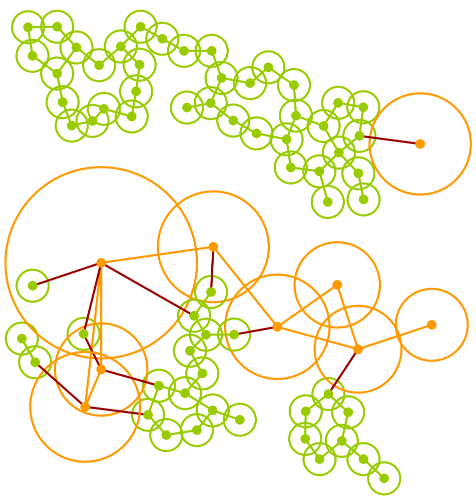
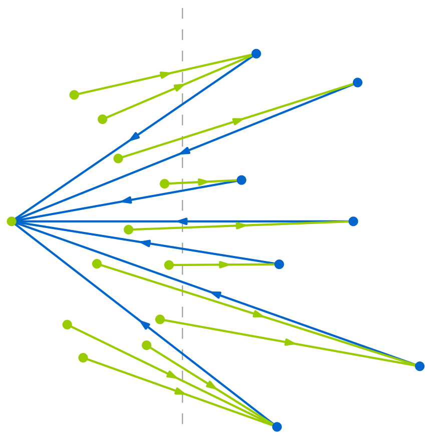
Geometric Graphs: Reachability, Long Trees and Short Cycles





Dissertation zur Erlangung des akademischen Grades
„Doktorin der Naturwissenschaften“

von

Katharina Klost

vorgelegt am

Fachbereich Mathematik und Informatik der Freien Universität Berlin

2021

Betreuer und Erstgutachter: Prof. Dr. Wolfgang Mulzer
Zweitgutachter: Prof. Dr. Sergio Cabello
Datum der Disputation: 21.09.2021

Abstract

Given a set S of n point sites, a geometric graph is a graph with S as its vertex set whose edges are drawn as line segments connecting the sites. The edges that are present between these sites can be defined by geometric properties of the site set. When talking about weighted edges, the weight of the edge connecting s and t is the Euclidean distance between s and t. One such class of graphs are spanning trees of the point set. That is, an acyclic graph defined on S such that all sites lie in the same connected component.

To define the second broad class of geometric graphs considered in this thesis, we extend each site with a radius. In this setting the sites can also be interpreted as disks, by using the site as the center. We consider two kinds of geometrically defined graphs on these extended sites. The first are ***disk graphs*** $\mathcal{D}(S)$. In a disk graph two sites are connected with an edge if and only if their corresponding disks intersect. The second type of graphs are ***transmission graphs*** $\mathcal{T}(S)$. These can be seen as a directed version of disk graphs. In a transmission graph a site s has a directed edge to a site t, if and only if t is contained in the disk defined by s.

We consider three main types of problems on these graphs:

Triangles and Girth in Disk Graphs and Transmission Graphs We give algorithms for finding a (shortest) triangle and more generally for finding short cycles. In general graphs, finding substantially faster algorithms than the naive approach is notoriously hard. However, better algorithms for special graph classes such as planar graphs exist in the literature. In this thesis, we obtain similarly efficient results for disk graphs and for transmission graphs. More precisely, we show that in a transmission graph a triangle can be detected and a shortest such triangle can be found in $O(n \log n)$ expected time. Furthermore, the weighted girth of a disk graph can be found within the same time bound. We also show that cycle with k edges in a transmission graph can be identified in $O(n \log n) + n \cdot 2^{O(k)}$ expected time. For the results on transmission graphs, we develop batched range query data structures that are of independent interest.

Dynamic Disk Graph Connectivity We consider the problem of designing data structures that maintain a disk graph under the deletion of sites, while allowing interleaved connectivity queries. First we consider the setting, where each site has a radius in the range $[1, \Psi]$ for some fixed value Ψ. In this scenario, we give a data structure that supports m deletions in $O\Big((n \log^5 n + m \log^9 n)\lambda_6(\log n) + n \log \Psi \log^4 n\Big)$ overall expected time, with $O\Big(\frac{\log n}{\log \log n}\Big)$ query time, where $\lambda_6(n)$ is the length of a Davenport-Schinzel sequence of order 6. If we consider disk graphs without bounding the maximal allowed radius, we

obtain a data structure that supports m deletions in $O\Big(\big(n\log^6 n + m\log^{10} n\big)\lambda_6(\log n)\Big)$ overall expected time, with the same $O\Big(\frac{\log n}{\log\log n}\Big)$ time bound for queries.

Long Plane Trees We also consider spanning trees on the site set S. To be precise, we aim to find a plane spanning tree T_{OPT} of S that maximizes the total edge length $|T_{\text{OPT}}|$. Despite more than two decades of research, it remains open if this problem is NP-hard.

We take two approaches to the problem. The first is to follow the venue of approximation algorithms which was also the focus of previous research. We describe a polynomial-time algorithm to construct a plane tree T_{ALG} with diameter at most four and $|T_{\text{ALG}}| \geq 0.546 \cdot |T_{\text{OPT}}|$, where $|T_{\text{OPT}}|$ is the total edge length of an optimal plane spanning tree. This constitutes a significant improvement over the state of the art. Second, we consider exact polynomial time algorithms for trees of bounded diameter. We give an $O(n^4)$ time algorithm for finding an exact solution for trees of diameter at most three and then extend this algorithm to special trees of diameter at most four.

Selbstständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbstständig und ohne Benutzung anderer als der angegebenen 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.

Berlin, den 30. Juni 2021

Acknowledgments

First of all I would like to thank my advisor Wolfgang Mulzer for making this thesis possible on many levels. The advanced classes taught by him during my masters studies strengthened my interest in theoretical computer science and during my time as a PhD student, he always found a good balance between giving me freedom and guidance in both research and teaching.

I would also like to thank Sergio Cabello for agreeing to be the second reviewer of this thesis, especially considering my challenging time constraints.

I would like to thank all members of the theoretical computer science working group for the pleasant work environment. In particular I thank them for the discussions during lunch and coffee round that were often about more than work. I would also like to thank all people that offered to proofread this thesis.

I also thank the Berlin Mathematical School for providing travel funds, and Stefan Felsner, my mentor in the BMS.

I would like to thank my coauthors, especially those involved in the results that are presented in this thesis: Alexander, Haim, Kristin, Liam, Michael, Paul, Pepa, Sergio and Wolfgang. During my time as a PhD student I had the opportunity to attend several workshops all over the world. I would like to thank the organizers and participants of the research stay in Sendai, and the DACH and TAUFU workshops for creating such a pleasant research environment.

Outside of work, I would like to thank my friends for the occasionally needed distraction from research and teaching, in particular I would like to thank Manuel and Kristin for sharing the ups and downs of research and sometimes life with me.

A special thