mathrm{td}^k(s) \notin I^{k-1}(s)^+$ if $\mathrm{td}^{k-1}(s) \in I^{k-1}(s)^+$. Hence, $\mathrm{bd}^k(s) \notin I^{k-1}(s)^+$, as desired. □

Lemma 4.13. *Let* s *and* t *be vertices and* $k \geqslant 1$ *an integer such that* $t \notin I^{k+1}(s)$. *Then* $\mathrm{bd}^k(s)$ *or* $\mathrm{td}^k(s)$ *is on a shortest path from* s *to* t.

Proof. First, observe that $I^{k+1}(s) = I(\mathrm{bd}^k(s)) \cup I(\mathrm{td}^k(s)) \neq V$, as $t \notin I^{k+1}(s)$. Next, let $\pi : \langle s = p_0, \ldots, p_m = t\rangle$ be a shortest path from s to t, and p_j the last vertex in $I^{k+1}(s)$. Without loss of generality, p_{j+1} is strictly to the right of s. By Lemma 4.11, we get that p_j is not in $I^k(s)$ and thus, again by Lemma 4.11, we have $j \geqslant d(s, p_j) \geqslant k+1$.

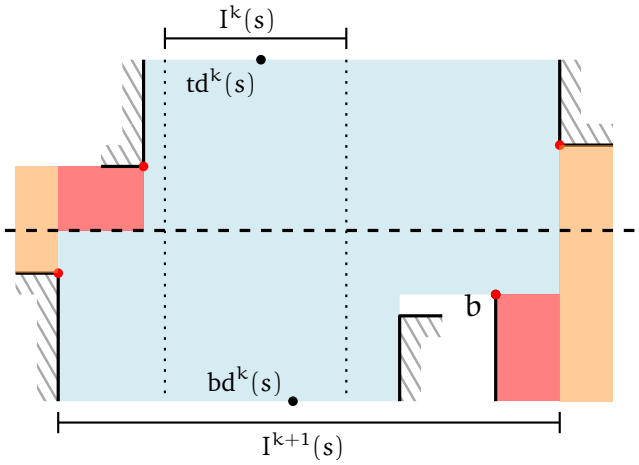


Figure 4.9: $bd^k(s)$ or $td^k(s)$ is on a shortest path. The vertex p_j lies in one of the red regions, p_{j+1} lies in one of the orange regions, and the blue region cannot contain any point outside of P.

First, suppose that p_j and $bd^k(s)$ (resp. $td^k(s)$) are co-visible. Then, $\pi_b(s,k) \circ \langle p_j, \dots, t\rangle$ (resp. $\pi_t(s,k) \circ \langle p_j, \dots, t\rangle$) is a valid path of length $k+1+(m-j) \leqslant m$. Here, $\circ$ concatenates two paths. Second, suppose p_j be visible from neither $bd^k(s)$ nor $td^k(s)$. First, we claim that p_j is strictly to the right of $bd^k(s)$ and $td^k(s)$. Otherwise, since p_{j+1} is strictly to the right of both dominators, we would get $bd^k(s), td^k(s) \in I(p_j)$. Moreover, $p_j \in I^{k+1}(s) = I(bd^k(s)) \cup I(td^k(s))$. Observation 4.1 now would imply that p_j can see $bd^k(s)$ or $td^k(s)$ — a contradiction. The claim follows. Next, we claim $bd^k(s) \neq td^k(s)$. If not, $p_j \in I(bd^k(s)) = I^{k+1}(s)$, and since p_j can see a point outside of $I^{k+1}(s)$, we would get $p_j \in \{\ell(bd^k(s)), r(bd^k(s))\}$, which again contradicts our assumption that p_j cannot see $bd^k(s)$. The claim follows. There are two cases, depending on which dominator sees further to the right.

Case 1: $r(bd^k(s))_x < r(td^k(s))_x$; see Figure 4.9. Let b be the leftmost vertex in the interval $[r(bd^k(s)), r(td^k(s))]^-$ closest to the base line. Observe that b is strictly to the right of $I^k(s)$, because $r(bd^k(s))$ is strictly to the right of $I^k(s)$. Since p_j is not visible from $td^k(s)$, it has to be strictly to the right of and strictly below b. Next, we claim that no vertex $v \in I^k(s)$ can see p_j. If one could, by Observation 4.1, we would have $v \in I(p_j)$. But since p_j is strictly to the right of and strictly below b, then v would be to the right of b, which is impossible. This shows the claim. Thus, by Lemma 4.11, $j \geqslant d(s,p_j) \geqslant k+2$. We apply Observation 4.2 to $td^k(s)$, p_j, $r(td^k(s))$ and p_{j+1} and get that $r(td^k(s))$ can see p_j. Therefore, $\pi_t(s,k) \circ \langle r(td^k(s)), p_j, \dots, t\rangle$ is a valid path of length $k+1+(1+m-j) \leqslant m$.

Case 2: $r(td^k(s))_x < r(bd^k(s))_x$. Let b be the leftmost vertex in $[r(td^k(s)), r(bd^k(s))]^+$ closest to the base line. Observe that b is strictly to the right of $I^k(s)$, because $r(td^k(s))$ is strictly to the right of $I^k(s)$. Since p_j is not visible from $bd^k(s)$, it has to be strictly to the right of and strictly above b. Next, we claim that no vertex $v \in I^k(s)$ can see p_j. If one could, by Observation 4.1, we would have $v \in I(p_j)$. But since p_j is strictly to the right of and strictly

above b, then v would be to the right of b, which is impossible. This shows the claim. Thus, by Lemma 4.11, $j \geqslant d(s, p_j) \geqslant k+2$. We apply Observation 4.2 to $bd^k(s)$, p_j, $r(bd^k(s))$ and p_{j+1} and get that $r(bd^k(s))$ can see p_j. Therefore, $\pi_b(s,k) \circ \langle r(bd^k(s)), p_j, \ldots, t\rangle$ is a valid path of length $k+1+(1+m-j) \leqslant m$, as desired. □

4.3 The Routing Scheme

Let $\mathrm{Vis}(P)$ be the visibility graph of a double histogram. We assume that the graph is represented with adjacency lists with direct access to the coordinates of the vertices $P \subset \mathbb{R}^2$. We say the representation is *canonical* if the x-coordinates of the vertices are integers in $[n/2-1]$. We describe the routing scheme for canonical representations. Later, we briefly discuss how to drop this assumption.

The labels and routing tables. Since we assumed that no three points are on a horizontal line, we conclude that for every integer $k \in [n/2-1]$ there are exactly 2 vertices in P having x-coordinate k. Let v be a vertex. There is exactly one vertex u sharing the same vertical edge with v – it has the same x-coordinate. We define $b(v) = 1$ if v is closer to the base line and $b(v) = 0$ if u is closer to the base line. We set $\mathrm{lab}(v) = (v_x, b(v))$ and observe that $|\mathrm{lab}(v)| = \lceil \log n \rceil$. Moreover, $\mathrm{lab}(v)$ identifies v in the network. Next, let $w \in N(v)$ be a neighbor of v and p_w the port number pointing from v to w. We store $(\mathrm{lab}(w), \ell(w)_x, r(w)_x, p_w)$ in the table of v. Moreover, we store the bounding x-coordinates of $I^2(\mathrm{bd}(v))$ as well as the bounding x-coordinates of $I^2(\mathrm{td}(v))$ in the table of v. Last but not least, let $\pi_b(v,2) = \langle v, q, \mathrm{bd}^2(v)\rangle$. Remark, that $q \in \{\mathrm{bd}(v), \mathrm{td}(v)\}$. We store p_q as well as the port p from q to $\mathrm{bd}^2(v)$ in $\mathrm{tab}(v)$. Finally, we store $\mathrm{lab}(v)$ in $\mathrm{tab}(v)$. In summary, we can conclude that the table size of v is $\lceil \log n \rceil (4\deg(v)+7) - 2(\deg(v)+2) \in \Theta(\log n \deg(v))$.

The routing function. The routing function for double histograms is presented in Algorithm 4.1. We are given a current vertex s together with its routing table $\mathrm{tab}(s)$, the label of a target vertex t, and a header h. If $t \in N(s)$, then $\mathrm{lab}(t)$ is in the routing table of s, and we send the data packet directly to t. If the header is non-empty, it will contain a port number. We clear the header and use this port number for the next hop. The remaining discussion assumes that the header is empty and that $t \notin N(s)$. The routing function now distinguishes four cases depending on whether $t \in I(s)$, $t \in I^2(s)$ or $t \in I^3(s)$. We can check the first and the second condition locally, using the routing table of s as well as the label of t (note that from the routing table of s, we can deduce $\ell(s)$, $r(s)$, $a^*(s)$ and $b^*(s)$, and their interval boundaries). To check the third condition locally, we use Lemma 4.10 which shows that $I^3(s) = I^2(\mathrm{bd}(s)) \cup I^2(\mathrm{td}(s))$. Since we stored the bounding x-coordinates of these two intervals in the routing table of s, we can check $t \in I^3(s)$ easily.

Case 1 ($t \in I(s) \setminus N(s)$): if $\mathrm{fd}(s,t)$ is a vertex, we can determine it by using the routing table and the label of t. The packet is sent to $\mathrm{fd}(s,t)$. If $\mathrm{fd}(s,t)$ is not a vertex, we determine $\mathrm{nd}(s,t)$ and send the packet there. The header remains empty.

Case 2 ($t \in I^2(s) \setminus I(s)$): there is an $i \geqslant 1$ with $t \in [\ell^i, \ell^{i-1}]$ or $t \in [r^{i-1}, r^i]$. We find i

Algorithm 4.1 The routing function for double histograms with stretch 2.

```
1:  procedure RoutingFunction(tab(s) ∈ Σ*, lab(t) ∈ Σ*, header h ∈ Σ*)
2:      Outputs: (port p ∈ Σ*, header h′ ∈ Σ*)
3:      if h ≠ ε then return (h, ε)
4:      if t = s then return (ε, ε)
5:      if t ∈ N(s) then return (p_t, ε)
6:      if t ∈ I(s) and fd(s,t) ∈ N(s) then return (p_fd(s,t), ε)
7:      if t ∈ I(s) and fd(s,t) ∉ N(s) then return (p_nd(s,t), ε)
8:      if t ∈ I²(s) and t_x < ℓ(s)_x then
9:          compute i with t ∈ [ℓ^i, ℓ^(i−1)] and return (p_a^i, ε)
10:     if t ∈ I²(s) and t_x > r(s)_x then
11:         compute i with t ∈ [r^(i−1), r^i] and return (p_b^i, ε)
12:     if t ∈ I²(bd(s)) then return (p_bd(s), ε)
13:     if t ∈ I²(td(s)) then return (p_td(s), ε)
14:     else
15:         extract the entry (p_q, p) from tab(s)
16:         return (p_q, p)
```

using the routing table and lab(t). The packet is sent to a^i or b^i. The header remains empty.
Case 3 ($t \in I^3(s) \setminus I^2(s)$)**:** if $t \in I^2(bd(s))$, we send the packet to bd(s). Otherwise, $t \in I^2(td(s))$, and we send the packet to td(s). In both cases, the header remains empty.
Case 4 ($t \notin I^3(s)$)**:** the routing table has the entry (p_q, p). We store p in the header and use p_q to send the packet to $q \in \{bd(s), td(s)\}$.

The label and table size have been analyzed so far. The header size is $H(n) = \lceil \log n \rceil$, obviously. It remains to analyze the stretch as well as the preprocessing time.

The stretch. First, we present a bound for the stretch factor. The following lemma proves the correctness and bounds the stretch factor by 2.

Lemma 4.14. *Let* $s, t \in V$. *After at most two steps of the routing scheme from* s *with target label* lab(t)*, we reach a vertex* v *with* $d(v, t) \leqslant d(s, t) - 1$.

Proof. First, if $t \in N(s)$, then we take one hop and decrease the distance to 0. Second, suppose that $t \in I(s) \setminus N(s)$. If fd(s, t) is not a vertex, the next vertex is nd(s, t), which is on a shortest path from s to t due to Lemma 4.5. Otherwise, fd(s, t) is the next vertex. If fd(s, t) is on a shortest path from s to t, we are done. Otherwise, t is not visible from fd(s, t), so $fd^2(s, t)$ has to be the second vertex on the routed path. By Lemma 4.6, we have $d(fd^2(s, t), t) = d(s, t) - 1$. Third, if $t \in I^2(s) \setminus I(s)$, there is an $i \geqslant 1$ such that the next vertex is either a^i or b^i. By Lemma 4.8, this vertex is on a shortest path. Fourth, assume $t \in I^3(s) \setminus I^2(s)$. Let v_1 and v_2 be the next two vertices on the routing path. We use Lemma 4.9 and Lemma 4.13 to conclude $d(v_1, t) \leqslant d(s, t)$, as v_1 is either td(s) or bd(s).

Due to the construction of the routing function, we have $t \in I^2(v_1) \setminus I(v_1)$. Thus, there is an $i \geqslant 1$, such that $v_2 = a^i(v_1)$ or $v_2 = b^i(v_1)$. By Lemma 4.8, the vertex v_2 is on a shortest path from v_1 to t and we can conclude $d(v_2, t) = d(v_1, t) - 1 \leqslant d(s, t) - 1$. Last, assume $t \notin I^3(s)$. Then, the packet is routed to a vertex $p \in \{bd(s), td(s)\}$, whichever is on a shortest path to $bd^2(s)$, and then $bd^2(s)$. Lemmas 4.9 and 4.13 give $d(bd^2(s), t) \leqslant d(s, t) - 1$, as claimed. □

The preprocessing time. Next, we want to take a closer look at the preprocessing and analyze its time complexity. Let m be the number of edges in the $\mathrm{Vis}(P)$. Again, since the output consists of $\Theta(m \log n)$ bits, the overall preprocessing time has to be at least $\Omega(m)$. First, we compute an explicit representation of the boundary of the double histogram in $O(m)$ time by using the adjacency lists of $\mathrm{Vis}(P)$. We can then translate the double histogram (and the visibility graph) in $O(n)$ time such that the base line is on the x-axis. Next, we want to present an algorithm that computes $\ell(v)$, $r(v)$, $td(v)$ and $bd(v)$, for all $v \in V$, in $O(n)$ time. The routing tables and labels are then immediate and can be computed in $O(m)$ time.

We introduce two dummy vertices: the intersection of the base line with the boundary of P. The left dummy vertex is denoted by v_l, while the right dummy vertex is denoted by v_r. We split the double histogram into two parts: the lower part P_L and the upper part P_U. Both parts are simple histograms with base vertices v_l and v_r. We focus our description on P_L. The preprocessing for the vertices of P_U is symmetric. Since P_L is a simple histogram, we can use the preprocessing algorithm to compute $\ell(v)$ as well as $r(v)$ for every vertex v of P_L. Moreover, we can immediately compute $bd(v)$, since

$$bd(v) = \begin{cases} v & \text{if } \ell(v) = v_l \wedge r(v) = v_r \\ \ell(v) & \text{if } \ell(v) \neq v_l \wedge r(v) = v_r \\ r(v) & \text{if } \ell(v) = v_l \wedge r(v) \neq v_r \\ \operatorname{argmax}\{\ell(v)_y, r(v)_y\} & \text{otherwise.} \end{cases}$$

The computation of $\ell(v)$, $r(v)$ and $bd(v)$ for every vertex v of P_L can be done in linear time. Next, we have to compute $td(v)$. This computation is a bit more involved. First, we observe that in the general case, i.e., $td(v)$ is located in P_U, the vertex $td(v)$ has to be reflex. Thus, we are looking for a reflex vertex in P_U with minimum y-coordinate that is contained in the interval $[\ell(v)_x, r(v)_x]$. This corresponds to the *RMQ-Problem* (range minimum query problem). It is defined as follows:

> **RMQ-Problem:**
>
> **Given:** Array A with n elements of a well ordered universe
>
> **Want:** Preprocess A such that the following *query* can be answered: for given $0 \leqslant k_1 \leqslant k_2 \leqslant n - 1$ find the smallest element x of A whose index i satisfies $k_1 \leqslant i \leqslant k_2$.

It is known that A can be preprocessed in linear time and queries can then be answered in constant time [BFC00, BV93, FH06]. Hence, we can do the following to compute $td(v)$ for all

vertices v of P_L: we introduce an array A with $n/2$ elements. By assumption we know that all x-coordinates of the vertices are integers in $[n/2-1]$. The array A is defined as follows:

$$A[i] = \begin{cases} v_y & \text{if there is a reflex vertex } (i, v_y) \text{ in } P_U \\ \infty & \text{otherwise.} \end{cases}$$

We can compute A in linear time using a simple scan of P_U from left to right. Next, we preprocess A according to the RMQ-Problem, which can be done in linear time, as well. Finally, to compute $td(v)$ for a given vertex v of P_L, we query the array A with the parameters $k_1 = \ell(v)_x$ and $k_2 = r(v)_x$. This means, we look for the vertex with smallest y-coordinate within $I(v)^+$, which is exactly $td(v)$. Let (v_x, v_y) be the query answer. If $v_y = \infty$, we know that there is no vertex in $I(v)^+$. Hence, we set $td(v) = bd(v)$. Otherwise, $td(v) = (v_x, v_y)$. A query for one vertex v can be done in constant time. Thus, to compute all $td(v)$ we need linear time. Once, this process is over, the labels and routing tables can now be computed in $O(\deg(v))$ time per vertex.

Non-canonical representations. Finally, we briefly discuss how to handle a visibility graph with vertices whose x-coordinates are arbitrary real numbers. First, we compute in $O(m)$ time an explicit representation of the boundary of the double histogram. Moreover, let $L = (e_0, e_1, \ldots, e_{n/2-1})$ be the list of all vertical edges of the boundary sorted by increasing x-coordinate. Since we have an explicit representation of the boundary we can compute L in $O(n)$ time by walking simultaneously on the upper and lower part. Now, we assign the x-coordinate i to both vertices of the edge $e_i \in L$, for each $i \in [n/2-1]$. The y-coordinate remains unchanged. We observe that the order of the vertices relative to the x-axis (resp. y-axis) has not been changed. Finally, it is well-known that the r-visibility graph does not change if the relative orders of the vertices do not change. Hence, we can apply the routing scheme for the canonical representation. This yields our theorem for double histograms.

Theorem 4.15. *Let P be a double histogram with n vertices and m edges in $\mathrm{Vis}(P)$. There is a routing scheme for $\mathrm{Vis}(P)$ with header size $\lceil \log n \rceil$, label size $\lceil \log n \rceil$, and each vertex v obtains a routing table of size $\Theta(\log n \deg(v))$. The labels and tables can be computed in time $O(m)$ and the stretch factor is 2.*

Remark. In the slightly different routing model where we distinguish the link and the routing table we again obtain link tables of size $\Theta(\log n \deg(v))$ but routing tables of size of $O(\log n)$; see [CCK+20].

CHAPTER 5

Span, Cover, Decompose

In this chapter we want to present results on how to span, cover and decompose planar graphs or unit disk graphs. These techniques are used to design the routing scheme for unit disk graphs. Every graph is considered to be undirected and weighted.

5.1 Planar Spanners

The first ingredient for our routing scheme is the concept of planar spanners.

Definition 5.1 (c-spanner). *Let* $G = (V, E_G, w_G)$ *be a simple, weighted graph and* $c \geqslant 1$ *a real number. A graph* $H = (V, E_H, w_H)$ *is called* c-spanner for G *if and only if the following two properties hold:*

(i) H is a subgraph of G, *i.e.,* $E_H \subseteq E_G$ *and* $w_H(e) = w_G(e)$ *for each* $e \in E_H$.

(ii) H preserves distances approximately, *i.e,* $\forall s, t \in V : d_G(s,t) \leqslant d_H(s,t) \leqslant c \cdot d_G(s,t)$.

The left inequality of Property *(ii)* is not necessary for the definition, since it immediately follows from the fact that H is a subgraph of G.

We are interested in a specific class of spanners: *planar* c-spanners of unit disk graphs. Let V be a set of n points in the plane. The graph $\mathrm{UDel}(V)$ is obtained from the Delaunay triangulation of V by removing all edges of weight larger than one; see Figure 5.1. The following lemma shows, that $\mathrm{UDel}(V)$ is a planar spanner of the unit disk graph $\mathrm{DG}(V)$ whose construction is easy and efficient. The proof for the spanning ratio is due to Li, Calinescu, and Wan.

Lemma 5.1 (see [LCW02]). *Let* V *be a set of* n *points in the euclidean plane. Then there exists a planar* 2.42*-spanner for the unit disk graph* $\mathrm{DG}(V)$ *that can be computed in time* $O(n \log n)$.

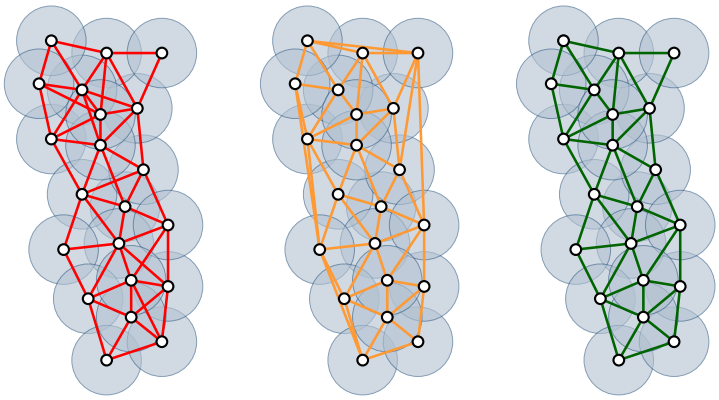


Figure 5.1: Left: $DG(V)$. Middle: Delaunay triangulation of V. Right: Intersection of both: $UDel(V)$.

5.2 Sparse Covers

The second technique can be applied to planar graphs and serves as a building block to cover a unit disk graph.

Definition 5.2 (r-cover)**.** *Let $H = (V, E)$ be a planar graph, and let $r \in \mathbb{N}$. An* r-cover *for H is a collection of m subgraphs $\{H_1, H_2, \ldots, H_m\}$ of H with the following two properties:*

(i) For each vertex $v \in V$, there is at least one subgraph H_i that contains all the vertices $w \in V$ with $d_H(v, w) \leqslant r$.

(ii) Every subgraph H_i is connected and has $\operatorname{diam}(H_i) \leqslant O(r)$.

A cover $\{H_i\}_i$ of H is said to be sparse *if and only if every vertex $v \in V$ is contained in $O(1)$ subgraphs H_i.*

The following lemma summarizes some key facts about sparse covers. The existence of a sparse r-cover was proven by Busch, LaFortune, and Tirthapura [BLT07]. They also gave an algorithm that computes a sparse r-cover. The running time was later analyzed by Kawarabayashi, Sommer, and Thorup [KST13].

Lemma 5.2. *For any planar graph $H = (V, E)$ with n vertices and for any $r \in \mathbb{N}$, we can compute a sparse r-cover $\{H_i\}_i$ for H in $O(n \log n)$ time. Moreover, each vertex $v \in V$ is contained in at most 18 subgraphs and has $\operatorname{diam}(H_i) \leqslant 48 \cdot r$, for each H_i.*[1] *For a given vertex v, we can identify in constant time*[2] *a subgraph H_i containing all vertices $w \in V$ with $d_H(v, w) \leqslant r$.*

[1] To be more precise: Every subgraph H_i contains a rooted spanning tree of depth at most $24 \cdot r - 8$.
[2] This was not mentioned explicitly by the authors but follows immediately from their construction.

Busch, LaFortune, and Tirthapura already presented an algorithm that computes a sparse r-cover [BLT14] where each subgraph H_i contains a spanning tree of depth at most $16r$, implying a diameter of at most $32r$. However, they proved a polynomial running time but do not present any precise order of magnitude. For our purposes it suffices to know that the diameters of the subgraphs are in $O(r)$. Hence, we will use the older result for which we have a precise running time.

5.3 Shortest-Path Separator Decomposition

In this section we want to look into a well-known hierarchical decomposition of planar graphs – *the shortest-path separator decomposition*. Throughout the whole section we will follow the presentation of Thorup [Tho04], in which only sketches of proofs were given. We provide the details for completeness. Another presentation of a shortest-path separator decomposition is given by Kawarabayashi, Sommer, and Thorup [KST13]. First of all, we have to introduce some notions that are necessary for the description of the decomposition.

The separation lemma. Let $H = (V, E)$ be a connected planar graph and let $T = (V, E_T)$ be a spanning tree of H rooted arbitrarily. Moreover, for $v \in V$ we define $T(v)$ as the unique path from the root to v in T. We extend the notion of $T(\cdot)$ to sets as follows: for $W \subseteq V$, let $T(W)$ be the tree that contains $T(w)$ for each $w \in W$, i.e.,

$$T(W) = \bigcup_{w \in W} T(w).$$

Here, the union of two graphs is defined as usual, that is $H_1 \cup H_2 = (V_{H_1} \cup V_{H_2}, E_{H_1} \cup E_{H_2})$. In addition to that, we need a second type of union for graphs. Let H_1 and H_2 be two subgraphs of H with disjoint vertex sets. The expression $H_1 +_H H_2$ denotes the graph that contains the vertices and edges of $H_1 \cup H_2$ but also contains all the edges of H that have one endpoint in H_1 and the other in H_2. We omit the index $_H$ if the context is clear. Formally, it is

$$H_1 + H_2 = \big(V_{H_1} \cup V_{H_2}, E_{H_1} \cup E_{H_2} \cup \{\{u, v\} \in E \mid u \in V_{H_1}, v \in V_{H_2}\}\big).$$

Finally, for each subset $W \subseteq V$ let $\chi_W \colon V \to \{0, 1\}$ be the characteristic function of W, i.e., $\chi_W(v) = 1$ if and only if $v \in W$. The following lemma is due to Mikkel Thorup [Tho04] and plays a crucial role in the development of the shortest-path separator decomposition for planar graphs.

Lemma 5.3. *Let H be a connected, planar n-vertex graph, T a spanning tree of H, and let $\chi \colon V_H \to \mathbb{R}_0^+$ be a non-negative function. We can find in $O(n)$ time three vertices $r, s, t \in V_H$ such that for every connected component H_i of $H \setminus T\big(\{r, s, t\}\big)$ we have*

$$\sum_{v \in V_{H_i}} \chi(v) \leqslant \frac{1}{2} \sum_{v \in V_H} \chi(v).$$

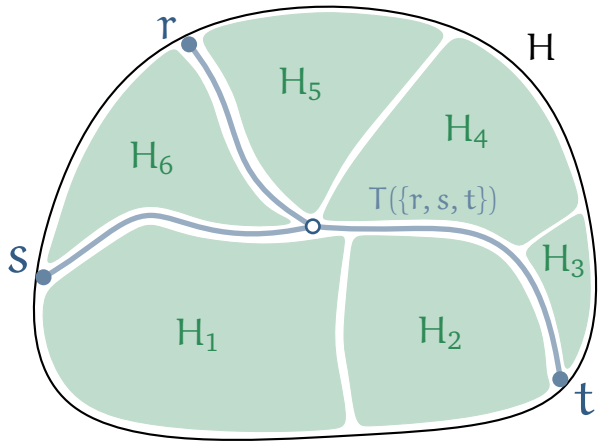


Figure 5.2: The algorithm SEPARATE computes a separator $T(\{r, s, t\})$ of H whose removal decomposes the graph into connected subcomponents $H_1, \ldots, H_6 \in \mathcal{C}$. We have $\chi(H_i) \leqslant \chi(H)/2$ for each component $H_i \in \mathcal{C}$.

Thorups lemma is similar to the classic version that has been proven by Lipton and Tarjan [LT79]. They give a construction that finds 2 paths instead of 3 but the components have 2/3 of the original weight. Thorup argued that his separation provides smaller constants when applied recursively. We want to follow his arguments and use $\text{SEPARATE}(H, T, \chi)$ to denote the implementation by Thorup [Tho04] that returns a triple (r, s, t) and a set of components $\mathcal{C}$ according to Lemma 5.3; see also Figure 5.2. Here, we can assume that SEPARATE outputs an explicit representation for each component $H_i \in \mathcal{C}$, where vertices and edges in H_i might have pointers to their copies in H, if necessary.[3] The following lemma is now immediate.

Lemma 5.4. *Let H, T and χ be as in Lemma 5.3 and $(r, s, t, \mathcal{C}) = \text{SEPARATE}(H, T, \chi)$. Moreover, let $H_i \in \mathcal{C}$ be a component of H and $a \in V_{H_i}$ one of its vertices. Then the subtree of T rooted at a is a subgraph of H_i, i.e., for each $b \in V$ with $a \in V_{T(b)}$ we have $b \in V_{H_i}$.*

Proof. First, assume for the sake of contradiction that b is a vertex of the separator $T(\{r, s, t\})$. Since a is an ancestor of b in T, the vertex a has to be in the separator as well – a contradiction.

Secondly, assume that b is a vertex of a second component $H_j \in \mathcal{C}$ (with $i \neq j$). Let π be the subpath of $T(b)$ with endpoints a and b. Since $T(\{r, s, t\})$ is a separator, every path in H between b and a, especially π, has to use a vertex of $T(\{r, s, t\})$. Next, since a and b are not part of the separator, the separator intersects π in a vertex x that is distinct from a and b. Let y be the lowest common ancestor of a and x in T. Since y is an ancestor of x it is an element of the separator and therefore, it is distinct from a. We can construct a cycle as follows: start at y, go on $T(b)$ until you reach x and go back to y by using the separator. This cycle only uses edges of the tree T, which is again a contradiction. Hence, the claim follows. □

[3] In fact, given H, T, r, s, and t, the components can be computed in linear time, see also[HT73].

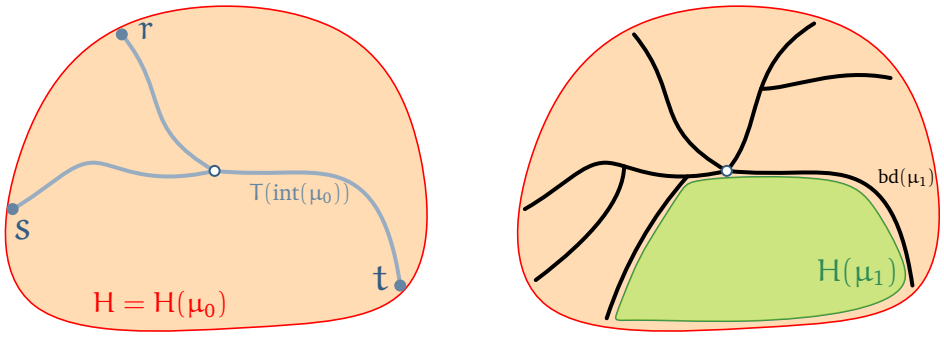


Figure 5.3: Left: The root μ_0 of $\mathcal{T}$ does not have a boundary but internal separators. Right: A leaf μ_1 of $\mathcal{T}$ consists of a subgraph $H(\mu_1)$ of H and a boundary $\text{bd}(\mu_1)$ which is, according to Lemma 5.8 disjoint from $H(\mu_1)$.

The decomposition tree. We want to present an algorithm that takes a planar graph H as well as a shortest-path spanning tree T of H (rooted arbitrarily) as input and computes a shortest-path separator decomposition. The overall idea of this algorithm is as follows: We use Lemma 5.3 to decompose the graph into smaller parts and then recurse on these parts. This process continues until the parts have constant size. During the recursion we alternate between two types of decomposition: we either reduce the size of the subgraph or the number of pieces being on the boundary of the subgraph. This recursion yields a *decomposition tree* which we denote by $\mathcal{T}(H, T)$. We abbreviate it by $\mathcal{T}$ since H and T are clear from the context.

Before we describe the computation of such a decomposition tree, we introduce the essential notation. Let μ be a node of the decomposition tree. It is assigned a set of vertices $V(\mu) \subseteq V$ whose induced subgraph $H(\mu)$ is a connected component of H. Moreover, μ gets two different sets of shortest-path separators: the *external separator paths* and the *internal separator paths*. These paths are root-to-node-paths in the shortest-path tree T and thus, they are shortest paths. Since $T(\cdot)$ is a bijection between the vertices of H and the root-to-node-paths in T, it suffices to store the endpoints distinct from the root, i.e., $\text{ext}(\mu)$ is a set of endpoints of the external separator paths $T(\text{ext}(\mu))$. Analogously, $\text{int}(\mu)$ is a set of endpoints of the internal separator paths. The external separator paths serve as the *boundary* of $V(\mu)$, i.e., every path in H with one endpoint inside $V(\mu)$ and one endpoint outside of $V(\mu)$ intersects $T(\text{ext}(\mu))$. We will prove this statement in Lemma 5.11. We use $\text{bd}(\mu)$ as notation for $T(\text{ext}(\mu))$ and call $T(r)$ *boundary part*, for $r \in \text{ext}(\mu)$. Finally, the node μ is assigned its depth in $\mathcal{T}$, denoted by $\text{depth}(\mu)$. This depth is not necessary for the data structure but will help us with the analysis.

The computation. We refer to Algorithm 5.1 which presents the pseudocode of the computation of $\mathcal{T}$. Let μ be a node created during the execution of $\textsc{spsd}(H, T)$. If the node μ has depth 0, i.e., μ is the root, then $H(\mu)$ is the input graph H and μ has no external separator paths, see Line 4 of the algorithm and Figure 5.3 (left). Next, if $V(\mu)$ is of constant size, μ is a leaf and will not have any internal separator paths, see Line 9 and Figure 5.3 (right).

Algorithm 5.1 The shortest-path separator decomposition.

```
1:  procedure SPSD(graph H, shortest-path spanning tree T of H)
2:      Outputs: shortest-path separator decomposition tree 𝒯
3:      create a new node μ
4:      V(μ) = V_H; ext(μ) = ∅; depth(μ) = 0
5:      return NODE(μ)
6:  procedure NODE(μ)
7:      Outputs: shortest-path separator decomposition subtree of 𝒯 for H(μ)
8:      if |V(μ)| is constant then
9:          children(μ) = ∅; int(μ) = ∅;
10:         return μ
11:     H' = H(μ) + bd(μ)
12:     T' = T(V(μ)) ∪ bd(μ)
13:     W = (depth(μ) mod 2 == 1) ? ext(μ) : V(μ)
14:     (r, s, t, 𝒞) = SEPARATE(H', T', χ_W)
15:     int(μ) = {r, s, t}
16:     for H_i ∈ 𝒞 with V_{H_i} \ V_{bd(μ)} ≠ ∅ do
17:         create a new node σ_i
18:         V(σ_i) = V_{H_i} \ V_{bd(μ)}
19:         ext(σ_i) = int(μ) ∪ (ext(μ) ∩ V_{H_i})
20:         depth(σ_i) = depth(μ) + 1
21:         children(μ).add(NODE(σ_i))
22:     return μ
```

Suppose $V(\mu)$ is not of constant size. We focus on the subgraph $H' = H(\mu) + \mathrm{bd}(\mu)$, see Line 11. Let T' be the subtree of T spanning H', see Line 12. Next, we compute a suitable separator for H'. As mentioned, we distinguish two cases depending on the parity of $\mathrm{depth}(\mu)$, see Line 13 as well as Figure 5.4 for both cases. If the depth is even, we want to reduce the size of the subgraph $H(\mu)$. Hence, we compute a separator $T(\{r, s, t\})$ of H' such that each component of $H' \setminus T(\{r, s, t\})$ has at most half the number of vertices of $H(\mu)$. Otherwise, if the depth is odd, we reduce the number of boundary parts, i.e., the size of $\mathrm{ext}(\mu)$. For this, we split H' so that the boundary of each subcomponent has half the number of separator paths. In both cases we recurse on the subpieces, see Lines 16-21. Remark, that the internal separator paths of μ become external separator paths of all the children of μ, see Line 19. The whole process is demonstrated for the subgraph reducing case in Figure 5.5.

Lemma 5.5. *Let μ be a node of $\mathcal{T}$ and $x \in \mathrm{int}(\mu) \cup \mathrm{ext}(\mu)$. Then $T(x)$ is a shortest path in H.*

Proof. This follows from Lemma 5.3 and the fact, that T is a shortest-path tree. □

Lemma 5.6. *Let μ be a node of $\mathcal{T}$ and σ_1 and σ_2 two distinct children of μ. Then $V(\sigma_1)$ and $V(\sigma_2)$ are disjoint and $V(\sigma_i) \subseteq V(\mu)$, for $i \in \{1, 2\}$.*

Proof. This follows from Lemma 5.3 as well as Line 18. □

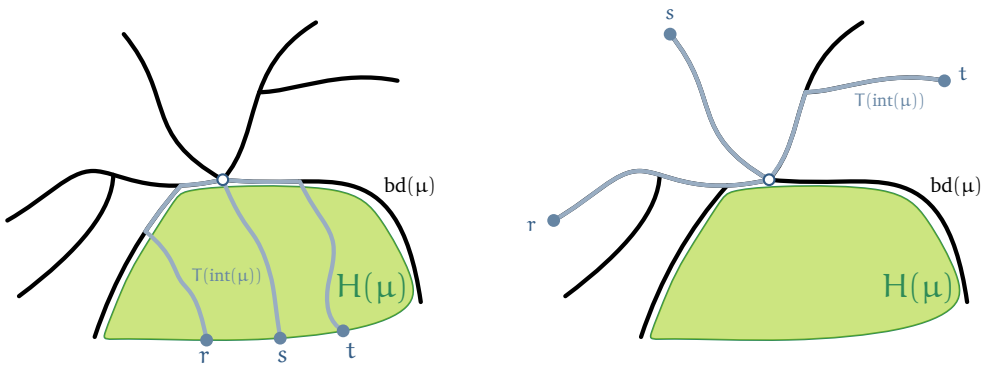


Figure 5.4: Left: A subgraph reducing node of even depth. Right: A boundary reducing node of odd depth. Observe that μ will only have one child σ with $H(\mu) = H(\sigma)$ but less boundary parts.

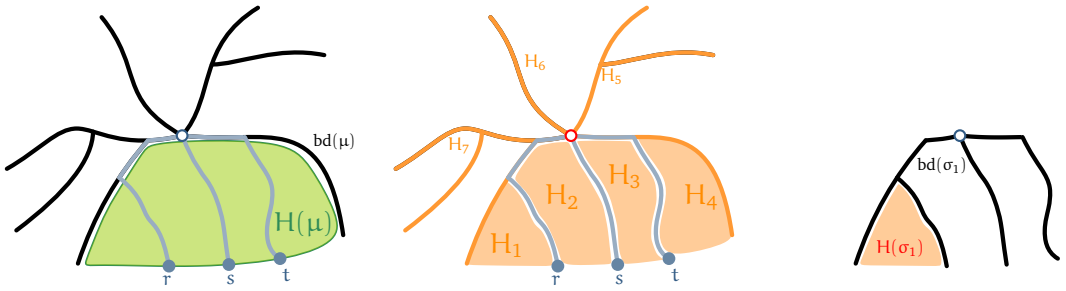


Figure 5.5: The subgraph reducing process. After $\text{int}(\mu)$ has been computed, the graph is split into subgraphs $H_1, \ldots, H_7$. In this case, there will be a node σ_i for $i = 1, \ldots, 4$. The subgraphs H_5, H_6, and H_7 are discarded by Line 16 of the algorithm.

Lemma 5.7. *Let μ be a node of $\mathcal{T}$ and $x \in \text{ext}(\mu)$. Then there is an ancestor μ' of μ in $\mathcal{T}$ such that $x \in \text{int}(\mu')$.*

Proof. New separator paths are only created in Line 14. They become the internal separator paths of the corresponding nodes and are then passed down to its children together with the external paths, see Line 19. The claim follows from a simple inductive argument. □

Lemma 5.8. *Let σ be a node of $\mathcal{T}$. Then we have $V(\sigma) \cap V_{\text{bd}(\sigma)} = \emptyset$.*

Proof. The statement is obvious for the root, see Line 4. Thus, assume that σ is not the root. Let μ be the parent node of σ. Moreover, let $\mathcal{C}$ be the set of components and $\text{int}(\mu) = \{r, s, t\}$ the leafs of the separator paths computed in Line 14. We can conclude that the node σ was created with respect to a subgraph $H_i \in \mathcal{C}$. Since $V(\sigma) = V_{H_i} \setminus V_{\text{bd}(\mu)}$, by Line 18, it suffices to show that $T(\text{int}\,\mu)$ does not share any vertices with $V(\sigma)$, by Line 19. However, this is clear, since $T(\{r, s, t\})$ is the internal separator of $H(\mu)$ and therefore, it is disjoint from $V(\sigma)$. The claim follows. □

Lemma 5.9. *Let μ be a node of depth d in $\mathcal{T}$ and σ a child of μ.*

(i) If d is even, then $|V(\sigma)| \leqslant \frac{1}{2}|V(\mu)|$ and $|\operatorname{ext}(\sigma)| \leqslant |\operatorname{ext}(\mu)| + 3$.

(ii) If d is odd, then $|V(\sigma)| \leqslant |V(\mu)|$ and $|\operatorname{ext}(\sigma)| \leqslant \frac{1}{2}|\operatorname{ext}(\mu)| + 3$.

Proof. Let $(r, s, t, \mathcal{C})$ be the result of the computation in Line 14 with $\operatorname{int}(\mu) = \{r, s, t\}$. Let $H_i \in \mathcal{C}$ be the component for which the child σ was created in Line 17 and let $H(\sigma) = H_i \setminus \operatorname{bd}(\mu)$. By Lemma 5.8, we know that $V(\mu)$ and $V_{\operatorname{bd}(\mu)}$ are disjoint and thus, $H(\sigma)$ is a subgraph of $H(\mu)$. Hence, we get $|V(\sigma)| \leqslant |V(\mu)|$. Additionally, from Line 19, we can derive $|\operatorname{ext}(\sigma)| \leqslant |\operatorname{ext}(\mu)| + 3$. Both results do not depend on the parity of the depth of μ. The other two inequalities will depend on it.

For the first case, let d be even, then $\chi = \chi_{V(\mu)}$ was used for the separation. Furthermore, we know by Lemma 5.8 that $\chi(v) = 0$ for each $v \in V_{\operatorname{bd}(\mu)}$. Hence, we can use Lemma 5.3 to derive

$$|V(\sigma)| = \sum_{v \in V(\sigma)} \chi(v) = \sum_{v \in V_{H_i}} \chi(v) \leqslant \frac{1}{2} \sum_{v \in V_{H'}} \chi(v) = \frac{1}{2} \sum_{v \in V(\mu)} \chi(v) = \frac{1}{2}|V(\mu)|.$$

For the second case, let d be odd, then $\chi = \chi_{\operatorname{ext}(\mu)}$. From Lemma 5.3, we conclude

$$|\operatorname{ext}(\sigma)| \leqslant |\operatorname{ext}(\mu) \cap V_{H_i}| + 3 \leqslant \sum_{v \in V_{H_i}} \chi(v) + 3 \leqslant \frac{1}{2} \sum_{v \in V_{H'}} \chi(v) + 3 = \frac{1}{2}|\operatorname{ext}(\mu)| + 3.$$

This finishes the proof. □

Lemma 5.10. *The algorithm terminates and $\mathcal{T}$ has height $O(\log n)$.*

Proof. By Lemma 5.9, the size of $V(\cdot)$ is halved in every second level. Once, the size is constant, there are no more recursive calls. Hence, the algorithm terminates and the resulting tree has height $O(\log n)$. □

As stated earlier, every path between vertices of different components must intersect a separator path. We can now formally prove that a path that leaves a component $H(\mu)$ must first intersect its boundary $\operatorname{bd}(\mu)$.

Lemma 5.11. *Let μ be a node of $\mathcal{T}$. Then, we have $N[V(\mu)] \subseteq V(\mu) \cup V_{\operatorname{bd}(\mu)}$.*

Proof. Let $\{v, w\} \in E_H$ be an edge such that $v \in V(\mu)$ and $w \notin V(\mu)$. We need to show that $w \in V_{\operatorname{bd}(\mu)}$. Let P be the set of all nodes μ_i of $\mathcal{T}$ satisfying $v \in V(\mu_i)$ and $w \notin V(\mu_i)$, we let $k = |P|$. By Lemma 5.6, P is a path in $\mathcal{T}$ with nodes of increasing depth. Let μ_1 be the highest node in P, and let μ_i be the child of μ_{i-1} in P, for each $i = 2, \ldots, k$.

We prove by induction on i, that each node $\mu_i \in P$ satisfies $w \in V_{\operatorname{bd}(\mu_i)}$. Then, the claim follows, since $\mu \in P$. For the induction base, let μ_0 be the parent node of μ_1. By definition of P, we get $v, w \in V(\mu_0)$. Since $w \notin V(\mu_1)$, either $w \in V_{T(\operatorname{int}(\mu_0))}$ or there is a child σ of μ_0 distinct from μ_1 such that $w \in V(\sigma)$. However, the second case is not possible, since the

separator $T(\text{int}(\mu))$ must intersect every path between v and w, especially the edge $\{v, w\}$. Hence, we have $w \in V_{T(\text{int}(\mu_0))} \subseteq V_{T(\text{ext}(\mu_1))} = V_{\text{bd}(\mu_1)}$ as claimed.

For the induction step, let μ_i be a node in P with $i > 1$. By the induction hypothesis we have $w \in V_{\text{bd}(\mu_{i-1})}$. Let $H' = H(\mu_{i-1}) + \text{bd}(\mu_{i-1})$ and let $H_i \in \mathcal{C}$ be the subgraph of H' computed in Line 14 that refers to the node μ_i. Because w is a vertex of H', it is either in $T(\text{int}(\mu_{i-1}))$ or in a subgraph $H_j \in \mathcal{C}$. In the first case, the claim follows from the fact that $T(\text{int}(\mu_{i-1}))$ is a subgraph of $\text{bd}(\mu_i)$, by Line 19. For the second case, w has to be in H_i since it is not possible to separate v from w. Because $w \in \text{bd}(\mu_{i-1})$, there exists an $s \in \text{ext}(\mu_{i-1})$ with $w \in T(s)$. By Lemma 5.4, we know that s is a vertex of H_i. Now, because $s \in \text{ext}(\mu_{i-1}) \cap V_{H_i}$, it follows from Line 17 that $s \in \text{ext}(\mu_i)$. Finally, since $w \in T(s)$ we derive $w \in \text{bd}(\mu_i)$ as claimed. □

Lemma 5.12. *Let μ be a node of $\mathcal{T}$. We have $|\text{int}(\mu)| = 3$ and $|\text{ext}(\mu)| \leqslant 11$.*

Proof. The first statement follows immediately from Line 15. For the second statement we take a look at the following recursively defined function $f : \mathbb{N}_0 \to \mathbb{N}_0$, namely

$$f_d = \begin{cases} 0, & \text{if } d = 0 \\ f_{d-1} + 3, & \text{if } d \text{ is odd} \\ \left\lfloor \frac{f_{d-1}}{2} \right\rfloor + 3, & \text{otherwise} \end{cases}$$

We show by induction on d that $f_d \leqslant 11$. For the induction base, we have $f_0 = 0 \leqslant 11$ as well as $f_1 = 3 \leqslant 11$. Next, for the induction step we let $d \geqslant 2$ and distinguish two cases according to its parity. If d is even, we conclude

$$f_d = \left\lfloor \frac{f_{d-1}}{2} \right\rfloor + 3 \leqslant \left\lfloor \frac{11}{2} \right\rfloor + 3 = 8 \leqslant 11.$$

If d is odd, we derive

$$f_d = f_{d-1} + 3 \leqslant \left\lfloor \frac{f_{d-2}}{2} \right\rfloor + 3 + 3 \leqslant \left\lfloor \frac{11}{2} \right\rfloor + 6 = 11.$$

Finally, let μ be of depth d. If $d = 0$, the node μ does not have any external separators, i.e., $|\text{ext}(\mu)| = f_0$. Otherwise, we can conclude from Lemma 5.9 that $|\text{ext}(\mu)| \leqslant f_d$. This finishes the proof. □

We can also bound the number of nodes in $\mathcal{T}$. Remark that the degree of a node in $\mathcal{T}$ might be one. This, however, will not influence the asymptotic size of $\mathcal{T}$.

Lemma 5.13. *$\mathcal{T}$ has $O(n)$ nodes.*

Proof. By Lemma 5.6, we know that the leafs of $\mathcal{T}$ represent a partition of a subset $V' \subseteq V$. Hence, $\mathcal{T}$ has at most n leafs. Moreover, since the number of vertices in the components is halved in every second step, by Lemma 5.9, the claim follows. □

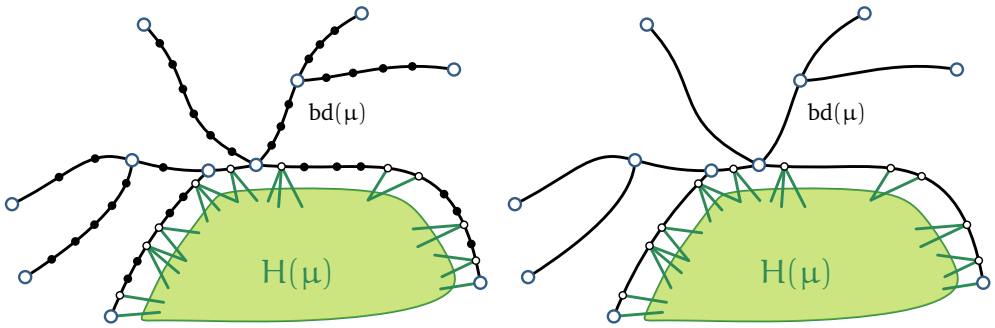


Figure 5.6: The graph $H' = H(\mu) + \mathrm{bd}(\mu)$. The filled black dots represent the unimportant vertices, the large blue dots represent the topological vertices, and the remaining ones are the selected vertices. The green lines are all edges between $H(\mu)$ and $\mathrm{bd}(\mu)$. Left: The boundary with unimportant vertices. Right: After the deletion of all unimportant vertices. The size of the boundary is significantly smaller.

The running time. Since our decomposition tree is later used as building block we are interested in its structure rather than its preprocessing time analysis. However, we briefly sketch the main issues of the running time analysis. We refer to Mikkel Thorups description [Tho04] and follow his argumentation. The following details are not presented in the pseudocode, since they would make its presentation unclear. The idea is depicted in Figure 5.6.

Let us fix a node μ in the tree and let $H' = H(\mu) + \mathrm{bd}(\mu)$. Moreover, let $n(\mu)$ be the number of vertices in H', that is, $n(\mu) = |V(\mu)| + |V_{\mathrm{bd}(\mu)}|$. Next, we let $E(\mu)$ be the set of all edges that have at least one endpoint in $H(\mu)$ and use $e(\mu)$ to denote the cardinality of $E(\mu)$. By Lemmas 5.8 and 5.11, we know that the endpoints of edges in $E(\mu)$ are either in $H(\mu)$ or in $\mathrm{bd}(\mu)$. Finally, we let $t(\mu)$ be the preprocessing time for the node μ without its recursive calls. Then, if μ is an inner node, we have $t(\mu) \in O(n(\mu))$ by Lemma 5.3, and if μ is a leaf, $t(\mu) \in O(n(\mu))$ holds trivially. Thus, we have to be sure that $n(\mu)$ is not too large. For this, we adapt $\mathrm{bd}(\mu)$ appropriately.

Recall, that $\mathrm{bd}(\mu)$ is a tree. We call a vertex v of $\mathrm{bd}(\mu)$ *topological* if it is a leaf or a branching vertex, i.e., it does not have degree 2 in $\mathrm{bd}(\mu)$. Observe, that by Lemma 5.12, the boundary must have a constant number of topological vertices. A vertex v is called *selected* if it is adjacent to a vertex in $H(\mu)$, i.e., $v \in N(V(\mu))$. The vertices that are neither topological nor selected are *unimportant*.

Now, to decrease the number of vertices in $\mathrm{bd}(\mu)$, we iterate over all unimportant vertices v and do the following: Let u and w be the two neighbors of v in $\mathrm{bd}(\mu)$, we remove v (and its two incident edges) from $\mathrm{bd}(\mu)$ and introduce a new edge $\{u, w\}$ whose weight is the sum of the weights of the two deleted edges. This iteration is performed in the recursive call where the node μ is created. After this iteration we have a boundary that consists of a constant number of topological vertices and at most $e(\mu)$ selected vertices, see Figure 5.6. This yields $n(\mu) \in O(e(\mu))$.

For the running time, we fix a number h and use L_h to denote the set of all nodes in $\mathcal{T}$ that

have level h. Let $\mu_i, \mu_j \in L_h$ be two distinct nodes. By Lemma 5.6, we know that $V(\mu_i)$ and $V(\mu_j)$ are disjoint. Thus, $E(\mu_i)$ and $E(\mu_j)$ are disjoint as well. From this we can derive

$$\sum_{\mu_i \in L_h} e(\mu_i) \leqslant |E| \in O(n).$$

Plugging everything together, we get

$$\sum_{\mu_i \in L_h} t(\mu_i) \in O\left(\sum_{\mu_i \in L_h} n(\mu_i)\right) = O\left(\sum_{\mu_i \in L_h} e(\mu_i)\right) = O(n).$$

The height of the decomposition tree is $O(\log n)$, see Lemma 5.10, which gives us a near-linear preprocessing time.

Lemma 5.14. *$\mathcal{T}$ can be computed in $O(n \log n)$ time.*

5.4 Approximate Shortest-Path Separator Decomposition

In this section, we want to present a decomposition of unit disk graphs for which the shortest-path separator decomposition is the main building block.

Let $DG(V)$ be a unit disk graph with vertex set V, diameter D, and let $\varepsilon > D^{-1}$. As usual, n denotes the number of vertices of $DG(V)$. We present a data structure for $DG(V)$ which we call *approximate shortest-path separator decomposition* (short: approximate-spsd). The data structure was first presented by Chan and Skrepetos [CS19] and follows an idea of Kawarabayashi, Sommer, and Thorup [KST13]. We slightly adapt the data structure for our purposes and shorten the proofs.

First of all, we compute the planar 2.42-spanner H of $DG(V)$ from Lemma 5.1 which needs $O(n \log n)$ time. Then, we use Dijkstras algorithm to compute in $O(n \log n)$ time a shortest-path tree T of H, rooted arbitrarily. We use this shortest-path tree to build the shortest-path separator decomposition $\mathcal{T}$ for H. According to Lemma 5.14, this can also be done in $O(n \log n)$ time .

We get the approximate-spsd tree $\mathcal{T}_\varepsilon$ by extending the nodes with further information. Therefore, let μ be a node of $\mathcal{T}$. We compute a set of *portals* for μ, denoted by $\text{port}(\mu)$, as follows. If μ is a leaf, we let $\text{port}(\mu) = V(\mu)$. Otherwise, assume μ to be an inner node. Let $x \in \text{int}(\mu)$ be the leaf of the internal separator path $T(x)$. We select a set of $O(1/\varepsilon)$ vertices in $V(\mu) \cap V_{T(x)}$ such that two selected consecutive vertices on $T(x)$ are at distance at most εD. This is possible since $\varepsilon D > 1$ by assumption. The result is denoted by $V_\varepsilon(x)$. Let $\text{int}(\mu) = \{r, s, t\}$. We define the portal set of μ as $\text{port}(\mu) = V_\varepsilon(r) \cap V_\varepsilon(t) \cap V_\varepsilon(t)$; see Figure 5.7. By construction, $\text{port}(\mu) \subseteq V(\mu)$ holds for each node μ. Furthermore, by Lemmas 5.3, 5.6 and 5.8, the portal sets of two distinct nodes μ_1 and μ_2 are disjoint, i.e., $\text{port}(\mu_1) \cap \text{port}(\mu_2) = \emptyset$. Finally, we prove the following lemma that bounds the preprocessing time and the size of the portal sets.

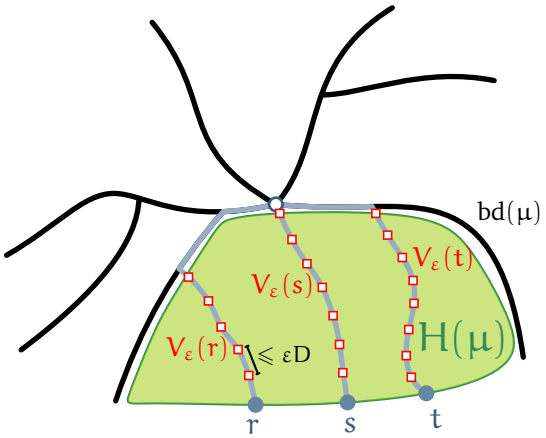


Figure 5.7: The portals of a node μ. Two consecutive portals have distance at most εD.

Lemma 5.15. *The portal sets can be computed in* $O(n)$ *additional time and for each node* μ *of* $\mathcal{T}$ *we have* $|\operatorname{port}(\mu)| \in O(1/\varepsilon)$.

Proof. The second part of the lemma follows on the one hand from the fact that every inner node has a constant number of internal separator paths and on the other hand from the fact that leafs have vertex sets of constant size.

For the running time, we first observe that, by Lemma 5.13, there are only $O(n)$ nodes in $\mathcal{T}$. Next, let μ_1 and μ_2 be two different inner nodes of $\mathcal{T}$. From Lemmas 5.3, 5.6 and 5.8, we can conclude that $V(\mu_1) \cap V_{T(\operatorname{int}(\mu_1))}$ and $V(\mu_2) \cap V_{T(\operatorname{int}(\mu_2))}$ are disjoint. Hence, we need to look at $O(n)$ vertices in total when computing the portal sets of all nodes. This gives a running time of $O(n)$. □

Finally, for each portal $p \in \operatorname{port}(\mu)$, we compute a shortest-path tree rooted at p spanning $DG(V(\mu))$ and store it in μ together with all distances between v and p in $DG(V(\mu))$, denoted by $d_\mu(v, p)$, for each $v \in V(\mu)$. The result is our approximate-spsd tree $\mathcal{T}_\varepsilon$. Wang and Xue have shown how to compute a shortest path tree in $O(n' \log^2 n')$ time if unit disk graph has n' vertices [WX20]. To argue about the size of the whole running time we let L_h be the set of all nodes in $\mathcal{T}_\varepsilon$ of height h. Remark, that by Lemma 5.6 the components located at height h in $\mathcal{T}$ represent a partition of a vertex set $V' \subseteq V$. Moreover, each node μ has at most $O(1/\varepsilon)$ portals. Hence, the running time $t(h)$ over all nodes of height h is bounded by

$$t(h) = \sum_{\mu \in L_h} O\big(\varepsilon^{-1}|V(\mu)| \log^2 |V(\mu)|\big) = O\big(\varepsilon^{-1}|V'| \log^2 |V'|\big) = O\big(\varepsilon^{-1} n \log^2 n\big).$$

Last but not least, since $\mathcal{T}$ and therefore $\mathcal{T}_\varepsilon$ has height $O(\log n)$, by Lemma 5.10, we can conclude the following lemma.

Lemma 5.16. *Let* $DG(V)$ *be an* n*-vertex unit disk graph. We can compute the approximate-spsd tree* $\mathcal{T}_\varepsilon$ *in* $O(\varepsilon^{-1} n \log^3 n)$ *time.*

Distance Oracle. Finally, let us describe what the approximate-spsd tree $\mathcal{T}_\varepsilon$ is used for. First, we know that the portal sets for two distinct nodes μ_1 and μ_2 of $\mathcal{T}_\varepsilon$ are disjoint. Hence let p be a portal, we let $\mu(p)$ be the unique node μ in $\mathcal{T}_\varepsilon$ for which $p \in \text{port}(\mu)$ holds. Let s and t be two vertices of $\text{DG}(V)$. We use $P(s,t)$ to denote the set of all portals that belong to a node μ containing s and t in its component $V(\mu)$, i.e.,

$$P(s,t) = \big\{p \in \text{port}(\mu) \mid s,t \in V(\mu)\big\}.$$

The following observation bounds the size of $P(s,t)$ and is straightforward.

Observation 5.17. $|P(s,t)| \in O\big(\varepsilon^{-1}\log n\big)$.

Proof. The set of nodes μ of $\mathcal{T}_\varepsilon$ with $s,t \in V(\mu)$ is a path with one endpoint being the root. Since $\mathcal{T}_\varepsilon$ has height $O(\log n)$ by Lemma 5.10, and since $\text{port}(\mu)$ has size $O(1/\varepsilon)$, by Lemma 5.15, the claim follows. □

Next, we define

$$\theta(s,t) = \min_{p\in P(s,t)} \big\{d_{\mu(p)}(s,p) + d_{\mu(p)}(p,t)\big\}.$$

The following lemma shows that $\theta(s,t)$ approximates the distance between s and t.

Lemma 5.18. *Let s and t be two vertices of $\text{DG}(V)$. Then we have*

$$d(s,t) \leqslant \theta(s,t) \leqslant d(s,t) + 6.84\varepsilon D.$$

Proof. The left inequality is easy to see, since each $p \in P(s,t)$ satisfies

$$d(s,t) \leqslant d_{\mu(p)}(s,t) \leqslant d_{\mu(p)}(s,p) + d_{\mu(p)}(p,t).$$

For the right inequality, let π be a shortest path in $\text{DG}(V)$ between s and t. Moreover, let μ be the lowest node in $\mathcal{T}_\varepsilon$ such that $\text{DG}(V(\mu))$ contains the path π. First, assume that μ is a leaf of $\mathcal{T}_\varepsilon$. Hence, we can conclude that $s \in \text{port}(\mu)$ and therefore, $s \in P(s,t)$ as well as $\mu(s) = \mu$. Moreover, since π is a shortest path in $\text{DG}(V)$ but also in $\text{DG}(V(\mu))$ we can derive $d(s,t) = d_\mu(s,t)$. Thus, we have

$$\theta(s,t) \leqslant d_{\mu(s)}(s,s) + d_{\mu(s)}(s,t) = d_\mu(s,t) = d(s,t).$$

Otherwise, let us assume that μ is not a leaf of $\mathcal{T}_\varepsilon$. Let σ be the child of μ with $s \in V(\sigma)$, see also Figure 5.8. We start at s and walk along π until we find the first edge $\{u,v\}$ such that $u \in V(\sigma)$ and $v \notin V(\sigma)$. This edge has to exist, since μ is the lowest node containing π. Next, since H is a 2.42-spanner of $\text{DG}(V)$, we know that there is a path $\pi_{u,v}$ in H of length at most $2.42 \cdot |uv|$ connecting u and v. We start at u and walk along $\pi_{u,v}$ until we find the first vertex w that is outside of $V(\sigma)$. This vertex exists, since v is outside of $V(\sigma)$. By Lemma 5.11, we know that w is on the boundary $\text{bd}(\sigma)$ of σ. Let μ' be the lowest ancestor of σ satisfying $w \in V(\mu') \cap V_{T(\text{int}(\mu'))}$. Such a node has to exist, by Lemma 5.7. Observe, that $\mu = \mu'$ is possible. Next, let p be the portal of μ' closest to w. Hence, it is $\mu(p) = \mu'$ and

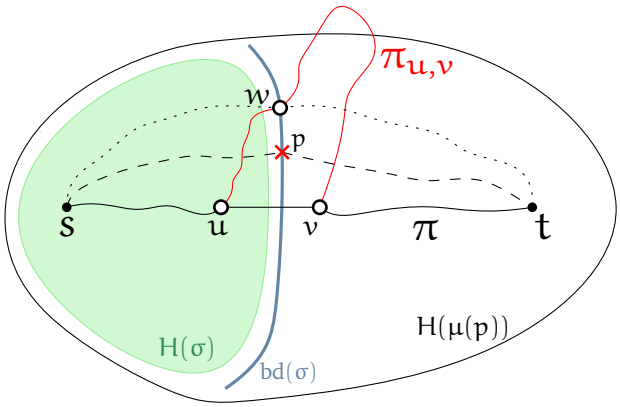


Figure 5.8: The vertex v is the first vertex on the shortest path π outside of $H(\sigma)$. The path $\pi_{u,v}$ is in H and approximates the edge $\{u,v\}$ which might not be in H. The vertex w is the first vertex on $\pi_{u,v}$ outside of $H(\sigma)$. It lies on $\mathrm{bd}(\sigma)$ and has p as closest portal.

by construction, we have $d_{\mu(p)}(w,p) = d(w,p) \leqslant \varepsilon D$. Moreover, since $V(\mu(p)) \supseteq V(\mu)$, we conclude $d_{\mu(p)}(s,t) = d_\mu(s,t) = d(s,t)$. Finally, we can use the triangle inequality to derive

$$\begin{aligned}\theta(s,t) &\leqslant d_{\mu(p)}(s,p) + d_{\mu(p)}(p,t) \leqslant d_{\mu(p)}(s,w) + d_{\mu(p)}(w,t) + 2\cdot d_{\mu(p)}(w,p)\\ &\leqslant d_{\mu(p)}(s,u) + d_{\mu(p)}(u,t) + 2\cdot|\pi_{u,v}| + 2\cdot d_{\mu(p)}(w,p)\\ &\leqslant d_{\mu(p)}(s,t) + 4.84\cdot|uv| + 2\cdot d(w,p) \leqslant d(s,t) + 6.84\cdot\varepsilon D.\end{aligned}$$

This finishes the proof. □

Last but not least, we want to summarize all the necessary information in one theorem.

Theorem 5.19. *Let* $\mathrm{DG}(V)$ *be an* n*-vertex unit disk graph of diameter* D *and let* $\varepsilon > D^{-1}$*. We can compute a* decomposition tree $\mathcal{T}_\varepsilon$ *for* $\mathrm{DG}(V)$ *that has the following properties.*

(i) $\mathcal{T}_\varepsilon$ *has* $O(n)$ *nodes and height* $O(\log n)$*.*

(ii) $\mathcal{T}_\varepsilon$ *can be computed in time* $O(\varepsilon^{-1} n \log^3 n)$*.*

(iii) Every node μ *of* $\mathcal{T}_\varepsilon$ *is assigned two sets:* $\mathrm{port}(\mu) \subseteq V(\mu) \subseteq V$*. The subgraph of* $\mathrm{DG}(V)$ *induced by* $V(\mu)$ *is connected and it is* $|\mathrm{port}(\mu)| \in O(1/\varepsilon)$*.*

(iv) If μ *is the root, then* $V(\mu) = V$*.*

(v) If μ *is an inner node with children* $\sigma_1, \ldots, \sigma_k$*, the sets* $\mathrm{port}(\mu), V(\sigma_1), \ldots, V(\sigma_k)$ *are pairwise disjoint, and we have* $V(\sigma_i) \subseteq V(\mu)$*, for* $1 \leqslant i \leqslant k$*.*

(vi) For each portal p*, there is a shortest-path tree* T_p *rooted at* p *spanning* $\mathrm{DG}(V(\mu(p)))$*.*

(vii) *If* μ_1 *and* μ_2 *are two distinct nodes, then* $\mathrm{port}(\mu_1)$ *and* $\mathrm{port}(\mu_2)$ *are disjoint.*

(viii) *For every pair of vertices* s *and* t *it holds*

$$d(s,t) \leqslant \theta(s,t) = \min_{p \in P(s,t)} \left\{ d_{\mu(p)}(s,p) + d_{\mu(p)}(p,t) \right\} \leqslant d(s,t) + 6.84 \cdot \varepsilon D.$$

Remark. If we use the approximate SSSP algorithm proposed by Wan et al. [WX20] it is possible to decrease the preprocessing time of $\mathcal{T}_\varepsilon$ to $O(\varepsilon^{-1} n \log^2 n + \varepsilon^{-1} n \log n \log^2(\varepsilon^{-1}))$. However, this would increase the constant in the stretch and will make the description of the routing scheme more cumbersome.

CHAPTER 6

Unit Disk Graphs

In this section, we describe the building blocks for our routing scheme. For this, we review some simple routing schemes from the literature, and we show how to obtain a new routing scheme for unit disk graphs that achieves an *additive* stretch. This later scheme is based on the data structure of Chan and Skrepetos [CS19], which we described in Section 5.4.

6.1 Small Diameter

First, we present a routing scheme that is efficient for unit disk graphs with small diameter. For this scheme, we use the labeled routing scheme for graphs with bounded doubling dimension by Konjevod, Richa, and Xia [KRX16]. The *doubling dimension* of a graph G is the smallest value α such that any ball $B(v, r)$ of G can be covered by at most 2^α balls of radius at most $r/2$. The following lemma is due to Konjevod et al. [KRX16]. The preprocessing time has been analyzed by Nils Goldmann [Gol21].

Lemma 6.1. *Let G be an n-vertex graph with doubling dimension α. Furthermore, let $\varepsilon > 0$. There is a routing scheme with label size $\lceil \log n \rceil$, table size $(1/\varepsilon)^{O(\alpha)} \log^3 n$, and header size $O(\log^2 n / \log\log n)$, whose routing function achieves stretch factor $1 + \varepsilon$. The preprocessing time is $O(\varepsilon^{-1} n^2 \log^2 n)$.*

The following lemma bounds the doubling dimension of a unit disk graph in terms of D.

Lemma 6.2. *Let $DG(V)$ be a unit disk graph, $v \in V$ a vertex, and $r > 0$. We can cover the ball $B = B(v, r)$ with $O(\max(1, r^2))$ balls of diameter at most $r/2$.*

Proof. Let $E \subset \mathbb{R}^2$ be the Euclidean disk of radius r centered at v. Obviously, $B \subset E$. Moreover, the Euclidean disk E can be covered by a set $\mathcal{E}$ of $K = O(\max(1, r^2))$ Euclidean disks each of radius $r' = \min(r/4, 1/2)$. This follows from a simple covering argument. For each disk $E_i \in \mathcal{E}$, we fix a vertex v_i as follows: if $E_i \cap B \neq \emptyset$, then v_i is an arbitrary vertex of $E_i \cap B$.

Otherwise, if $E_i \cap B = \emptyset$, we let v_i be an arbitrary vertex of B. Since $r' \leqslant 1/2$, the vertices in E_i form a clique in $DG(V)$. Hence, we have $E_i \cap B \subseteq B(v_i, 2r')$. Next, from $r' \leqslant r/4$ we get $B(v_i, 2r') \subseteq B(v_i, r/2)$. Thus,

$$B(v, r) \subseteq \bigcup_{i=1}^{K} (E_i \cap B) \subseteq \bigcup_{i=1}^{K} B(v_i, 2r') \subseteq \bigcup_{i=1}^{K} B(v_i, r/2).$$

This finishes the proof. □

Finally, the routing scheme for unit disk graphs with small diameter follows from Lemma 6.1 and Lemma 6.2.

Lemma 6.3. *Let $DG(V)$ be an n-vertex unit disk graph with diameter D. Furthermore, let $\varepsilon > 0$. There is a routing scheme with label size $\lceil \log n \rceil$, table size $(1/\varepsilon)^{O(D^2)} \log^3 n$, and header size $O(\log^2 n / \log\log n)$, whose routing function σ_{diam} achieves stretch factor $1+\varepsilon$. The preprocessing time is $O(\varepsilon^{-1} n^2 \log^2 n)$.*

6.2 Large Diameter

In the last section we presented a routing scheme that is efficient for unit disk graphs with low diameter. In this section we present a routing scheme that is efficient for unit disk graphs with large diameter. Let $DG(V)$ be an n-vertex unit disk graph with diameter D, and let $\varepsilon > D^{-1}$. Let $c_n = 2^{\lceil \log n \rceil}$, we define $x_* = \lfloor x \cdot c_n \rfloor$, for each $x \in \mathbb{R}_0^+$. We assume that the conversion from x to x_* needs $O(\log x + \log n)$ time.

The labels and tables. We start with a description of the labels and tables and analyze its size and preprocessing time. First, we assign to each vertex $v \in V$ a unique number $v_{\text{id}} \in [n-1]$. Then, we compute the approximate-spsd tree $\mathcal{T}_\varepsilon$ of $DG(V)$. Let μ be a node of $\mathcal{T}_\varepsilon$ and let $p \in \text{port}(\mu)$ be a portal of μ, i.e., $\mu = \mu(p)$. We perform a *postorder enumeration* of the vertices in the shortest-path tree T_p. We use $r_p(v)$ to denote the postorder number of v in T_p, for each $v \in V(\mu)$. Next, the subtree of T_p rooted at v is called $T_p(v)$ and we use $\ell_p(v)$ to denote the smallest postorder number in $T_p(v)$. The postorder enumeration provides the following observation.

Observation 6.4. *Let $w \in V(\mu(p))$. Then we have:*

$$w \in T_p(v) \Leftrightarrow r_p(w) \in [\ell_p(v), r_p(v)].$$

Finally, we apply the tree routing from Lemma 2.2 to T_p and denote by $\text{lab}_p(\cdot)$ and $\text{tab}_p(\cdot)$ the corresponding labels and tables, respectively. Now, for each $v \in V(\mu)$, we store $(p_{\text{id}}, d_{\mu(p)}(v,p)_*, r_p(v), \text{lab}_p(v))$ in $\text{lab}(v)$ and $(p_{\text{id}}, d_{\mu(p)}(v,p)_*, \ell_p(v), r_p(v), \text{tab}_p(v))$ in $\text{tab}(v)$. The following two lemmas bound the size of the labels and the preprocessing time. Finally, Lemma 2.2 implies that routing table and routing label have the same asymptotic size.

Lemma 6.5. *For every vertex $v \in V$, we have* $|\mathrm{lab}(v)| \in O\left(\varepsilon^{-1} \log^3 n / \log\log n\right)$.

Proof. Since $\mathcal{T}_\varepsilon$ has height $O(\log n)$, we know that v is in $O(\log n)$ different sets $V(\mu)$. Moreover, for every node μ, there are at most $O(1/\varepsilon)$ portals. Thus, the label of v contains $O(\varepsilon^{-1} \cdot \log n)$ different entries. The value $d_{\mu(p)}(v,p)_*$ is a natural number, and since $c_n \leqslant 2n$, we have

$$d_{\mu(p)}(v,p)_* = \lfloor d_{\mu(p)}(v,p) \cdot c_n \rfloor \leqslant 2n^2.$$

Thus, we need $O(\log n)$ bits for the number $d_{\mu(p)}(v,p)_*$. Moreover, the identifier p_{id} as well as the postorder numbers stored in one entry only need $O(\log n)$ bits. Finally, we apply Lemma 2.2 to conclude that one entry of the routing label has size $O(\log^2 n / \log\log n)$. The claim follows. □

Lemma 6.6. *The preprocessing time of the labels and tables is* $O(\varepsilon^{-1} n \log^3 n)$.

Proof. First, the approximate-spsd can be computed in time $O(\varepsilon^{-1} n \log^3 n)$, see Lemma 5.16. Next, let us fix a depth d and let μ be a node of depth d in $\mathcal{T}_\varepsilon$. If μ does not contain any portals, we do nothing. Otherwise, let $p \in \mathrm{port}(\mu)$. The computation of the postorder numbers $\ell_p(\cdot)$ and $r_p(\cdot)$ of the tree T_p can be done in time $O(|V(\mu)|)$. Moreover, the approximate distances $d_{\mu(p)}(\cdot, p)$ can be computed in time $O(|V(\mu)| \log n)$, since the correct distances are stored in μ and since $d_{\mu(p)}(v,p) \leqslant n$ for each $v \in V(\mu)$. The labels $\mathrm{lab}_p(\cdot)$ and tables $\mathrm{tab}_p(\cdot)$ can be computed in time $O(|V(\mu)|)$ because of Lemma 2.2. By Lemma 5.15, the node μ has $O(1/\varepsilon)$ different portals. Thus, we can conclude that the processing for one node μ needs $O(\varepsilon^{-1}|V(\mu)| \log n)$ time. Furthermore, by Lemma 5.6, the vertex sets of all nodes of depth d are a partition of a subset $V' \subseteq V$. According to Lemma 5.10, $\mathcal{T}_\varepsilon$ is of height $O(\log n)$ and thus, the computation of all labels and tables needs $O(\varepsilon^{-1} n \log^2 n)$ time once the approximate-spsd has been computed. In the end, the preprocessing time of the labels and tables is dominated by the computation of $\mathcal{T}_\varepsilon$. This finishes the proof. □

The routing function. Next, we describe the routing function. Let us first explain the overall idea of our routing scheme. In the preprocessing we covered the unit disk graph $\mathrm{DG}(V)$ by a set of $O(n/\varepsilon)$ shortest-path trees whose roots are the portals of the approximate-spsd. In every step of the routing we choose one of these trees in a greedy fashion and route one step in this tree. We never use any information from past, meaning that the dynamic header during the routing will remain empty. The analysis will show that the routing will terminate and that the length of the resulting path is related to the number $\theta(\cdot,\cdot)$, which is computed by the approximate distance oracle, see Section 5.4.

For the routing function, we are given the table $\mathrm{tab}(s)$ of the current vertex s and the label $\mathrm{lab}(t)$ for the target vertex t (the header will always be empty); see also Algorithm 6.1. First, we identify all portals $p \in P(s,t)$. We can do this by identifying all vertices p such that the entry $(p_{\mathrm{id}}, d_{\mu(p)}(s,p)_*, \ell_p(s), r_p(s), \mathrm{lab}_p(s))$ is in $\mathrm{tab}(s)$ while in $\mathrm{lab}(t)$ we find the entry $(p_{\mathrm{id}}, d_{\mu(p)}(t,p)_*, r_p(t), \mathrm{lab}_p(t))$. Next, let

$$\theta(s,t;p) = \begin{cases} d_{\mu(p)}(t,p) - d_{\mu(p)}(p,s), & \text{if } t \in T_p(s) \\ d_{\mu(p)}(t,p) + d_{\mu(p)}(p,s), & \text{otherwise.} \end{cases}$$

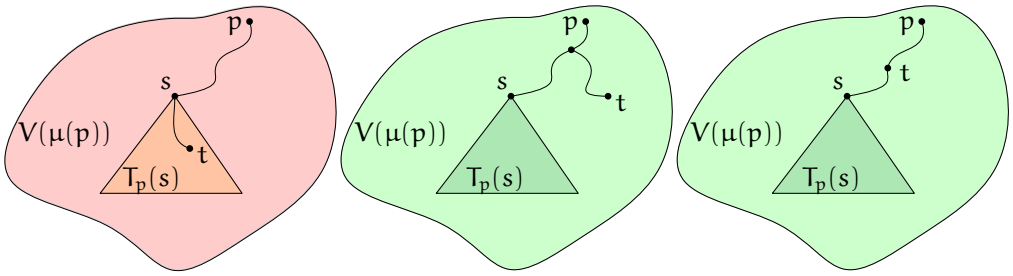


Figure 6.1: Left: If t is in $T_p(s)$, i.e., $\theta(s,t;p) = d_{\mu(p)}(t,p) - d_{\mu(p)}(p,s)$, we route away from p. Middle and Right: If t is not in $T_p(s)$, i.e., $\theta(s,t;p) = d_{\mu(p)}(t,p) + d_{\mu(p)}(p,s)$, we route towards p. The right picture suggests to define $\theta(s,t;p)$ as $d_{\mu(p)}(s,p) - d_{\mu(p)}(t,p)$. This does not influence the guarantees of our routing scheme but would lead to more cases.

See Figure 6.1, for an illustration of the two cases. Let p_{opt} be the portal that minimizes $\theta(s,t;p)$ among all portals p. Then, it is easy to see, that $d(s,t) \leqslant \theta(s,t;p_{opt}) \leqslant \theta(s,t)$. Hence, $\theta(s,t;p_{opt})$ is a good approximation for the distance between s and t. However, the routing function cannot compute the optimal portal p_{opt}, since we do not have direct access to the real value $d_{\mu(p)}(s,p_{opt})$. Instead, we use the values $d_{\mu(p)}(\cdot,p)_*$ to compute a near-optimal portal. Analogously to $\theta(s,t;p)$, we define

$$\theta_*(s,t;p) = \begin{cases} d_{\mu(p)}(t,p)_* - d_{\mu(p)}(p,s)_*, & \text{if } t \in T_p(s) \\ d_{\mu(p)}(t,p)_* + d_{\mu(p)}(p,s)_*, & \text{otherwise.} \end{cases}$$

Let p_0 be the portal that lexicographically minimizes $(\theta_*(s,t;p), p_{id})$, among all portals $p \in P(s,t)$. We call p_0 the s-t-*portal* and set $\theta_*(s,t) = \theta_*(s,t;p_0)$. Observe that the s-t-portal can be computed by using only the table of s, the label of t and Observation 6.4. The routing function now uses $tab_{p_0}(s)$ and $lab_{p_0}(t)$ to compute the next vertex in T_{p_0} and forwards the data packet to this vertex.

Algorithm 6.1 The routing function σ_{add} for unit disk graphs with additive stretch $O(\varepsilon D)$.

1: **procedure** ROUTINGFUNCTION($tab(s) \in \Sigma^*$, $lab(t) \in \Sigma^*$)
2: **Outputs:** port $p \in \Sigma^*$
3: $p_0 = \operatorname{argmin}\{(\theta_*(s,t;p), p_{id}) \mid p \in P(s,t)\}$
4: **return** $\sigma_{tree}(tab_{p_0}(s), lab_{p_0}(t))$

The stretch. Finally, we have to show that the routing scheme is correct and routes along a short (not necessarily shortest) path. For this, we first show that the routing process terminates.

Lemma 6.7. *Let s be the current vertex, t the target vertex, and suppose that the routing scheme sends the packet from s to v. Moreover, let p_0 be the s-t-portal. Then, p_0 is a possible candidate for the v-t-portal, and we have $\theta_*(s,t;p_0) \geqslant \theta_*(v,t;p_0) + |sv|_*$.*

Proof. First, let $\mu = \mu(p_0)$. Since $\{s,v\}$ is an edge of the shortest-path tree T_{p_0}, it follows that $v \in V(\mu(p_0))$. This gives the first part of the claim. For the second part, we distinguish two cases:
Case 1: $t \in T_{p_0}(s)$. In this case, we have $t \in T_{p_0}(v)$, and thus $\theta_*(v,t;p_0) = d_\mu(t,p_0)_* - d_\mu(p_0,v)_*$. Moreover, we have

$$\begin{aligned} d_\mu(p_0,v)_* &= \lfloor d_\mu(p_0,v)\cdot c_n \rfloor = \lfloor d_\mu(p_0,s)\cdot c_n + |sv|\cdot c_n \rfloor \geqslant \lfloor d_\mu(p_0,s)\cdot c_n \rfloor + \lfloor |sv|\cdot c_n \rfloor \\ &= d_\mu(p_0,s)_* + |sv|_*, \end{aligned}$$

since s is on the path in T_{p_0} from p_0 to v. Hence, we get

$$\theta_*(s,t;p_0) = d_\mu(t,p_0)_* - d_\mu(p_0,s)_* \geqslant d_\mu(t,p_0)_* - d_\mu(p_0,v)_* + |sv|_* = \theta_*(v,t;p_0) + |sv|_*.$$

Case 2: $t \notin T_{p_0}(s)$. Similarly to the first case, we have $d_\mu(p_0,s)_* \geqslant d_\mu(p_0,v)_* + |sv|_*$ and $\theta_*(v,t;p_0) \leqslant d_\mu(t,p_0)_* + d_\mu(p_0,v)_*$. Thus, we get

$$\theta_*(s,t;p_0) = d_\mu(t,p_0)_* + d_\mu(p_0,s)_* \geqslant d_\mu(t,p_0)_* + d_\mu(p_0,v)_* + |sv|_* \geqslant \theta_*(v,t;p_0) + |sv|_*,$$

and the claim follows. □

Corollary 6.8. *Let s, t, and v be as in Lemma 6.7. Then, $\theta_*(s,t) \geqslant \theta_*(v,t) + |sv|_*$.*

Proof. Let p_0 be the s-t-portal. From Lemma 6.7, we get

$$\theta_*(s,t) = \theta_*(s,t;p_0) \geqslant \theta_*(v,t;p_0) + |sv|_* \geqslant \theta_*(v,t) + |sv|_*.$$

The claim follows. □

Lemma 6.9. *Let s, t and v be as in Lemma 6.7. Let p be the s-t-portal and q be the v-t-portal. Then, if $\theta_*(s,t) = \theta_*(v,t)$, it follows that $p_{id} \geqslant q_{id}$.*

Proof. From Lemma 6.7, we have

$$\theta_*(v,t;q) = \theta_*(v,t) = \theta_*(s,t) = \theta_*(s,t;p) \geqslant \theta_*(v,t;p) + |sv|_* \geqslant \theta_*(v,t;p) \geqslant \theta_*(v,t;q).$$

Hence, $\theta_*(v,t;p) = \theta_*(v,t;q)$. Furthermore, by construction, we have $(\theta_*(v,t;p), p_{id}) \geqslant (\theta_*(v,t;q), q_{id})$. Thus, the claim follows. □

Lemma 6.10. *The routing scheme is correct.*

Proof. Let s be the current vertex, t the desired target vertex, and p the s-t-portal. To measure the progress towards t, we consider the triple $(\theta_*(s,t), p_{id}, h_p(s,t))$, where $h_p(s,t)$ denotes the hop distance in T_p between s and t.

Suppose that the routing scheme sends the packet from s to v, and let q be the v-t-portal. We argue that $(\theta_*(v,t), q_{\mathrm{id}}, h_q(v,t)) < (\theta_*(s,t), p_{\mathrm{id}}, h_p(s,t)))$. By Corollary 6.8 and Lemma 6.9, it suffices to show that if $\theta_*(s,t) = \theta_*(v,t)$ and $p = q$, then $h_p(s,t) > h_q(v,t)$. However, this is clear, because by Lemma 2.2, $\{s,v\}$ is an edge of T_p that leads from s towards t, and $T_q = T_p$.

Now, since the triples $(\theta_*(s,t), p_{\mathrm{id}}, h_p(s,t))$ lie in $\mathbb{N}^3$ and since $(0,0,0)$ is a global minimum, it follows that the data packet eventually arrives at the target vertex t. □

Lemma 6.11. *For any two vertices s and t, we have $\delta(s,t) \leqslant d(s,t) + 7.84 \cdot \varepsilon D$.*

Proof. First, we show that $\theta_*(s,t) \leqslant c_n \cdot \theta(s,t) + 1$ as follows: let p_0 be the s-t-portal, and let p_{opt} be the portal minimizing $\theta(s,t;\cdot)$ among all portals. Let $\mu = \mu(p_{\mathrm{opt}})$. We obtain

$$\begin{aligned}\theta_*(s,t) = \theta_*(s,t;p_0) &\leqslant \theta_*(s,t;p_{\mathrm{opt}}) = \lfloor c_n \cdot d_\mu(t,p_{\mathrm{opt}})\rfloor \pm \lfloor c_n \cdot d_\mu(p_{\mathrm{opt}},s)\rfloor \\ &\leqslant \lfloor c_n \cdot \big(d_\mu(t,p_{\mathrm{opt}}) \pm d_\mu(p_{\mathrm{opt}},s)\big)\rfloor + 1 \leqslant \lfloor c_n \cdot \theta(s,t)\rfloor + 1 \leqslant c_n \cdot \theta(s,t) + 1,\end{aligned}$$

where the $\pm$-operator is used to cover the two possible cases in the definition of θ_*, and because $\lfloor a\rfloor + \lfloor b\rfloor \leqslant \lfloor a+b\rfloor$ and $\lfloor a\rfloor - \lfloor b\rfloor \leqslant \lfloor a-b\rfloor + 1$, for all $a, b \geqslant 0$.

Next, by Lemma 6.10, we know that the routing terminates. Let $\pi : \langle s = w_0, \ldots, w_m = t\rangle$ be the routing path. From Corollary 6.8, we get $|w_i w_{i+1}|_* \leqslant \theta_*(w_i,t) - \theta_*(w_{i+1},t)$, which yields

$$\begin{aligned}\delta(s,t) = \sum_{i=0}^{m-1} |w_i w_{i+1}| &\leqslant \sum_{i=0}^{m-1} \frac{|w_i w_{i+1}|_* + 1}{c_n} = \frac{m}{c_n} + \frac{1}{c_n}\sum_{i=0}^{m-1} |w_i w_{i+1}|_* \\ &\leqslant \frac{m}{c_n} + \frac{1}{c_n}\sum_{i=0}^{m-1} (\theta_*(w_i,t) - \theta_*(w_{i+1},t)) = \frac{m}{c_n} + \frac{\theta_*(s,t)}{c_n} \\ &\leqslant \frac{m}{c_n} + \frac{c_n \cdot \theta(s,t) + 1}{c_n} = \frac{m+1}{c_n} + \theta(s,t)\end{aligned}$$

Now, using Lemma 5.18, the choices of $c_n = 2^{\lceil \log n\rceil}$ and $\varepsilon D > 1$ as well as the fact that $m \leqslant n-1$, we get

$$\delta(s,t) \leqslant \frac{m+1}{c_n} + \theta(s,t) \leqslant \frac{n}{2^{\lceil \log n\rceil}} + d(s,t) + 6.84 \cdot \varepsilon D \leqslant d(s,t) + 7.84 \cdot \varepsilon D,$$

as claimed. □

We can now conclude with our next theorem.

Theorem 6.12. *Let $\mathrm{DG}(V)$ be an n-vertex unit disk graph with diameter D. Furthermore, let $\varepsilon > D^{-1}$. There is a routing scheme with label size $O\left(\varepsilon^{-1}\log^3 n/\log\log n\right)$ whose routing function σ_{add} routes any data packet on a path with additive stretch $7.84 \cdot \varepsilon D$. The preprocessing time is $O(\varepsilon^{-1} n \log^3 n)$.*

6.3 A Routing Scheme with Stretch $1+\varepsilon$

Let $DG(V)$ be an n-vertex unit disk graph with diameter D and let $\varepsilon > 0$.

The labels and tables. Again, let us first describe the preprocessing phase in which the labels and tables will be computed. First, we compute a planar 2.42-spanner H of $DG(V)$, as in Lemma 5.1. Then, we have $\operatorname{diam}(H) \leqslant 2.42D$. Next, we use Lemma 5.2 to construct a sparse 2^k-cover $(H_1^k, H_2^k, \ldots, H_{m_k}^k)$ of H, for each $k \in \mathcal{I} = \{\lceil \log \frac{5}{\varepsilon} \rceil, \ldots, \lceil \log(\operatorname{diam}(H)) \rceil\}$. At this point we should remark, that $\operatorname{diam}(H)$ is not known in advance and a computation of it (approximate or exact) might take too much time. Therefore, it is not possible to use a `foreach`-loop. Instead we can use a `while`-loop in which k is incremented in every step and which terminates if the corresponding sparse 2^k-cover contains the graph H as cluster. Finally, it is $|\mathcal{I}| \in O(\log(\operatorname{diam} H)) = O(\log D)$. Let V_i^k be the vertex set of H_i^k. A short look at Lemma 5.2 gives the following observation.

Observation 6.13. *For a fixed $k \in \mathcal{I}$ we have $m_k \in O(n)$ and*

$$\sum_{i=1}^{m_k} |V_i^k| \in O(n).$$

Let G_i^k be the induced unit disk graph of V_i^k. Let $k_0 = \lceil \log \frac{5}{\varepsilon} \rceil$, for each $G_i^{k_0}$, we apply the preprocessing mechanism of the low diameter routing scheme from Lemma 6.3. For each $k \in \mathcal{I} \setminus \{k_0\}$, we apply to each G_i^k the preprocessing step of the routing scheme with additive stretch from Theorem 6.12. We use $\operatorname{lab}_{k,i}(\cdot)$ and $\operatorname{tab}_{k,i}(\cdot)$ to denote the resulting labels and tables for the graph G_i^k, for $k \in \mathcal{I}$.

Finally, we can describe how to obtain the labels and tables for our routing scheme. Let v be a vertex of $DG(V)$ and let $k \in \mathcal{I}$. Since $(H_1^k, H_2^k, \ldots, H_{m_k}^k)$ is a sparse 2^k-cover, there exists an index $i(v,k)$ such that $H_{i(v,k)}^k$ contains all vertices $w \in V$ with $d_H(v,w) \leqslant 2^k$. We store $\big(k, i(v,k), \operatorname{lab}_{k,i(v,k)}(v)\big)$ in $\operatorname{lab}(v)$. Moreover, for each i with $v \in V_i^k$ we store $\big(k, i, \operatorname{tab}_{k,i}(v)\big)$ in $\operatorname{tab}(v)$. The following lemma bounds the size of the labels and tables.

Lemma 6.14. *For every vertex $v \in V$, we have $|\operatorname{lab}(v)| \in O\left(\varepsilon^{-1} \log D \log^3 n / \log\log n\right)$ and $|\operatorname{tab}(v)| \in \varepsilon^{-O(\varepsilon^{-2})} \log^3 n (1 + \log D / \log\log n)$.*

Proof. The label of v contains a tuple for each index $k \in \mathcal{I}$. First of all, let us fix one entry $\big(k, i(v,k), \operatorname{lab}_{k,i(v,k)}(v)\big)$. Clearly, by Observation 6.13, the numbers k and $i(v,k)$ are in $O(n)$ and thus, $O(\log n)$ bits suffice for their representation.[1] Moreover, by Lemma 6.3 and Theorem 6.12, we know that each tuple consists of at most $O\left(\varepsilon^{-1} \log^3 n / \log\log n\right)$ bits. Finally, since $|\mathcal{I}| \in O(\log D)$, the first part of the claim follows.

By Lemma 5.2, the vertex v appears in $O(1)$ subgraphs G_i^k for each $k \in \mathcal{I}$. Hence, $\operatorname{tab}(v)$ contains $O(\log D)$ entries. Let us fix a tuple $\big(k, i, \operatorname{tab}_{k,i}(v)\big)$. Again, $O(\log n)$ bits suffice to

[1] In fact, it is not necessary to store the value k. Instead the tuples can be ordered by increasing k. However, this does not change the asymptotic label size.

represent k and i. If $k = k_0$, the table $\mathrm{tab}_{k_0,i}(v)$ comes from the low diameter routing scheme. Since $\mathrm{diam}(G_i^{k_0}) \in O(1/\varepsilon)$, Lemma 6.3 implies that $\mathrm{tab}_{k_0,i}(v)$ needs $\varepsilon^{-O(\varepsilon^{-2})} \log^3 n$ bits. For $k \in \mathcal{I} \setminus \{k_0\}$, we derive the table $\mathrm{tab}_{k,i}(v)$ from the additive stretch routing scheme, Theorem 6.12. Thus, the corresponding tuple takes in total $O(\varepsilon^{-1} \log^3 n / \log\log n)$ bits. In summary, there is a constant number of tuples requiring $\varepsilon^{-O(\varepsilon^{-2})} \log^3 n$ bits and $O(\log D)$ tuples that need $O(\varepsilon^{-1} \log^3 n / \log\log n)$ bits. The claim follows. □

Lemma 6.15. *We need* $O(\varepsilon^{-1} n^2 \log^2 n)$ *time to compute the labels and routing tables.*

Proof. First, the spanner H can be computed in $O(n \log n)$ time according to Lemma 5.1. Next, for each $k \in \mathcal{I}$ we compute the sparse 2^k-cover. Lemma 5.2 gives the running time of $O(n \log n)$ for a fixed k. Hence, all covers can be computed in $O(n \log n \log D)$ time because $|\mathcal{I}| \in O(\log D)$. Let G_i^k be a subgraph for a fixed $k \in \mathcal{I} \setminus \{k_0\}$ and a fixed $i \in \{1, \ldots, m_k\}$. We compute the labels and tables for G_i^k which is possible in time $O(\varepsilon^{-1} |V_i^k| \log^3 n)$, see Theorem 6.12. Next, for each $v \in V_i^k$, we store $(k, i, \mathrm{tab}_{k,i}(v))$ in $\mathrm{tab}(v)$. Moreover, due to Lemma 5.2, we can check in constant time, whether $i = i(v, k)$. If this is the case, we store $(k, i, \mathrm{lab}_{k,i}(v))$ in $\mathrm{lab}(v)$. In summary, the preprocessing of G_i^k needs $O(\varepsilon^{-1} |V_i^k| \log^3 n)$ in total. For a fixed k the preprocessing of all G_i^k needs $O(\varepsilon^{-1} n \log^3 n)$ time. Since $|\mathcal{I} \setminus \{k_0\}| \in O(\log D)$, the preprocessing for all $k \neq k_0$ needs $O(\varepsilon^{-1} n \log^3 n \log D)$ time in total. Finally, for $k = k_0$, every subgraph G_i^k has diameter at most $O(1/\varepsilon)$ and thus, the preprocessing time is $O(\varepsilon^{-1} n^2 \log^2 n)$ by following the same argumentation as before but using Lemma 6.3 instead of Theorem 6.12. The claim follows. □

The routing function. We next describe the routing function σ, see Figure 6.2 as well as Algorithm 6.2. Suppose we are given the target label $\mathrm{lab}(t)$, the source routing table $\mathrm{tab}(s)$, together with the header h. The routing function works as follows: We find the smallest number $k = k(s,t) \in \mathcal{I}$ such that the tuple $(k, i, \mathrm{lab}_{k,i}(t))$ is in $\mathrm{lab}(t)$ and the tuple $(k, i, \mathrm{tab}_{k,i}(s))$ is in $\mathrm{tab}(s)$. By construction, it must hold $i = i(t,k)$ and therefore, G_i^k contains each vertex v with $d_H(v,t) \leqslant 2^k$, by the first property of a sparse 2^k-cover. Moreover, we can derive the following observation:

Observation 6.16. *Let* s, t *be vertices of* G_i^k *with* $k = k(s,t)$. *Then we have* $d(s,t) \leqslant 48 \cdot 2^k$. *Moreover, if* $k > k_0$ *we have* $d(s,t) \geqslant \frac{1}{5} \cdot 2^k$.

Proof. By property (ii) of our sparse cover we get $d(s,t) \leqslant \mathrm{diam}(G_i^k) \leqslant \mathrm{diam}(H_i^k) \leqslant 48 \cdot 2^k$. This proves the first inequality.

Next, let $k > k_0$. The minimality of k and property (i) of our sparse cover show that $d_H(s,t) \geqslant 2^{k-1}$ Finally, since H is a 2.42-spanner of G we derive $d(s,t) \geqslant \frac{2^k}{5}$ and the claim follows. □

Once we have k and i, we can distinguish three cases.

If $k > k_0$, we ignore the header (it will be empty) and use the function σ_{add} of the additive stretch routing scheme to route within G_i^k. For this, we take the table $\mathrm{tab}_{k,i}(s)$ from $\mathrm{tab}(s)$ and the label $\mathrm{lab}_{k,i}(t)$ from $\mathrm{lab}(t)$ to compute the next port. The header remains empty. We use the computed port to route to the next vertex.

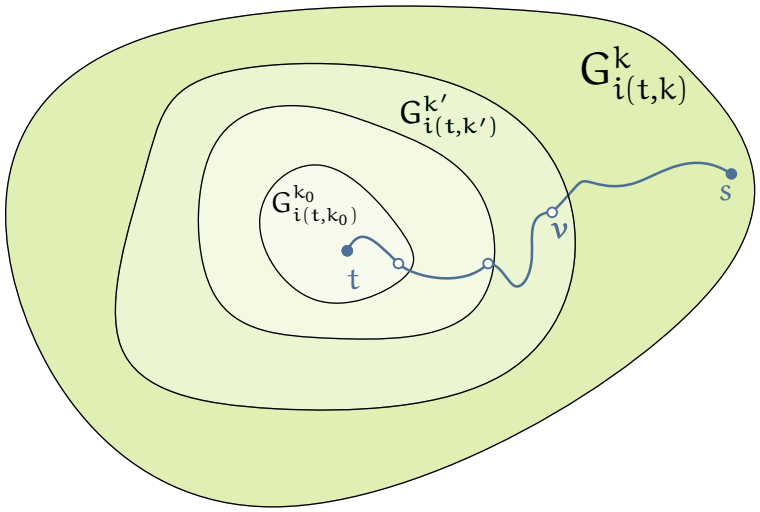


Figure 6.2: It holds $\frac{1}{5} \cdot 2^k \leqslant \operatorname{diam}\left(G^k_{i(t,k)}\right) \leqslant 48 \cdot 2^k$. We use the additive stretch routing scheme to route within $G^k_{i(t,k)}$ until we find a vertex v that is in $G^{k'}_{i(t,k')}$ for $k' < k$. This process continues until we find a vertex that is in $G^{k_0}_{i(t,k_0)}$, here we use the low diameter routing scheme until we reach t.

If $k = k_0$, we first check the header. If it is empty, we use the function σ_{diam} of the low diameter routing scheme to route within $G^{k_0}_i$. Again, we can take the label $\text{lab}_{k_0,i}(t)$ from $\text{lab}(t)$ and the table $\text{tab}_{k_0,i}(s)$ from $\text{tab}(s)$ to compute the next port. This time the routing function σ_{diam} also outputs a new string h of length $O(\log^2 n / \log\log n)$. Without loss of generality we assume that $h \neq \varepsilon$. We store h in the header and route the data packet along the computed port.

If $k = k_0$ and the header contains the non-empty string h, we use $\text{tab}_{k_0,i}(s)$, $\text{lab}_{k_0,i}(t)$ and h to route in $G^{k_0}_i$, while updating the header according to σ_{diam}.

Algorithm 6.2 The routing function for unit disk graphs with stretch $1+\varepsilon$.

```
1: procedure RoutingFunction(tab(s) ∈ Σ*, lab(t) ∈ Σ*, header h ∈ Σ*)
2:     Outputs: (port p ∈ Σ*, header h' ∈ Σ*)
3:     (k, i) = min{(k', i') | (k', i', lab_{k',i'}(t)) ∈ lab(t) and (k', i', tab_{k',i'}(s)) ∈ tab(s)}
4:     if k > ⌈log 5/ε⌉ then
5:         return (σ_add(tab_{k,i}(s), lab_{k,i}(t)), ε)
6:     else
7:         return σ_diam(tab_{k,i}(s), lab_{k,i}(t), h)
```

The stretch. Last but not least, we want to show the correctness and analyze the stretch factor. We start with the correctness. Its proof is quite similar to the correctness proof of σ_{add}.

Lemma 6.17. *The routing scheme is correct.*

Proof. Let s be the current vertex, t the desired target vertex and suppose that the routing scheme sends the packet from s to the vertex t. Moreover, let $k = k(s,t)$ and $i = i(t,k)$ be the indices that were used by the routing function to determine v. Since the routing step from s to v takes place in the graph G_i^k, we know that k is a potential candidate for $k(v,t)$. Thus, $k(v,t) \leqslant k$. If $k(v,t) < k$, we have made progress. However, if $k(v,t) = k$, the routing continues in the subgraph G_i^k, since for each k there is exactly one entry in $\mathrm{lab}(t)$. We already proved in Lemmas 6.3 and 6.10 that the underlying routing schemes for this task are correct. Hence, after a finite number of steps, we either reach t, or we decrease the value k. Since there is only a finite number of values for k, correctness follows. □

The next lemma bounds the additive stretch as a function of k.

Lemma 6.18. *Let s and t be two vertices and let $k = k(s,t)$. Then, we have*

$$\delta(s,t) \leqslant d(s,t) + 753\varepsilon \cdot 2^k.$$

Proof. We use induction on $k \geqslant k_0$. First, suppose that $k = k_0 = \lceil \log(5/\varepsilon) \rceil$ and let s,t be two vertices with $k(s,t) = k_0$. Let G_i^k be the graph that is used to determine the next vertex after s. Since k can only decrease while routing, and since k_0 is the minimum possible value of k, we route within G_i^k, using the low diameter routing scheme, until we reach t. Moreover, by Lemma 6.3 and Observation 6.16, we get

$$\delta(s,t) \leqslant (1+\varepsilon)d(s,t) \leqslant d(s,t) + 48\varepsilon \cdot 2^k.$$

Next, assume that $k > k_0$. Let s,t be two vertices with $k(s,t) = k$, and assume that for every vertex w with $k(w,t) < k$, we have $\delta(w,t) \leqslant d(w,t) + 753\varepsilon \cdot 2^{k(w,t)}$. Let G_i^k be the graph in which our scheme chooses to route the data packet from s to the next node. Let v be the first node on the routing path from s to t for which $k(v,t) < k$, see Figure 6.2. Moreover, let $\delta'(\cdot,\cdot)$ measure the length of the routing path within the subgraph G_i^k, using the additive stretch routing scheme. Next, by the definition of k_0 and since $k > k_0$ we get $\mathrm{diam}(G_i^k) \geqslant d(s,t) \geqslant \frac{1}{5} \cdot 2^k \geqslant 1/\varepsilon$ from Observation 6.16. Furthermore, we know that $d(v,t) \leqslant \delta'(v,t)$, since t is a vertex in G_i^k. Finally, we use the inductive hypothesis as well as Theorem 6.12 to derive

$$\begin{aligned}
\delta(s,t) &= \delta'(s,v) + \delta(v,t) \leqslant \delta'(s,v) + d(v,t) + 753\varepsilon \cdot 2^{k(v,t)} \\
&\leqslant \delta'(s,v) + \delta'(v,t) + 753\varepsilon \cdot 2^{k-1} = \delta'(s,t) + 753\varepsilon \cdot 2^{k-1} \\
&\leqslant d(s,t) + 7.84 \cdot \varepsilon \cdot 48 \cdot 2^k + 753\varepsilon \cdot 2^{k-1} \leqslant d(s,t) + 753\varepsilon \cdot 2^k,
\end{aligned}$$

Hence, the claim follows. □

Finally, we can put everything together to obtain our main theorem.

Theorem 6.19. *Let $\mathrm{DG}(V)$ be an n-vertex unit disk graph and D its diameter. Furthermore, let $\varepsilon > 0$. There is a routing scheme with $O\left(\varepsilon^{-1} \log D \log^3 n / \log\log n\right)$ label size, $\varepsilon^{-O(\varepsilon^{-2})} \log^3 n(1 + \log D/\log\log n)$ table size and $O(\log^2 n/\log\log n)$ header size whose routing function achieves the stretch factor $1+\varepsilon$. The preprocessing time is $O(\varepsilon^{-1} n^2 \log^2 n)$.*

Proof. It remains to show the stretch factor. Here, it suffices to show that the stretch factor is $1 + O(\varepsilon)$. Let s and t be two vertices and $k = k(s,t)$. If $k = k_0$ the stretch factor immediately follows from Lemma 6.3. Thus, assume $k \neq k_0$. On the one hand we know from Observation 6.16 that $\frac{1}{5}2^k \leqslant d(s,t)$, and on the other hand we know from Lemma 6.18 that $\delta(s,t) \leqslant d(s,t) + 753\varepsilon \cdot 2^k$. Putting everything together, we get the desired stretch as follows:

$$\delta(s,t) \leqslant d(s,t) + 753\varepsilon \cdot 2^k \leqslant d(s,t) + 753\varepsilon \cdot 5d(s,t) = (1 + 3765\varepsilon)d(s,t).$$

This finishes the proof. □

II

Stabbing

Part Outline

In this part of the thesis we focus our attention to sets $\mathcal{D}$ of pairwise intersecting disks. First, we present an upper bound on the stabbing number of $\mathcal{D}$ and a simple lower bound with 13 disks. Subsequently, we use the proof for the upper bound to present an algorithm that finds 5 stabbing points in linear time. Here, we use the well-known description of *LP-type problems* [SW92]. These problems and their properties will be discussed as well.

CHAPTER 7

Pairwise Intersecting Disks

In this chapter, we want to present some bounds on the stabbing number of pairwise intersecting disks in the Euclidean plane.

7.1 Upper Bound

Let $\mathcal{D}$ be a set of n pairwise intersecting disks in the plane. A disk $D_i \in \mathcal{D}$ is given by its center c_i and its radius r_i, that is $D_i = D(c_i, r_i)$. To simplify the analysis, we make the following assumptions: (i) the radii of the disks are pairwise distinct; (ii) the intersection of any two disks has a nonempty interior; and (iii) the intersection of any three disks is either empty or has a nonempty interior. A simple perturbation argument can then handle the degenerate cases.

The *lens* of two disks $D_i, D_j \in \mathcal{D}$ is the set $L_{i,j} = D_i \cap D_j$. Let u be any of the two intersection points of the boundary of D_i and the boundary of D_j. The angle $\angle c_i u c_j$ is called the *lens angle* of D_i and D_j. It is at most π. A finite set $\mathcal{C}$ of disks is *Helly* if their common intersection $\bigcap \mathcal{C}$ is nonempty. Otherwise, $\mathcal{C}$ is *non-Helly*. We present some useful geometric lemmas.

Lemma 7.1. *Let $\{D_1, D_2, D_3\}$ be a set of three pairwise intersecting disks that is non-Helly. Then, the set contains two disks with lens angle larger than $2\pi/3$.*

Proof. Since $\{D_1, D_2, D_3\}$ is non-Helly, the lenses $L_{1,2}$, $L_{1,3}$ and $L_{2,3}$ are pairwise disjoint. Let u be the vertex of $L_{1,2}$ nearer to D_3, and let v, w be the analogous vertices of $L_{1,3}$ and $L_{2,3}$ (see Figure 7.1, left). Consider the simple hexagon $c_1 u c_2 w c_3 v$, and write $\angle u$, $\angle v$, and $\angle w$ for its interior angles at u, v, and w. The sum of all interior angles is 4π. Thus, $\angle u + \angle v + \angle w < 4\pi$,

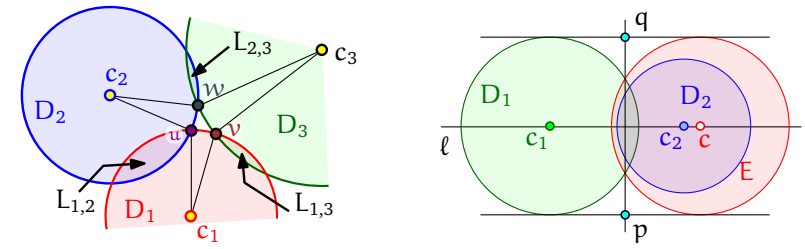


Figure 7.1: Left: At least one lens angle is large. Right: D_1 and E have the same radii and lens angle $2\pi/3$. By Lemma 7.2, D_2 is a subset of E. $\{c_1, c, p, q\}$ is the set P from Lemma 7.4.

so at least one angle is less than $4\pi/3$. It follows that the corresponding lens angle, which is the exterior angle at u, v, or w must be larger than $2\pi/3$. □

Lemma 7.2. *Let D_1 and D_2 be two intersecting disks with $r_1 \geqslant r_2$ and lens angle at least $2\pi/3$. Let E be the unique disk with radius r_1 and center c, such that*

(i) the centers c_1, c_2, and c are collinear and c lies on the same side of c_1 as c_2; and

(ii) the lens angle of D_1 and E is exactly $2\pi/3$ (see Figure 7.1, right).

Then, if c_2 lies between c_1 and c, we have $D_2 \subseteq E$.

Proof. Let $x \in D_2$. Since c_2 lies between c_1 and c, the triangle inequality gives

$$|xc| \leqslant |xc_2| + |c_2c| = |xc_2| + |c_1c| - |c_1c_2|. \tag{7.1}$$

Since $x \in D_2$, we get $|xc_2| \leqslant r_2$. Also, since D_1 and E have radius r_1 each and lens angle $2\pi/3$, it follows that $|c_1c| = \sqrt{3}\,r_1$. Finally, $|c_1c_2| = \sqrt{r_1^2 + r_2^2 - 2r_1r_2\cos\alpha}$, by the law of cosines, where α is the lens angle of D_1 and D_2. As $\alpha \geqslant 2\pi/3$ and $r_1 \geqslant r_2$, we get $\cos\alpha \leqslant -1/2 = (\sqrt{3} - 3/2) - \sqrt{3} + 1 \leqslant (\sqrt{3} - 3/2)r_1/r_2 - \sqrt{3} + 1$, As such, we have

$$\begin{aligned}
|c_1c_2|^2 = r_1^2 + r_2^2 - 2r_1r_2\cos\alpha &\geqslant r_1^2 + r_2^2 - 2r_1r_2\left((\sqrt{3} - 3/2)\frac{r_1}{r_2} - \sqrt{3} + 1\right) \\
&= r_1^2 - 2(\sqrt{3} - 3/2)r_1^2 + 2(-\sqrt{3} + 1)r_1r_2 + r_2^2 \\
&= (1 - 2\sqrt{3} + 3)r_1^2 + 2(-\sqrt{3} + 1)r_1r_2 + r_2^2 = \left(r_1(\sqrt{3} - 1) + r_2\right)^2.
\end{aligned}$$

Plugging this into Equation (7.1) gives $|xc| \leqslant r_2 + \sqrt{3}r_1 - (r_1(\sqrt{3} - 1) + r_2) = r_1$, i.e., $x \in E$. □

Lemma 7.3. *Let D_1 and D_2 be two intersecting disks with equal radius r and lens angle $2\pi/3$. There is a set P of four points so that any disk F of radius at least r that intersects both D_1 and D_2 contains a point of P.*

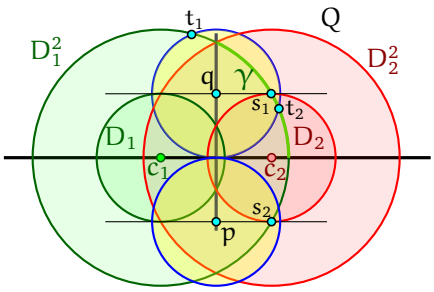


Figure 7.2: Left: $P = \{c_1, c_2, p, q\}$ is the stabbing set. The green arc $\gamma = \partial D_1^2 \cap Q$ is covered by $D_2 \cup D_q$.

Proof. Consider the two tangent lines of D_1 and D_2, and let p and q be the midpoints on these lines between the respective two tangency points. We set $P = \{c_1, c_2, p, q\}$; see Figure 7.2.

Given the disk F that intersects both D_1 and D_2, we shrink its radius, keeping its center fixed, until either the radius becomes r or until F is tangent to D_1 or D_2. Suppose the latter case holds and F is tangent to D_1. We move the center of F continuously along the line spanned by the center of F and c_1 towards c_1, decreasing the radius of F to maintain the tangency. We stop when either the radius of F reaches r or F becomes tangent to D_2. We obtain a disk $G \subseteq F$ with center $c = (c_x, c_y)$ so that either: (i) radius$(G) = r$ and G intersects both D_1 and D_2; or (ii) radius$(G) \geqslant r$ and G is tangent to both D_1 and D_2. Since $G \subseteq F$, it suffices to show that $G \cap P \neq \emptyset$.

We introduce a coordinate system, setting the origin o midway between c_1 and c_2, so that the y-axis passes through p and q. Then, as in Figure 7.2, we have $c_1 = (-\sqrt{3}\,r/2, 0)$, $c_2 = (\sqrt{3}\,r/2, 0)$, $q = (0, r)$, and $p = (0, -r)$.

For case (i), let D_1^2 be the disk of radius $2r$ centered at c_1, and D_2^2 the disk of radius $2r$ centered at c_2. Since G has radius r and intersects both D_1 and D_2, its center c has distance at most $2r$ from both c_1 and c_2, i.e., $c \in D_1^2 \cap D_2^2$. Let D_p and D_q be the two disks of radius r centered at p and q. We will show that $D_1^2 \cap D_2^2 \subseteq D_1 \cup D_2 \cup D_p \cup D_q$. Then it is immediate that $G \cap P \neq \emptyset$. By symmetry, it is enough to focus on the upper-right quadrant $Q = \{(x, y) \mid x \geqslant 0, y \geqslant 0\}$. We show that all points in $D_1^2 \cap Q$ are covered by $D_2 \cup D_q$. Without loss of generality, we assume that $r = 1$. Then, the two intersection points of D_1^2 and D_q are $t_1 = (\frac{5\sqrt{3}-2\sqrt{87}}{28}, \frac{38+3\sqrt{29}}{28}) \approx (-0.36, 1.93)$ and $t_2 = (\frac{5\sqrt{3}+2\sqrt{87}}{28}, \frac{38-3\sqrt{29}}{28}) \approx (0.98, 0.78)$, and the two intersection points of D_1^2 and D_2 are $s_1 = (\frac{\sqrt{3}}{2}, 1) \approx (0.87, 1)$ and $s_2 = (\frac{\sqrt{3}}{2}, -1) \approx (0.87, -1)$. Let γ be the boundary curve of D_1^2 in Q. Since $t_1, s_2 \notin Q$ and since $t_2 \in D_2$ and $s_1 \in D_q$, it follows that γ does not intersect the boundary of $D_2 \cup D_q$ and hence $\gamma \subset D_2 \cup D_q$. Furthermore, the subsegment of the y-axis from o to the start point of γ is contained in D_q, and the subsegment of the x-axis from o to the endpoint of γ is contained in D_2. Hence, the boundary of $D_1^2 \cap Q$ lies completely in $D_2 \cup D_q$, and since $D_2 \cup D_q$ is simply connected, it follows that $D_1^2 \cap Q \subseteq D_2 \cup D_q$, as desired.

For case (ii), since G is tangent to D_1 and D_2, the center c of G is on the perpendicular

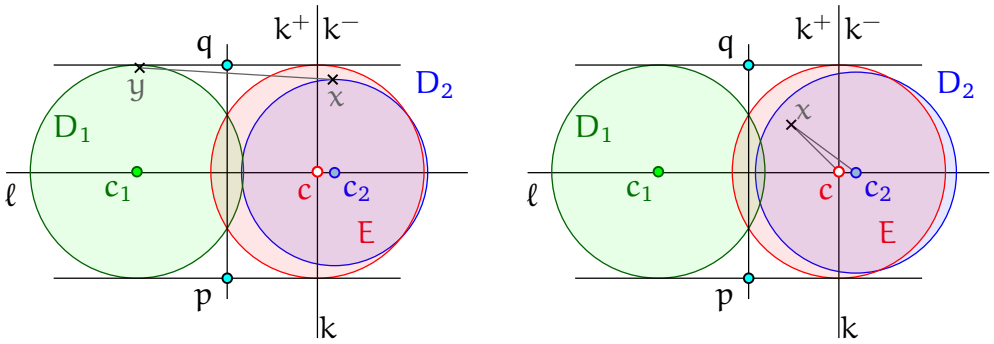


Figure 7.3: Proof of Lemma 7.4. Left (Case (i)): x is an arbitrary point in $D_2 \cap F \setminus k^+$ and y is an arbitrary point in $D_1 \cap F$. Right (Case (ii)): x is an arbitrary point in $D_2 \cap F \cap k^+$. The angle at c in the triangle $\Delta x c c_2$ is $\geqslant \pi/2$.

bisector of c_1 and c_2, so the points p, o, q and c are collinear. Suppose without loss of generality that $c_y \geqslant 0$. Then, it is easily checked that c lies above q, and $\text{radius}(G) + r = |c_1 c| \geqslant |oc| = r + |qc|$, so $q \in G$. □

Lemma 7.4. *Consider two intersecting disks D_1 and D_2 with $r_1 \geqslant r_2$ and lens angle at least $2\pi/3$. Then, there is a set P of four points such that any disk F of radius at least r_1 that intersects both D_1 and D_2 contains a point of P.*

Proof. Let ℓ be the line through c_1 and c_2. Let E be the disk of radius r_1 and center $c \in \ell$ that satisfies the conditions (i) and (ii) of Lemma 7.2. Let $P = \{c_1, c, p, q\}$ as in the proof of Lemma 7.3, with respect to D_1 and E (see Figure 7.1, right). We claim that

$$D_1 \cap F \neq \emptyset \ \wedge \ D_2 \cap F \neq \emptyset \ \Rightarrow \ E \cap F \neq \emptyset. \qquad (*)$$

Once (*) is established, we are done by Lemma 7.3. If $D_2 \subseteq E$, then (*) is immediate, so assume that $D_2 \not\subseteq E$. By Lemma 7.2, c lies between c_1 and c_2. Let k be the line through c perpendicular to ℓ, and let k^+ be the open halfplane bounded by k with $c_1 \in k^+$ and k^- the open halfplane bounded by k with $c_1 \notin k^-$. Since $|c_1 c| = \sqrt{3}\, r_1 > r_1$, we have $D_1 \subset k^+$; see Figure 7.3. Recall that F has radius at least r_1 and intersects D_1 and D_2. We distinguish two cases: (i) there is no intersection of F and D_2 in k^+, and (ii) there is an intersection of F and D_2 in k^+; see Figure 7.3 for the two cases.

For case (i), let x be any point in $D_1 \cap F$. Since we know that $D_1 \subset k^+$, we have $x \in k^+$. Moreover, let y be any point in $D_2 \cap F$. By assumption, y is not in k^+, but it must be in the infinite strip defined by the two tangents of D_1 and E. Thus, the line segment $\overline{xy}$ intersects the diameter segment $k \cap E$. Since F is convex, the intersection of $\overline{xy}$ and $k \cap E$ is in F, so $E \cap F \neq \emptyset$.

For case (ii), fix $x \in D_2 \cap F \cap k^+$ arbitrarily. Consider the triangle $\Delta x c c_2$. Since $x \in k^+$, the angle at c is at least $\pi/2$. Thus, $|xc| \leqslant |xc_2|$. Also, since $x \in D_2$, we know that $|xc_2| \leqslant r_2 \leqslant r_1$. Hence, $|xc| \leqslant r_1$, so $x \in E$ and (*) follows, as $x \in E \cap F$. □

With these tools we can now show that there is a stabbing set with five points.

Theorem 7.5. *Let $\mathcal{D}$ be a set of n pairwise intersecting disks in the plane. There is a set P of five points such that each disk in $\mathcal{D}$ contains at least one point from P.*

Proof. If $\mathcal{D}$ is Helly, there is a single point that lies in all disks of $\mathcal{D}$. Thus, assume that $\mathcal{D}$ is non-Helly, and let $D_1, D_2, \ldots, D_n$ be the disks in $\mathcal{D}$ ordered by increasing radius. Let i^* be the smallest index with $\bigcap_{i \leqslant i^*} D_i = \emptyset$. By Helly's theorem [Hel23, Hel30, Rad21], there are indices $j, k < i^*$ such that $\{D_{i^*}, D_j, D_k\}$ is non-Helly. By Lemma 7.1, two disks in $\{D_{i^*}, D_j, D_k\}$ have lens angle at least $2\pi/3$. Applying Lemma 7.4 to these two disks, we obtain a set P' of four points so that every disk D_i with $i \geqslant i^*$ contains at least one point from P'. Furthermore, by definition of i^*, we have $\bigcap_{i<i^*} D_i \neq \emptyset$, so there is a point q that stabs every disk D_i with $i < i^*$. Thus, $P = P' \cup \{q\}$ is a set of five points that stabs every disk in $\mathcal{D}$, as desired. □

Remark. A weakness in our proof is that it combines two different stages, one of finding the point q that stabs all the small disks, and one of constructing the four points of Lemma 7.4 that stab all the larger disks. It is an intriguing challenge to merge the two arguments so that altogether they only require four points. The proof of Carmi et al. [CKM18] uses a different approach.

7.2 Simple Bounds

In this section, we provide some lower and upper bounds on the number of disks for which a certain number of stabbing points is necessary or sufficient.

Eight disks can be stabbed py three points. For the proof that any set of eight pairwise intersecting disks can be stabbed by at most three points, we show the following lemma.

Lemma 7.6. *Let $\mathcal{D}$ be a set of at least 5 pairwise intersecting disks. Then, $\mathcal{D}$ contains a Helly-triple.*

Proof. Let $\mathcal{D}$ be a set of exactly 5 pairwise intersecting disks. We assume that no three centers of the disks are on a line, since otherwise these three disks are a Helly-triple. Since the complete graph K_5 does not have a planar embedding, there have to be four different disks $D_1, \ldots, D_4 \in \mathcal{D}$ with centers $c_1, \ldots, c_4$ and radii $r_1, \ldots, r_4$ such that the line segments c_1c_3 and c_2c_4 intersect, see Figure 7.4. Let x be the intersection point. Moreover, let α (resp., β) be the intersection of the lens $L_{1,3}$ (resp., $L_{2,4}$) and the line segment c_1c_3 (resp., c_2c_4). If x is in α or β, we are done. Otherwise, let y be the point of α that is closest to x and let z be the point of β closest to x. We can assume without loss of generality that $|xy| \leqslant |xz|$ and $x \notin D_4$. We can derive $|c_2y| \leqslant |c_2x| + |xy| \leqslant |c_2x| + |c_2z| \leqslant r_2$ to conclude that $y \in D_1 \cap D_2 \cap D_3$. □

Now consider a set of 8 pairwise intersecting disks. Using the previous lemma, we can find a Helly-triple. Among the other 5 disks we find a second Helly-triple. The remaining two

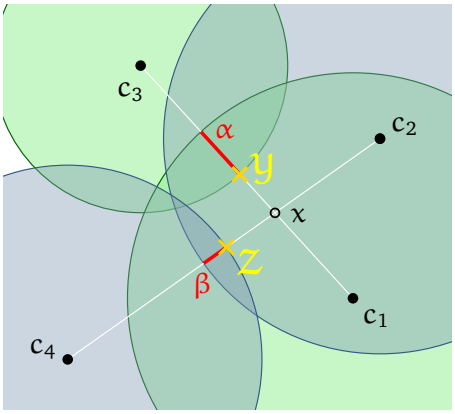


Figure 7.4: Proof of Lemma 7.6.

disks can be stabbed by one point. This reasoning yields the following corollary, which was already mentioned by Stachó [Sta65].

Corollary 7.7. *Every set $\mathcal{D}$ of at most 8 pairwise intersecting disks can be stabbed by 3 points.*

Danzer presented a set of 10 pairwise intersecting pseudo-disks with stabbing number four [Dan86]. However, it is not clear how these 10 pseudo-disks can be realized as pairwise Euclidean Disks achieving the same stabbing number. Moreover, it is another open problem whether 9 pairwise intsersecting disks can be stabbed by three points. Instead, we want to describe a set of 13 pairwise intersecting disks in the plane such that no point set of size three can pierce all of them.

13 disks with 4 stabbing points. The construction begins with an inner disk A of radius 1 and three larger disks D_1, D_2, D_3 of equal radius, so that each pair of disks in $\{A, D_1, D_2, D_3\}$ is tangent. For $i = 1, 2, 3$, we denote the contact point of A and D_i by ξ_i.

We add six more disks as follows. For $i = 1, 2, 3$, we draw the two common outer tangents to A and D_i, and denote by T_i^- and T_i^+ the halfplanes that are bounded by these tangents and are openly disjoint from A. The labels T_i^- and T_i^+ are chosen such that the points of tangency between A and T_i^-, D_i, and T_i^+, appear along the boundary of A in this counterclockwise order. One can show that the nine points of tangency between A and the other disks and tangents are pairwise distinct (see Figure 7.5). We regard the six halfplanes T_i^-, T_i^+, for $i = 1, 2, 3$, as (very large) disks; in the end, we can apply a suitable inversion to turn the disks and halfplanes into actual disks, if so desired.

Finally, we construct three additional disks A_1, A_2, A_3. To construct A_i, we slightly expand A into a disk A_i' of radius $1 + \varepsilon_1$, while keeping the tangency with D_i at ξ_i. We then roll A_i' clockwise along D_i, by a tiny angle $\varepsilon_2 \ll \varepsilon_1$, to obtain A_i.

This gives a set of 13 disks. For sufficiently small ε_1 and ε_2, we can ensure the following properties for each A_i: (i) A_i intersects all other 12 disks; (ii) the nine intersection regions $A_i \cap D_j$, $A_i \cap T_j^-$, $A_i \cap T_j^+$, for $j = 1, 2, 3$, are pairwise disjoint; and (iii) $\xi_i \notin A_i$.

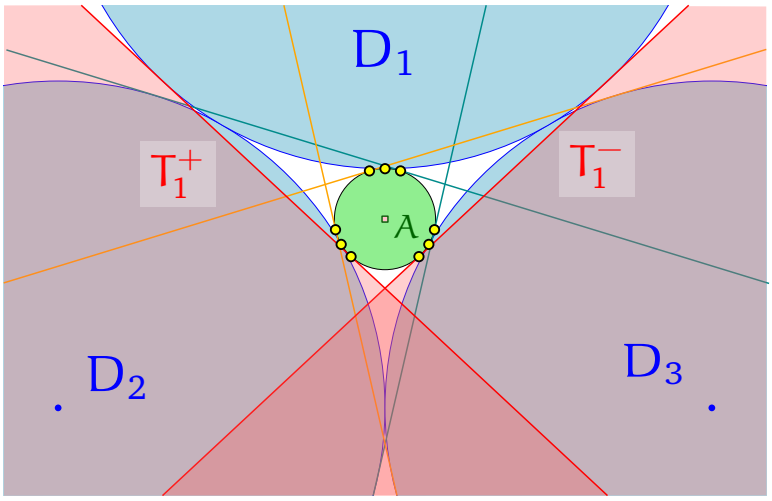


Figure 7.5: Each common tangent ℓ between A and D_i represents a very large disk, whose interior is disjoint from A. The nine points of tangency are pairwise distinct.

Theorem 7.8. *The construction yields a set of* 13 *disks that cannot be stabbed by* 3 *points.*

Proof. Consider any set P of three points. Set $A^* = A \cup A_1 \cup A_2 \cup A_3$. If $P \cap A^* = \emptyset$, we have unstabbed disks, so suppose that $P \cap A^* \neq \emptyset$. For $p \in P \cap A^*$, property (ii) implies that p stabs at most one of the nine remaining disks D_j, T_j^+ and T_j^-, for $j = 1, 2, 3$. Thus, if $P \subset A^*$, we would have unstabbed disks, so we may assume that $|P \cap A^*| \in \{1, 2\}$.

Suppose first that $|P \cap A^*| = 2$. As just argued, at most two of the remaining disks are stabbed by $P \cap A^*$. The following cases can then arise.

(a) None of D_1, D_2, D_3 is stabbed by $P \cap A^*$. Since $\{D_1, D_2, D_3\}$ is non-Helly and a non-Helly set must be stabbed by at least two points, at least one disk remains unstabbed.

(b) Two disks among D_1, D_2, D_3 are stabbed by $P \cap A^*$. Then the six unstabbed halfplanes form many non-Helly triples, e.g., T_1^-, T_2^-, and T_3^-, and again, a disk remains unstabbed.

(c) The set $P \cap A^*$ stabs one disk in $\{D_1, D_2, D_3\}$ and one halfplane. Then, there is (at least) one disk D_i such that D_i and its two tangent halfplanes T_i^-, T_i^+ are all unstabbed by $P \cap A^*$. Then, $\{D_i, T_i^-, T_i^+\}$ is non-Helly, and at least 2 more points are needed to stab it.

Suppose now that $|P \cap A^*| = 1$, and let $P \cap A^* = \{p\}$. We may assume that p stabs all four disks A, A_1, A_2, A_3, since otherwise a disk would stay unstabbed. By property (iii), we can derive $p \notin \{\xi_1, \xi_2, \xi_3\}$. Now, since $p \in A \setminus \{\xi_1, \xi_2, \xi_3\}$, the point p does not stab any of D_1, D_2, D_3. Moreover, by property (ii), the point p can only stab at most one of the remaining halfplanes. Since $\{D_1, D_2, D_3\}$ is non-Helly, it requires two stabbing points. Moreover, since $|P \setminus \{p\}| = 2$, it must be the case that one point q of $P \setminus A^*$ is the point of tangency of two of these disks, say $q = D_2 \cap D_3$. Then, q stabs only two of the six halfplanes, say, T_1^- and T_1^+. But then, $\{D_1, T_2^+, T_3^-\}$ is non-Helly and does not contain any point from $\{p, q\}$. At least one disk remains unstabbed. □

CHAPTER 8

Computation of five Stabbing Points

In the last chapter we provided a proof that the stabbing number of a set of pairwise intersecting disks is at most 5. In this chapter, we want to use this property to present a linear-time algorithm for finding the 5 stabbing points. However, we first present a simple near-linear time algorithm that already shows some properties about stabbing disks.

8.1 A simple near-linear time algorithm

The proof of Theorem 7.5 leads to a simple $O(n \log n)$ time deterministic algorithm for finding a stabbing set of size five. For this, we need an oracle that decides whether a given set of disks is Helly. This has already been done by Löffler and van Kreveld [LvK10], in a more general context:

Lemma 8.1 (Theorem 6 in [LvK10])**.** *Given a set of n disks, the problem of choosing a point in each disk such that the smallest enclosing circle of the resulting point set is as small as possible can be solved in $O(n)$ time.*

Now, an $O(n \log n)$-time algorithm for finding the five stabbing points is based on the analysis in the proof of Theorem 7.5. It works as follows: first, we sort the disks in $\mathcal{D}$ by increasing radius, this costs $O(n \log n)$ time. Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be the resulting order. Next, we use binary search with the oracle from Lemma 8.1 to determine the smallest index i^* such that $\{D_1, \ldots, D_{i^*}\}$ is non-Helly. This yields the disk D_{i^*}. Here, we have to use the oracle $O(\log n)$ times, which gives a total of $O(n \log n)$ for this step. After that, we use another binary search with the oracle from Lemma 8.1 to determine the smallest index $k < i^*$ such that $\{D_{i^*}, D_1, \ldots, D_k\}$ is non-Helly. This costs $O(n \log n)$ time as well. Then, we perform a

linear search to find an index $j < k$ such that $\{D_j, D_k, D_{i^*}\}$ is a non-Helly triple. This step works in linear time. Finally, we use Lemma 8.1 to obtain in linear time a stabbing point q for the Helly set $\{D_1, \ldots, D_{i^*-1}\}$ and the method from the proof of Theorem 7.5 to extend q to a stabbing set for the whole set $\mathcal{D}$. This last step works in $O(1)$ time since the result depends solely on $\{D_j, D_k, D_{i^*}\}$. Hence, we can present our claimed theorem.

Theorem 8.2. *Given a set $\mathcal{D}$ of n pairwise intersecting disks in the plane, we can find in $O(n \log n)$ time a set P of five points such that every disk of $\mathcal{D}$ contains at least one point of P.*

8.2 A linear time algorithm

The proof of Lemma 8.1 uses the *LP-type framework* by Sharir and Welzl [Cha01, SW92]. As we will see next, a more sophisticated application of the framework directly leads to a deterministic linear time algorithm to find a stabbing set with five points.

The LP-type framework. An *LP-type problem* $(\mathcal{H}, w, \leqslant)$ is an abstract generalization of a low-dimensional linear program. It consists of a finite set of *constraints* $\mathcal{H}$, a *weight function* $w : 2^{\mathcal{H}} \to \mathcal{W}$, and a *total order* $(\mathcal{W}, \leqslant)$ on the weights. The weight function w assigns a weight to each subset of constraints. It must fulfill the following two axioms:

- **Monotonicity**: for any $\mathcal{H}' \subseteq \mathcal{H}$ and $H \in \mathcal{H}$, we have $w\big(\mathcal{H}' \cup \{H\}\big) \leqslant w(\mathcal{H}')$;
- **Locality:** for any $\mathcal{B} \subseteq \mathcal{H}' \subseteq \mathcal{H}$ with $w(\mathcal{B}) = w(\mathcal{H}')$ and for any $H \in \mathcal{H}$, we have that if $w\big(\mathcal{B} \cup \{H\}\big) = w(\mathcal{B})$, then also $w\big(\mathcal{H}' \cup \{H\}\big) = w(\mathcal{H}')$.

Given a subset $\mathcal{H}' \subseteq \mathcal{H}$, a *basis* for $\mathcal{H}'$ is an inclusion-minimal set $\mathcal{B} \subseteq \mathcal{H}'$ with $w(\mathcal{B}) = w(\mathcal{H}')$. The *combinatorial dimension* of $(\mathcal{H}, w, \leqslant)$ is the maximum size of any basis of any subset of $\mathcal{H}$. The goal in an LP-type problem is to determine $w(\mathcal{H})$ and a corresponding basis $\mathcal{B}$ for $\mathcal{H}$. Next, given a set $\mathcal{B} \subseteq \mathcal{H}$ and a constraint $H \in \mathcal{H}$, we say that H *violates* $\mathcal{B}$ if $w\big(\mathcal{B} \cup \{H\}\big) < w(\mathcal{B})$.

A generalization of Seidel's algorithm for low-dimensional linear programming [Sei91, SW92] shows that we can solve an LP-type problem in $O(|\mathcal{H}|)$ expected time, provided that a constant time algorithm for the following problem is available. Here and below, the constant factor in the O-notation may depend on the combinatorial dimension.

- **Violation test:** Given a basis $\mathcal{B}$ and a constraint $H \in \mathcal{H}$, determine whether H violates $\mathcal{B}$ and return an error message if $\mathcal{B}$ is not a basis for any $\mathcal{H}' \subseteq \mathcal{H}$.[1]

For a deterministic solution, we need an additional computational assumption. Let $\mathcal{B} \subseteq \mathcal{H}$ be a basis of any subset $\mathcal{H}' \subseteq \mathcal{H}$, we use $\mathrm{vio}(\mathcal{B})$ to denote the set of all constraints $H \in \mathcal{H}$ that

[1] Here, we follow the presentation of Chazelle and Matoušek [CM96]. Sharir and Welzl [SW92] use a violation test without the error message. Instead, they need an additional *basis computation* primitive: given a basis $\mathcal{B}$ and a constraint $H \in \mathcal{H}$, find a basis for $\mathcal{B} \cup \{H\}$. If a violation test with error message exists and if the combinatorial dimension is a constant, a basis computation primitive can easily be implemented by brute-force enumeration.

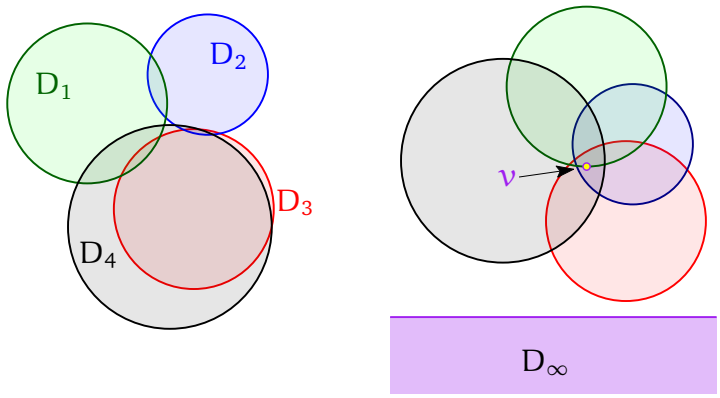


Figure 8.1: Left: The disks D_3 and D_4 are destroyers of the Helly set $\{D_1, D_2\}$. Moreover, D_3 is the smallest destroyer of the whole set $\{D_1, D_2, D_3, D_4\}$. Right: The disks without D_∞ form a Helly set $\mathcal{C}$. The smallest destroyer of $\mathcal{C}$ is D_∞ and the point v is the extreme point for $\mathcal{C}$ and D_∞, i.e., $\text{dist}(\mathcal{C}) = d(v, D_\infty)$.

violate $\mathcal{B}$, i.e., that have $w(\mathcal{B} \cup \{H\}) < w(\mathcal{B})$. Consider the range space $(\mathcal{H}, \mathcal{R} = \{\text{vio}(\mathcal{B}) \mid \mathcal{B} \text{ is a basis for some } \mathcal{H}' \subseteq \mathcal{H}\})$ and let δ be its shattering dimension. Chazelle and Matoušek [CM96] have shown that an LP-type problem can be solved in $O(|\mathcal{H}|)$ *deterministic* time if there is a constant-time violation test as stated above, δ is a constant, and the following computational assumption holds:

- **Oracle:** Given a subset $\mathcal{Y} \subseteq \mathcal{H}$, we can compute some superset $\mathcal{R}' \supseteq \mathcal{R}_\mathcal{Y}$ in time $|\mathcal{Y}|^{\delta+1}$, where $(\mathcal{Y}, \mathcal{R}_\mathcal{Y})$ is the range space induced by $\mathcal{Y}$.

During the following discussion, we will show that the problem of finding a non-Helly triple as in Theorem 7.5 is LP-type and fulfills the requirements for the algorithm of Chazelle and Matoušek .

Remark. Löffler and van Kreveld provide proofs that the underlying problem in Lemma 8.1 is of LP-type, but they do not give arguments for the two computational assumptions, see [LvK10]. However, it is not difficult to also verify the two missing statements.

Geometric observations. The *distance* between two closed sets $A, B \subseteq \mathbb{R}^2$ is defined as $d(A, B) = \inf\{|ab| \mid a \in A, b \in B\}$. From now on, we assume that all points in $\bigcup \mathcal{D}$ have positive y-coordinates. This can be ensured with linear overhead by an appropriate translation of the input. We denote by D_∞ the closed halfplane below the x-axis. It is interpreted as a disk with radius ∞ and center at $(0, -\infty)$. First, observe that for any subsets $\mathcal{C}_1 \subseteq \mathcal{C}_2 \subseteq \mathcal{D} \cup \{D_\infty\}$, we have that if $\mathcal{C}_1$ is non-Helly, then $\mathcal{C}_2$ is non-Helly. For any $\mathcal{C} \subseteq \mathcal{D} \cup \{D_\infty\}$, we say that a disk D *destroys* $\mathcal{C}$ if $\mathcal{C} \cup \{D\}$ is non-Helly. Observe that D_∞ destroys every non-empty subset of $\mathcal{D}$. Moreover, if $\mathcal{C}$ is non-Helly, then every disk is a destroyer. See Figure 8.1 for an example. We can make the following two observations.

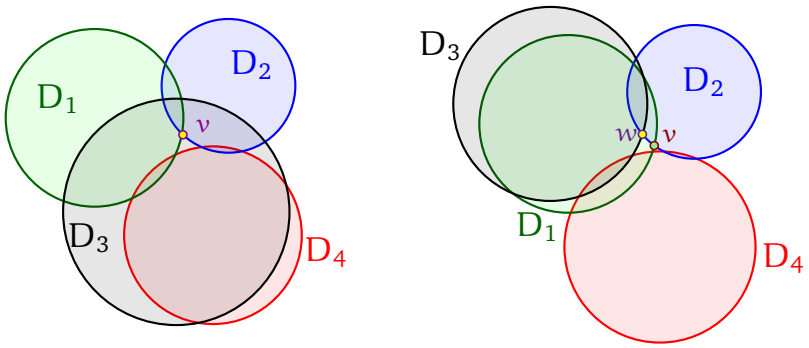


Figure 8.2: Left: The disk D_4 is a destroyer for the Helly sets $\{D_1, D_2\}$ and $\{D_1, D_2, D_3\}$. The extreme point v for $\{D_1, D_2\}$ is also the extreme point for $\{D_1, D_2, D_3\}$. Right: The disk D_4 is a destroyer for the Helly sets $\{D_1, D_2\}$ and $\{D_1, D_2, D_3\}$. The extreme point v for $\{D_1, D_2\}$ is not in D_3. The distance to D_4 increases.

Lemma 8.3. *Let $\mathcal{C} \subseteq \mathcal{D}$ be Helly and D a destroyer of $\mathcal{C}$. Then, the point $v \in \bigcap \mathcal{C}$ with minimum distance to D is unique.*

Proof. Suppose there are two distinct points $v \neq w \in \bigcap \mathcal{C}$ with $d(v, D) = d\big(\bigcap \mathcal{C}, D\big) = d(w, D)$. Since $\bigcap \mathcal{C}$ is convex, the segment $\overline{vw}$ lies in $\bigcap \mathcal{C}$. Now, if $D \neq D_\infty$, then every point in the relative interior of $\overline{vw}$ is strictly closer to D than v and w. If $D = D_\infty$, then all points in $\overline{vw}$ have the same distance to D, but since $\bigcap \mathcal{C}$ is strictly convex, the relative interior of $\overline{vw}$ lies in the interior of $\bigcap \mathcal{C}$, so there must be a point in $\bigcap \mathcal{C}$ that is closer to D than v and w. In either case, we obtain a contradiction to the assumption $v \neq w$ and $d(v, D) = d\big(\bigcap \mathcal{C}, D\big) = d(w, D)$. The claim follows. □

Let $\mathcal{C} \subseteq \mathcal{D}$ be Helly and D a destroyer of $\mathcal{C}$. The unique point $v \in \bigcap \mathcal{C}$ with minimum distance to D is called the *extreme point* for $\mathcal{C}$ and D (see Figure 8.1, right).

Lemma 8.4. *Let $\mathcal{C}_1 \subseteq \mathcal{C}_2 \subseteq \mathcal{D}$ be two Helly sets and D a destroyer of $\mathcal{C}_1$ (and thus of $\mathcal{C}_2$). Let $v \in \bigcap \mathcal{C}_1$ be the extreme point for $\mathcal{C}_1$ and D. We have $d\big(\bigcap \mathcal{C}_1, D\big) \leqslant d\big(\bigcap \mathcal{C}_2, D\big)$. In particular, if $v \in \bigcap \mathcal{C}_2$, then $d\big(\bigcap \mathcal{C}_1, D\big) = d\big(\bigcap \mathcal{C}_2, D\big)$ and v is also the extreme point for $\mathcal{C}_2$ and D. If $v \notin \bigcap \mathcal{C}_2$, then $d\big(\bigcap \mathcal{C}_1, D\big) < d\big(\bigcap \mathcal{C}_2, D\big)$.*

Proof. The first claim holds trivially: let $w \in \bigcap \mathcal{C}_2$ be the extreme point for $\mathcal{C}_2$ and D. Since $\mathcal{C}_1 \subseteq \mathcal{C}_2$, it follows that $w \in \bigcap \mathcal{C}_1$, so $d\big(\bigcap \mathcal{C}_1, D\big) \leqslant d(w, D) = d\big(\bigcap \mathcal{C}_2, D\big)$. If $v \in \bigcap \mathcal{C}_2$, then $d\big(\bigcap \mathcal{C}_1, D\big) \leqslant d\big(\bigcap \mathcal{C}_2, D\big) \leqslant d(v, D) = d\big(\bigcap \mathcal{C}_1, D\big)$, so $v = w$, by Lemma 8.3. If $v \notin \bigcap \mathcal{C}_2$, then $d\big(\bigcap \mathcal{C}_1, D\big) < d\big(\bigcap \mathcal{C}_2, D\big)$, by Lemma 8.3 and the fact that $\mathcal{C}_1 \subseteq \mathcal{C}_2$. See Figure 8.2. □

Let $\mathcal{C}$ be a subset of $\mathcal{D}$. For $0 < r \leqslant \infty$ we define $\mathcal{C}_{<r}$ as the set of all disks in $\mathcal{C}$ with radius smaller than r. Recall that we assume that all the radii are pairwise distinct. A disk D with radius r, $0 < r \leqslant \infty$, is called *smallest destroyer* of $\mathcal{C}$ if (i) $D \in \mathcal{C}$ or $D = D_\infty$, (ii) D destroys

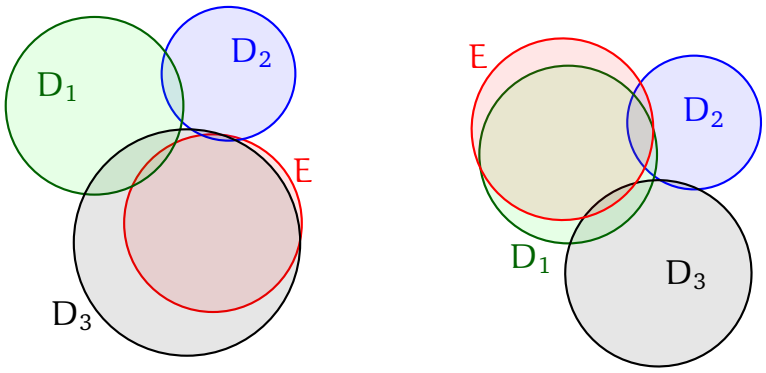


Figure 8.3: Monotonicity: In both cases, $\{D_1, D_2, D_3\}$ is non-Helly with smallest destroyer D_3. Adding a disk E either decreases the radius of the smallest destroyer (left) or increases the distance to the smallest destroyer (right).

$\mathcal{C}_{<r}$, and (iii) there is no disk $D' \in \mathcal{C}_{<r}$ that destroys $\mathcal{C}_{<r}$. Observe that Property (iii) is the same as saying that $\mathcal{C}_{<r}$ is Helly. See Figure 8.1 for an example.

Let $\mathcal{C}$ be a subset of $\mathcal{D}$ and D the smallest destroyer of $\mathcal{C}$. We write $\mathrm{rad}(\mathcal{C})$ for the radius of D and $\mathrm{dist}(\mathcal{C})$ for the distance between D and the set $\bigcap \mathcal{C}_{<\mathrm{rad}(\mathcal{C})}$, i.e., $\mathrm{dist}(\mathcal{C}) = d\big(\bigcap \mathcal{C}_{<\mathrm{rad}(\mathcal{C})}, D\big)$. Now, if $\mathcal{C}$ is Helly, then $D = D_\infty$ and thus $\mathrm{rad}(\mathcal{C}) = \infty$. If $\mathcal{C}$ is non-Helly, then $D \in \mathcal{C}$ and thus $\mathrm{rad}(\mathcal{C}) < \infty$. In both cases, $\mathrm{dist}(\mathcal{C})$ is the distance between D and the extreme point for $\mathcal{C}_{<\mathrm{rad}(\mathcal{C})}$ and D. We define the *weight of* $\mathcal{C}$ as $w(\mathcal{C}) = (\mathrm{rad}(\mathcal{C}), -\mathrm{dist}(\mathcal{C}))$, and we denote by $\leqslant$ the lexicographic order on $\mathbb{R}^2$. Chan observed, in a slightly different context, that $(\mathcal{D}, w, \leqslant)$ is LP-type [Cha04]. However, Chan's paper does not contain a detailed proof for this fact. Thus, in the following lemmas, we show the two LP-type axioms, present a constant time violation test, and a polynomial-time oracle. We start with the monotonicity axiom followed by the locality axiom.

Lemma 8.5. *For any $\mathcal{C} \subseteq \mathcal{D}$ and $E \in \mathcal{D}$, we have $w\big(\mathcal{C} \cup \{E\}\big) \leqslant w(\mathcal{C})$.*

Proof. Set $\mathcal{C}^* = \mathcal{C} \cup \{E\}$. Let D be the smallest destroyer of $\mathcal{C}$, and let $r = \mathrm{rad}(\mathcal{C})$ be the radius of D. Since D destroys $\mathcal{C}_{<r}$, the set $\mathcal{C}_{<r} \cup \{D\}$ is non-Helly. Moreover, since $\mathcal{C}_{<r} \cup \{D\} \subseteq \mathcal{C}^*_{<r} \cup \{D\}$, we know that $\mathcal{C}^*_{<r} \cup \{D\}$ is also non-Helly. Therefore, D destroys $\mathcal{C}^*_{<r}$ and we can derive $\mathrm{rad}(\mathcal{C}^*) \leqslant \mathrm{rad}(\mathcal{C})$. If we have $\mathrm{rad}(\mathcal{C}^*) < \mathrm{rad}(\mathcal{C})$, we are done. Hence, assume that $\mathrm{rad}(\mathcal{C}^*) = \mathrm{rad}(\mathcal{C})$. Then D is the smallest destroyer of $\mathcal{C}^*$, and Lemma 8.4 gives $-\mathrm{dist}(\mathcal{C}^*) = -d\big(\bigcap \mathcal{C}^*_{<r}, D\big) \leqslant -d(\bigcap \mathcal{C}_{<r}, D) = -\mathrm{dist}(\mathcal{C})$. Hence, $w\big(\mathcal{C}^*\big) \leqslant w(\mathcal{C})$. See Figure 8.3 for an illustration. □

Lemma 8.6. *Let $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{D}$ with $w(\mathcal{B}) = w(\mathcal{C})$ and let $E \in \mathcal{D}$. Then, if $w\big(\mathcal{B} \cup \{E\}\big) = w(\mathcal{B})$, we also have $w\big(\mathcal{C} \cup \{E\}\big) = w(\mathcal{C})$.*

Proof. Set $\mathcal{C}^* = \mathcal{C} \cup \{E\}$, $\mathcal{B}^* = \mathcal{B} \cup \{E\}$. Let $r = \mathrm{rad}(\mathcal{C})$ and D be the smallest destroyer of $\mathcal{C}$. Since $w(\mathcal{C}) = w(\mathcal{B}) = w(\mathcal{B}^*)$, we have that D is also the smallest destroyer of $\mathcal{B}$ and of $\mathcal{B}^*$. If the radius of E is larger than r, then E cannot be the smallest destroyer of $\mathcal{C}^*$, so $w\big(\mathcal{C}^*\big) = w(\mathcal{C})$. Thus, assume that E has radius less than r. Let v be the extreme point of

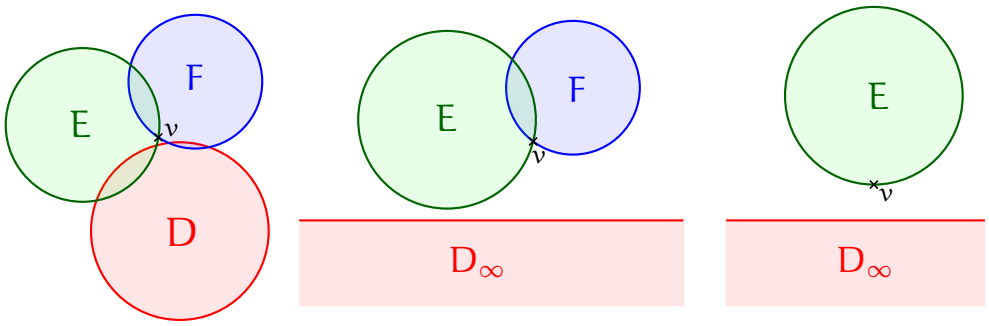


Figure 8.4: A basis can either be a non-Helly triple (left), a pair of intersecting disks E and F where the point of minimum y-coordinate in $E \cap F$ is a vertex (middle), or a single disk (right).

$\mathcal{C}_{<r}$ and D. Since $w(\mathcal{B}^*) = w(\mathcal{B})$, we know that $d\big(\bigcap \mathcal{B}_{<r}, D\big) = d\big(\bigcap \mathcal{B}^*_{<r}, D\big) = d(v, D)$. Now, Lemma 8.4 implies that $v \in E$, since $E \in \mathcal{B}^*_{<r}$. Thus, the set $\mathcal{C}^*_{<r} = \mathcal{C}_{<r} \cup \{E\}$ is Helly and therefore, there is no disk $D' \in \mathcal{C}^*_{<r}$ that destroys $\mathcal{C}^*_{<r}$. Furthermore, since D destroys $\mathcal{C}_{<r}$ and $\mathcal{C}_{<r} \subset \mathcal{C}^*_{<r}$, the disk D also destroys $\mathcal{C}^*_{<r}$. Therefore, D is also the smallest destroyer of $\mathcal{C}^*$, so $\operatorname{rad}(\mathcal{C}^*) = r = \operatorname{rad}(\mathcal{C})$. Finally, since $\mathcal{B}^*_{<r} \subseteq \mathcal{C}^*_{<r}$ we can use Lemma 8.4 to derive

$$d\Big(\bigcap \mathcal{C}_{<r}, D\Big) = d\Big(\bigcap \mathcal{B}^*_{<r}, D\Big) \leqslant d\Big(\bigcap \mathcal{C}^*_{<r}, D\Big) \leqslant d(v, D) = d\Big(\bigcap \mathcal{C}_{<r}, D\Big).$$

The claim follows. □

Next, we want to describe the violation test for $(\mathcal{D}, w, \leqslant)$: given a basis $\mathcal{B} \subseteq \mathcal{D}$ and a disk $E \in \mathcal{D}$, check whether E violates $\mathcal{B}$, i.e., whether $w\big(\mathcal{B} \cup \{E\}\big) < w(\mathcal{B})$, and return an error message if $\mathcal{B}$ is not a basis. But before, we show that the combinatorial dimension of $(\mathcal{D}, w, \leqslant)$ is at most 3.

Lemma 8.7. *For each $\mathcal{C} \subseteq \mathcal{D}$, there is a set $\mathcal{B} \subseteq \mathcal{C}$ with $|\mathcal{B}| \leqslant 3$ and $w(\mathcal{B}) = w(\mathcal{C})$.*

Proof. Let D be the smallest destroyer of $\mathcal{C}$. Let $r = \operatorname{rad}(\mathcal{C})$ be the radius of D, and let $v \in \bigcap \mathcal{C}_{<r}$ be the extreme point for $\mathcal{C}_{<r}$ and D. First of all, we observe that v cannot be in the interior of $\bigcap \mathcal{C}_{<r}$, since v minimizes the distance to D. Thus, there has to be a non-empty subset $\mathcal{A} \subseteq \mathcal{C}_{<r}$ such that v lies on the boundary of each disk of $\mathcal{A}$. Let $\mathcal{A}$ be a minimal set such that $d(\bigcap \mathcal{A}, D) = d(v, D)$. It follows that $|\mathcal{A}| \leqslant 2$. See Figure 8.4 for an illustration.

First, assume that $\mathcal{A} = \{E\}$. Then, since $d(E, D) = d(v, D) > 0$, we know that $E \cap D = \emptyset$. As the disks in $\mathcal{C}$ intersect pairwise, we derive $D \notin \mathcal{C}$ and hence $D = D_\infty$. Setting $\mathcal{B} = \mathcal{A}$, we get $\operatorname{rad}(\mathcal{C}) = \infty = \operatorname{rad}(\mathcal{B})$ and $\operatorname{dist}(\mathcal{C}) = d(v, D) = d(E, D) = \operatorname{dist}(\mathcal{B})$. Thus, $|\mathcal{B}| \leqslant 3$ and $w(\mathcal{B}) = w(\mathcal{C})$.

Second, assume that $\mathcal{A} = \{E, F\}$. Then, v is one of the two vertices of the lens $L = E \cap F$. Next, we show that $d(L, D) \geqslant d(v, D)$. Assume for the sake of contradiction that there is a point $w \in L$ with $d(w, D) < d(v, D)$. By general position and since v is the intersection of two disk boundaries, there is a relatively open neighborhood N around v in $\bigcap \mathcal{C}_{<r}$ such that

N is also relatively open in L. Since L is convex, there is a point $x \in N$ that also lies in the relative interior of the line segment $\overline{wv}$. Then, $d(x, D) < d(v, D)$ and $x \in \bigcap \mathcal{C}_{<r}$. This yields a contradiction, as v is the extreme point for $\mathcal{C}_{<r}$ and D. Thus, we have $d(L, D) \geqslant d(v, D)$ which also shows hat $D \cap E \cap F = \emptyset$.

We set $\mathcal{B} = \{E, F\}$, if $\mathcal{C}$ is Helly (i.e., $D = D_\infty$), and $\mathcal{B} = \{D, E, F\}$, if $\mathcal{C}$ is non-Helly (i.e., $D \in \mathcal{C}$). In both cases, we have $\mathcal{B} \subseteq \mathcal{C}$ and $|\mathcal{B}| \leqslant 3$. Moreover, we can conclude that D destroys $\mathcal{B}_{<r} = \{E, F\}$, and since $\mathcal{B}_{<r}$ is Helly, D is the smallest destroyer of $\mathcal{B}$. Hence, we have $\mathrm{rad}(\mathcal{C}) = r = \mathrm{rad}(\mathcal{B})$.

To obtain $\mathrm{dist}(\mathcal{B}) = \mathrm{dist}(\mathcal{C})$, it remains to show $d(\bigcap \mathcal{B}_{<r}, D) = d(\bigcap \mathcal{C}_{<r}, D)$. Since $\mathcal{B}_{<r} \subseteq \mathcal{C}_{<r}$, we can use Lemma 8.4 as well as $d(L, D) \geqslant d(v, D)$ to derive

$$d\Big(\bigcap \mathcal{C}_{<r}, D\Big) \geqslant d\Big(\bigcap \mathcal{B}_{<r}, D\Big), = d(L, D) \geqslant d(v, D) = d\Big(\bigcap \mathcal{C}_{<r}, D\Big)$$

as desired. We conclude that $w(\mathcal{B}) = w(\mathcal{C})$.

We remark that the set $\mathcal{B}$ is actually a basis for $\mathcal{C}$: if $\mathcal{B}$ is a non-Helly triple, then removing any disk from $\mathcal{B}$ creates a Helly set and increases the radius of the smallest destroyer to ∞. If $|\mathcal{B}| \leqslant 2$, then D_∞ is the smallest destroyer of $\mathcal{B}$ and the minimality follows directly from the definition. □

Algorithm 8.1 The violation test.

```
1:  procedure VIOLATES(set B ⊆ D, disk E ∈ D with radius r′)
2:      if |B| > 3 or |B| = 3 and B is Helly then return "B is not a basis."
3:      if |B| = 2 and y-minimum of ⋂B is also y-minimum of a single disk of B then
4:          return "B is not a basis."
5:      if B = {D1} then
6:          if y-minimum in E ∩ D1 differs from y-minimum in D1 then
7:              return "E violates B."
8:          else return "E does not violate B."
9:      if B = {D1, D2} then
10:         v = argmin {w_y | w ∈ D1 ∩ D2}
11:         if v ∉ E then return "E violates B."
12:         else return "E does not violate B."
13:     else                          ▷ B is of size 3, non-Helly, and does not contain D∞.
14:         D = smallest destroyer of B
15:         {D1, D2} = B \ {D}
16:         r = rad(B)
17:         if r′ > r then return "E does not violate B."
18:         else
19:             v = argmin {d(w, E) | w ∈ D1 ∩ D2}
20:             if v ∉ E then return "E violates B."
21:             else return "E does not violate B."
```

Following the argumentation of the last proof, the violation test is now immediate and presented in Algorithm 8.1. It obviously needs constant time. Finally, to apply the algorithm of Chazelle and Matoušek we still need to check that there is a polynomial-time oracle that computes a superset of $\mathcal{R}_\mathcal{Y}$ for a given set of disks $\mathcal{Y}$.

Lemma 8.8. *The range space* $(\mathcal{D}, \mathcal{R} = \{\text{vio}(\mathcal{B}) \mid \mathcal{B} \text{ is a basis for some } \mathcal{D}' \subseteq \mathcal{D}\})$ *has constant shattering dimension and constant VC-dimension. Moreover, given a set* $\mathcal{Y} \subseteq \mathcal{D}$ *of disks, we can compute a superset of* $\mathcal{R}_\mathcal{Y}$ *in time* $O(|\mathcal{Y}|^4)$.

Proof. Let $v \in \mathbb{R}^2$ and $r > 0$. First, we let $R_v = \{D \in \mathcal{Y} \mid v \notin D\}$ be the range of all disks that do not contain v. Second, let $R_{v,r}$ be the range of all disks of diameter smaller than r that do not contain the point v, i.e., $R_{v,r} = \{D \in \mathcal{Y} \mid v \notin D \text{ and } r_D < r\}$. We define $\mathcal{R}'$ to be the set of all ranges R_v over all v and subsequently, we let $\mathcal{R}''$ be the set of all ranges $R_{v,r}$ over all v and r, that is, $\mathcal{R}'' = \{R_{v,r} \mid v \in \mathbb{R}^2 \text{ and } r > 0\}$.

The discussion from the previous lemmas shows that for any basis $\mathcal{B}$, there is a point $v_\mathcal{B} \in \mathbb{R}^2$ and a radius $r_\mathcal{B} > 0$ such that a disk $E \in \mathcal{D}$ with radius r_E violates $\mathcal{B}$ if and only if $v_\mathcal{B} \notin E$ and $r_E < r_\mathcal{B}$. Hence, we have $\mathcal{R}'' \supseteq \mathcal{R}_\mathcal{Y}$. We show how to compute $\mathcal{R}''$ in polynomial time. For this, we first construct $\mathcal{R}'$.

For the given set $\mathcal{Y}$ of disks, we compute the arrangement $A(\mathcal{Y})$ and then focus on the facets of $A(\mathcal{Y})$. Since the arrangement has $O(|\mathcal{Y}|^2)$ facets, we can compute $A(\mathcal{Y})$ in time $O(|\mathcal{Y}|^3)$ using a simple brute-force approach (faster algorithms exist, but are not needed here). Clearly, for two points v and w of the same facet of $A(\mathcal{Y})$, we have $R_v = R_w$. Therefore, for a given facet f, we pick an arbitrary point $v \in f$, and we compute R_v by a linear scan of $\mathcal{Y}$. Summing over all facets, we can thus compute $\mathcal{R}'$ in time $O(|\mathcal{Y}|^3)$.

Finally, to compute $\mathcal{R}''$, we iterate over all $O(|\mathcal{Y}|^2)$ ranges in $\mathcal{R}'$. Given a range $R_v \in \mathcal{R}'$, we get all $R_{v,r}$ for $r > 0$ by first sorting R_v by increasing radii and then taking every prefix of the sorted list of disks. For a fixed v, this can be done in time $O(|\mathcal{Y}|^2)$. Hence, $\mathcal{R}''$ can be computed in $O(|\mathcal{Y}|^4)$ time. Moreover, since $\mathcal{R}''$ has cardinality $O(|\mathcal{Y}|^3)$ and $\mathcal{R}_\mathcal{Y} \subseteq \mathcal{R}''$ we conclude that the shattering dimension is at most 3 and by Lemma 2.1 the VC-dimension is a constant. This finishes the proof. □

The following lemma summarizes the discussion so far.

Lemma 8.9. *Given a set* $\mathcal{D}$ *of* n *pairwise intersecting disks in the plane, we can decide in* $O(n)$ *deterministic time whether* $\mathcal{D}$ *is Helly. If so, we can compute a point in* $\bigcap \mathcal{D}$ *in* $O(n)$ *deterministic time. If not, we can compute the smallest destroyer* D *of* $\mathcal{D}$ *and two disks* $E, F \in \mathcal{D}_{<r}$ *that form a non-Helly triple with* D. *Here,* r *is the radius of* D.

Proof. Since (i) $(\mathcal{D}, w, \leqslant)$ is LP-type, (ii) the violation test needs constant time, and (iii) the oracle needs polynomial time, we can apply the deterministic algorithm of Chazelle and Matoušek [CM96] to compute $w(\mathcal{D}) = (\text{rad}(\mathcal{D}), -\text{dist}(\mathcal{D}))$ and a corresponding basis $\mathcal{B}$ in $O(n)$ time. Then, $\mathcal{D}$ is Helly if and only if $\text{rad}(\mathcal{D}) = \infty$. If $\mathcal{D}$ is Helly, then $|\mathcal{B}| \leqslant 2$. We compute the unique point $v \in \bigcap \mathcal{B}$ with $d(v, D_\infty) = d\big(\bigcap \mathcal{B}, D_\infty\big)$. Since $\mathcal{B} \subseteq \mathcal{D}$ and $d\big(\bigcap \mathcal{B}, D_\infty\big) = d\big(\bigcap \mathcal{D}, D_\infty\big)$, we have $v \in \bigcap \mathcal{D}$ by Lemma 8.4. We output v. If $\mathcal{D}$ is non-Helly, we simply output $\mathcal{B}$, because $\mathcal{B}$ is a non-Helly triple with the smallest destroyer D of $\mathcal{D}$ and two disks $E, F \in \mathcal{D}_{<r}$, where r is the radius of D. □

Theorem 8.10. *Given a set $\mathcal{D}$ of n pairwise intersecting disks in the plane, we can find in deterministic $O(n)$ time a set P of five points such that every disk of $\mathcal{D}$ contains at least one point of P.*

Proof. Using the algorithm from Lemma 8.9, we decide whether $\mathcal{D}$ is Helly. If so, we return the extreme point computed by the algorithm. Otherwise, the algorithm gives us a non-Helly triple $\{D, E, F\}$, where D is the smallest destroyer of $\mathcal{D}$ and $E, F \in \mathcal{D}_{<r}$, with r being the radius of D. Since $\mathcal{D}_{<r}$ is Helly, we can obtain in $O(n)$ time a stabbing point $q \in \bigcap \mathcal{D}_{<r}$ by using the algorithm from Lemma 8.9 again. Next, by Lemma 7.1, there are two disks in $\{D, E, F\}$ whose lens angle is at least $2\pi/3$. Let P' be the set of four points from the proof of Lemma 7.4. Then, $P = P' \cup \{q\}$ is a set of five points that stabs every disk in $\mathcal{D}$. □

CHAPTER 9

Conclusions

We presented three completely new routing schemes for geometrically induced graph classes and gave insights on the stabbing number of pairwise intersecting disks.

Routing in histograms. We gave the first routing schemes for the hop-distance in simple polygons. In particular, we have a routing scheme for simple and double histograms with label size $\lceil \log n \rceil$, routing table size $O(\log n \deg(v))$ for each vertex v and preprocessing time $O(m)$. While in simple histograms we can route on optimal paths, we have stretch 2 in double histograms and need additional headers of size $\lceil \log n \rceil$. This constitutes a first step towards an efficient routing scheme for the hop-distance in orthogonal polygons. Moreover, compared to the best known result by Roditty and Tov [RT16] we have better routing table sizes and achieve much better stretch factors. The following open problems arise naturally.

First, it would be interesting to know, whether it is possible to decrease the stretch in double histograms to say $1 + \varepsilon$, for $\varepsilon > 0$. One approach here might be to store more k-dominators in the routing table. Hence, the routing table would increase and might depend on ε but we should be able to have more hops on a shortest path. However, the crucial problem of deciding whether the near or the far dominator is the better choice remains open.

As a next step, it would be interesting to see whether the general position assumption can be removed. Next, we would like to know how the routing scheme extends to monotone polygons as well as arbitrary orthogonal polygons, assuming r-visibility. A promising approach seems to be the well-known *window decomposition* that partitions an orthogonal polygon into histograms; see for instances [Bär11, BGM+14, BGR17, BS14, HKS+18]. This decomposition yields a tree T, in which a node corresponds to a histogram and two histograms are adjacent in T if and only if they are adjacent in the polygon. Combining the tree routing with the (double or simple) histogram routing might give a proper routing scheme. However, the details remain open.

After that, it will be interesting to take a closer look at (orthogonal) polygons assuming ℓ-visibility. Here, the structure of visibility – even in simple histograms – is much more complicated. Moreover, we can no longer assume integer coordinates.

Routing in unit disk graphs. We presented an efficient and compact routing scheme for unit disk graphs. It achieves stretch $1+\varepsilon$ and uses $\varepsilon^{-O(\varepsilon^{-2})}\log^3 n(1+\log D/\log\log n)$ bits in the tables and labels. The header size is bounded by $O(\log^2 n/\log\log n)$. It would be interesting to see if this result can be extended to disk graphs in general. If the radii of the disks are unbounded, the decomposition of Chan and Skrepetos cannot be applied immediately. Moreovoer, if we want to decrease the size of the dynamic header and analyze the preprocessing time we have to take a closer look into the routing scheme of Konjevod et al. [KRX16] which we used as black box.

Finally, let us compare the routing scheme to the known schemes for unit disk graphs. The model of the routing scheme of Kaplan et al. [KMRS18] is very close to ours. The routing scheme can be implemented using the fixed-port model. Instead of the shortest-path separator decomposition they use the well-separated pair decomposition by Gao and Zhang [GZ05]. We achieve the same stretch factor and still use additional information of poly-logarithmic size. Their scheme was generalized to non-unit disk graphs with constant bounded radii [Wil16]. Our main advantage is, that we do not use neighborhood oracles: Kaplan et al. assumes that the routing function has direct access on the neighborhood of a current vertex (without using label, table or header), see Section 5.4 in [KMRS18]. The existence of such a neighborhood oracle makes the routing much easier, since it is a crucial problem to efficiently route in the neighborhood. However, it is not clear how their scheme can be implemented without such an oracle.

The idea of the routing scheme of Yan et al. [YXD12] is similar to ours: the graph is covered by $O(\log n)$ different trees. When the routing starts, the labels of the source and the target are used to determine the identity of a tree and an $O(\log n)$-bit label of the target within this tree. Finally, they completely forget the original labels and route within this tree until they reach the target. For any two vertices $s, t \in V$, the routing path between s and t has length at most $5 \cdot d(s,t) + 13$. Our routing scheme can also be transformed to match into this model, but we have $O(\log D \log n)$ different trees that cover the unit disk graph and the label of a vertex in one of the trees has size $O(\log^2 n/\log\log n)$. Nevertheless, we achieve the near optimal stretch $1+\varepsilon$. Moreover, Yan et al. use the designer-port model and thus, they can route within a tree using labels of size $O(\log n)$. But since nodes are contained in more than one tree, there have to be lookup-tables for the port assignments. Their routing scheme can easily be turned into the fixed-port model: the stretch would not change and the label size would increase to $O(\log^3 n/\log\log n)$. Last but not least, their routing scheme also achieves constant hop stretch. It is unlikely that the hop stretch of our routing scheme is bounded by a constant. In conclusion, our routing scheme needs an $O(\log D)$-factor more in the label size but achieves a better stretch if $\varepsilon < 4$. Moreover, our underlying routing model is specified more clearly.

Stabbing We gave a simple linear-time algorithm to find five stabbing points for a set of pairwise intersecting disks in the plane. Here we used the concept of LP-type problems. It remains open how to use the proofs of Danzer or Stachó [Sta84, Dan86] for an efficient construction of four stabbing points. However, the arXiv proposal by Carmi, Katz, and Morin [CKM18] claims a linear-time algorithm for finding four stabbing points. It would now be

interesting to see whether these results, the ones by Danzer, Stachó, and ours could be used to find new deterministic approximation algorithms for computing large cliques in disk graphs; refer to [AW05, BBB+18] for the known algorithms. Moreover, it is still not known whether nine disks can always be stabbed by three points or not. For eight disks we provided a proof that three points always suffice, see also [Sta65]. Furthermore, the lower bound construction of Danzer with ten disks [Dan86] works for pseudo-disks. The example is not easy to draw, even with the help of geometry processing software. However, until now, we were not able to check whether his pseudo-disk arrangement can be realized as Euclidean disk arrangement.

Bibliography

[ABNLP90] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, 11(3):307–341, 1990.

[ACL+06] Marta Arias, Lenore J Cowen, Kofi A Laing, Rajmohan Rajaraman, and Orjeta Taka. Compact routing with name independence. *SIAM Journal on Discrete Mathematics*, 20(3):705–726, 2006.

[AG11] Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *Proceedings 25th International Symposium on Distributed Computing (DISC)*, pages 404–415, 2011.

[AGGM06] Ittai Abraham, Cyril Gavoille, Andrew V. Goldberg, and Dahlia Malkhi. Routing in networks with low doubling dimension. In *Proceedings 26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, page 75, 2006.

[AGM04a] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. Routing with improved communication-space trade-off. In *Proceedings 18th International Symposium on Distributed Computing (DISC)*, pages 305–319, 2004.

[AGM+04b] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In *Proceedings 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 20–24, 2004.

[AK92] Noga Alon and Daniel J. Kleitman. Piercing convex sets and the Hadwiger-Debrunner (p, q)-problem. *Advances in Mathematics*, 96(1):103–112, 1992.

[AW05] Christoph Ambühl and Uli Wagner. The clique problem in intersection graphs of ellipses and triangles. *Theory of Computing Systems*, 38(3):279–292, 2005.

[Bär11] Andreas Bärtschi. Coloring variations of the art gallery problem. Master's thesis, Department of Mathematics, ETH Zürich, 2011.

[BBB+18] Marthe Bonamy, Edouard Bonnet, Nicolas Bousquet, Pierre Charbit, and Stéphan Thomassé. EPTAS for max clique on disks and unit balls. In *Proceedings 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 568–579, 2018.

[BCK+17] Bahareh Banyassady, Man-Kwun Chiu, Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, Paul Seiferth, Yannik Stein, Birgit Vogtenhuber, and Max Willert. Routing in polygonal domains. In *Proceedings 28th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 10:1–10:13, 2017.

[BFC00] Michael A Bender and Martin Farach-Colton. The lca problem revisited. In *Proceedings 4th Latin American Symposium in Theoretical Informatics. (LATIN)*, pages 88–94, 2000.

[BFvRV15] Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Optimal local routing on Delaunay triangulations defined by empty equilateral triangles. *SIAM Journal on Computation (SICOMP)*, 44(6):1626–1649, 2015.

[BFvRV17] Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Competitive local routing with constraints. *Journal of Computational Geometry*, 8(1):125–152, 2017.

[BGM+14] Andreas Bärtschi, Subir Kumar Ghosh, Matúš Mihalák, Thomas Tschager, and Peter Widmayer. Improved bounds for the conflict-free chromatic art gallery problem. In *Proceedings 30th Annual Symposium on Computational Geometry (SoCG)*, page 144, 2014.

[BGP17] Pritam Bhattacharya, Subir Kumar Ghosh, and Sudebkumar Pal. Constant approximation algorithms for guarding simple polygons using vertex guards. `arXiv:1712.05492`, 2017.

[BGR17] Pritam Bhattacharya, Subir Kumar Ghosh, and Bodhayan Roy. Approximability of guarding weak visibility polygons. *Discrete Applied Mathematics*, 228:109–129, 2017.

[Bin20] Ahmad Biniaz. Plane hop spanners for unit disk graphs: Simpler and better. *Computational Geometry*, 89:101622, 2020.

[BKvRV17a] Prosenjit Bose, Matias Korman, André van Renssen, and Sander Verdonschot. Constrained routing between non-visible vertices. In *Proceedings 23rd International Conference on Computing and Combinatorics (COCOON)*, pages 62–74, 2017.

[BKvRV17b] Prosenjit Bose, Matias Korman, André van Renssen, and Sander Verdonschot. Routing on the visibility graph. In *Proceedings 28th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 18:1–18:12, 2017.

[BLT07] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Improved sparse covers for graphs excluding a fixed minor. In *Proceedings 26th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 61–70, 2007.

[BLT14] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Sparse covers for planar graphs and graphs that exclude a fixed minor. *Algorithmica*, 69(3):658–684, 2014.

[BM04] Prosenjit Bose and Pat Morin. Competitive online routing in geometric graphs. *Theoretical Computer Science*, 324(2):273–288, 2004.

[BS14] Andreas Bärtschi and Subhash Suri. Conflict-free chromatic art gallery coverage. *Algorithmica*, 68(1):265–283, 2014.

[BV93] Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242, 1993.

[CCJ90] Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.

[CCK+20] Man-Kwun Chiu, Jonas Cleve, Katharina Klost, Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, and Max Willert. Routing in histograms. In *Proceedings 14th International Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 43–54, 2020.

[CGMZ16] T.-H. Hubert Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. *ACM Transactions on Algorithms*, pages 55:1–55:22, 2016.

[Cha01] Bernard Chazelle. *The Discrepancy Method—Randomness and Complexity*. Cambridge University Press, Cambridge, 2001.

[Cha04] Timothy M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–436, 2004.

[Che13] Shiri Chechik. Compact routing schemes with improved stretch. In *Proceedings ACM Symposium on Principles of Distributed Computing (PODC)*, pages 33–41, 2013.

[CKM18] Paz Carmi, Matthew J. Katz, and Pat Morin. Stabbing pairwise intersecting disks by four points. `arXiv:1812.06907`, 2018.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[CM96] Bernard Chazelle and Jiři Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21(3):579–597, 1996.

[Cow01] Lenore J Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001.

[CS19] Timothy M. Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *Journal of Computational Geometry*, 10(2):3–20, 2019.

[Dan86] Ludwig Danzer. Zur Lösung des Gallaischen Problems über Kreisscheiben in der Euklidischen Ebene. *Studia Scientiarum Mathematicarum Hungarica*, 21(1-2):111–134, 1986.

[DBVKOS97] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry*. Springer, 1997.

[DJ11] Adrian Dumitrescu and Minghui Jiang. Piercing translates and homothets of a convex body. *Algorithmica*, 61(1):94–115, 2011.

[EGP03] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *Journal of Algorithms*, 46(2):97–114, 2003.

[FG01] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proceedings 28th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 757–772, 2001.

[FG02] Pierre Fraigniaud and Cyril Gavoille. A space lower bound for routing in trees. In *Proceedings 19th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 65–75, 2002.

[FH06] Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Proceedings 17th Annual Symposium on Combinatorial Pattern Matching*, pages 36–48, 2006.

[Gav01] Cyril Gavoille. Routing in distributed networks: overview and open problems. *SIGACT News*, pages 36–52, 2001.

[GG01] Cyril Gavoille and Marc Gengler. Space-efficiency for routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61(5):679–687, 2001.

[Gol21] Nils Goldmann. Complexity analysis for a labeled routing scheme. Bachelor's thesis, Department of Computer Science, Freie Universität Berlin, 2021.

[Grü59] Branko Grünbaum. On intersections of similar sets. *Portugaliae Mathematica*, 18:155–164, 1959.

[GS04] Silvia Giordano and Ivan Stojmenovic. Position based routing algorithms for ad hoc networks: A taxonomy. In *Ad hoc wireless networking*, pages 103–136. Springer, 2004.

[GZ05] Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computation (SICOMP)*, pages 151–169, 2005.

[HD55] Hugo Hadwiger and Hans Debrunner. Ausgewählte Einzelprobleme der kombinatorischen Geometrie in der Ebene. *Enseignement Mathématique*, 1:56–89, 1955.

[Hel23] Eduard Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.

[Hel30] Eduard Helly. Über Systeme von abgeschlossenen Mengen mit gemeinschaftlichen Punkten. *Monatshefte für Mathematik*, 37(1):281–302, 1930.

[HKM+18] Sariel Har-Peled, Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, Micha Sharir, and Max Willert. Stabbing pairwise intersecting disks by five points. In *Proceedings 29th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 50:1–50:12, 2018.

[HKS+18] Frank Hoffmann, Klaus Kriegel, Subhash Suri, Kevin Verbeek, and Max Willert. Tight bounds for conflict-free chromatic guarding of orthogonal art galleries. *Computational Geometry Theory & Applications (CGTA)*, 73:24–34, 2018.

[Hof90] Frank Hoffmann. On the rectilinear art gallery problem. In *Proceedings 17th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 717–728, 1990.

[HP11] Sariel Har-Peled. *Geometric approximation algorithms*. American Mathematical Society, 2011.

[HT73] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[JV09] Philippe Jacquet and Laurent Viennot. Remote-spanners: What to know beyond neighbors. In *Proceedings 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–10, 2009.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

[KMRS18] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. *Algorithmica*, 80(3):830–848, 2018.

[KRX16] Goran Konjevod, Andréa W Richa, and Donglin Xia. Scale-free compact routing schemes in networks of low doubling dimension. *ACM Transactions on Algorithms*, 12(3):1–29, 2016.

[KST13] Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In *Proceedings 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 550–563, 2013.

[LCW02] Xiang-Yang Li, Gruia Calinescu, and Peng-Jun Wan. Distributed construction of a planar spanner and routing for ad hoc wireless networks. In *Proceedings 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1268–1277, 2002.

[LT79] Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Computation (SICOMP)*, 36(2):177–189, 1979.

[LvK10] Maarten Löffler and Marc van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry Theory & Applications (CGTA)*, 43(4):419–433, 2010.

[MRS90] Rajeev Motwani, Arvind Raghunathan, and Huzur Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *Journal of Computer and System Sciences (JCSS)*, 40(1):19–48, 1990.

[MW20] Wolfgang Mulzer and Max Willert. Compact routing in unit disk graphs. In *Proceedings 31st Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 16:1–16:14, 2020.

[O'R87] Joseph O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press, 1987.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, 1989.

[Rad21] Johann Radon. Mengen konvexer Körper, die einen gemeinsamen Punkt enthalten. *Mathematische Annalen*, 83(1):113–115, 1921.

[RT15] Liam Roditty and Roei Tov. New routing techniques and their applications. In *Proceedings ACM Symposium on Principles of Distributed Computing (PODC)*, pages 23–32, 2015.

[RT16] Liam Roditty and Roei Tov. Close to linear space routing schemes. *Distributed Computing*, 29(1):65–74, 2016.

[Sau72] Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

[Sei91] Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry (DCG)*, 6:423–434, 1991.

[She72] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

[SK85] Nicola Santoro and Ramez Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28(1):5–8, 1985.

[Sli05] Aleksandrs Slivkins. Distance estimation and object location via rings of neighbors. In *Proceedings 24th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 41–50, 2005.

[Sta65] Lajos Stachó. Über ein Problem für Kreisscheibenfamilien. *Acta Scientiarum Mathematicarum (Szeged)*, 26:273–282, 1965.

[Sta84] Lajos Stachó. A solution of Gallai's problem on pinning down circles. *Matematikai Lapok*, 32(1-3):19–47, 1981/84.

[SW92] Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. *Proceedings 9th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 567–579, 1992.

[Tal04] Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings 36th ACM Symposium on Theory of Computing (STOC)*, pages 281–290, 2004.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings 13th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2001.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[VC15] Vladimir Vapnik and Alexey Chervonegnkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[Wil16] Max Willert. Routing schemes for disk graphs and polygons. Master's thesis, Department of Computer Science, Freie Universität Berlin, 2016.

[WK07] Chris Worman and J Mark Keil. Polygon decomposition and the orthogonal art gallery problem. *International Journal of Computational Geometry & Applications*, 17(02):105–138, 2007.

[WX20] Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete & Computational Geometry (DCG)*, pages 1–26, 2020.

[YXD12] Chenyu Yan, Yang Xiang, and Feodor F Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *Computational Geometry Theory & Applications (CGTA)*, 45(7):305–325, 2012.

[Zuc06] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings 38th ACM Symposium on Theory of Computing (STOC)*, pages 681–690, 2006.

Zusammenfassung

Routen. Wir betrachten *Routen mit Vorverarbeitung* in einem Graphen G. Während der Vorverarbeitung von G erhält jeder Knoten ein *Label* und eine *Routingtabelle*. Danach müssen wir in der Lage sein ein Datenpaket zwischen je zwei Knoten s und t von G zu routen, wobei jeder Schritt lediglich das Label von t, die Routingtabelle von s und den Header des Pakets benutzen darf. Das Routingproblem wurde bereits ausführlich für allgemeine Graphen erforscht. Es hat sich herausgestellt, dass kompakte (polylogarithmisch große Routingtabellen) und effiziente Routingschemata für allgemeine Graphen nicht existieren.

Sei P ein x-monotones orthogonales Polygon mit n Ecken. Wir bezeichnen P als *einfaches Histogramm*, wenn der obere Teil des Randes eine einzelne Strecke ist. P ist ein *doppeltes Histogramm*, wenn es eine horizontale Sehne gibt, welche vom linken zum rechten Rand geht. Zwei Punkte p und q sind *co-sichtbar* genau dann, wenn das achsenparallele Rechteck, aufgespannt von p und q, komplett in P liegt. Im r-*Sichtbarkeitsgraphen* $\mathrm{Vis}(P)$ gibt es eine Kante zwischen je zwei co-sichtbaren Ecken von P. Wir präsentieren ein Routingschema für Sichtbarkeitsgraphen von einfachen und doppelten Histogrammen, welches Labelgröße $\lceil \log n \rceil$ und Routingtabellengröße $O(\log n \deg(v))$ für jede Ecke v von P erreicht, wobei $\deg(v)$ der Grad von v in $\mathrm{Vis}(P)$ ist. In einfachen Histogrammen können wir entlang eines kürzesten Weges routen und benötigen keinen zusätzlichen Header. Für doppelte Histogramme benötigen wir einen Header der Größe $\lceil \log n \rceil$ und erreichen Stretch 2. In beiden Fällen ist die Vorverarbeitungszeit asymptotisch zur Anzahl der Kanten von $\mathrm{Vis}(P)$.

Sei $V \subset \mathbb{R}^2$ eine Menge von n Punkten in der Ebene. Der *Einheitskreisgraph* $\mathrm{DG}(V)$ ist ein Graph mit Knotenmenge V und einer Kante zwischen je zwei Knoten v und w, wenn ihr Euklidischer Abstand höchstens 1 ist. Die Kantengewichte sind die Euklidischen Abstände. Sei außerdem D der Durchmesser des Graphen. Wir konstruieren für jedes $\varepsilon > 0$ ein Routingschema für $\mathrm{DG}(V)$. Das Schema erreicht Stretch $1 + \varepsilon$, Labelgröße $O\left(\varepsilon^{-1} \log D \log^3 n / \log\log n\right)$, Routingtabellengröße $\varepsilon^{-O(\varepsilon^{-2})} \log^3 n (1 + \log D / \log\log n)$ und Headergröße $O(\log^2 n / \log\log n)$. Die Vorverarbeitungszeit beträgt $O(\varepsilon^{-1} n^2 \log^2 n)$.

Piercen. Sei $\mathcal{D}$ eine Menge von n sich paarweise schneidenden Kreisscheiben in der Ebene. Eine Punktmenge P *pierct* $\mathcal{D}$ genau dann, wenn jede Kreisscheibe aus $\mathcal{D}$ mindestens einen Punkt aus P enthält. Wir präsentieren einen deterministischen Algorithmus, der $O(n)$ Zeit benötigt um 5 Punkte zu finden, die $\mathcal{D}$ piercen. Damit liefern wir eine einache, wenn auch etwas schwächere, algorithmische Version des klassischen Ergebnisses von Danzer, wonach eine solche Menge $\mathcal{D}$ immer von 4 Punkten gepierct werden kann. Außerdem geben wir ein einfaches Beispiel mit 13 sich paarweise schneidenden Kreisscheiben an, welches nicht von 3 Punkten gepierct werden kann.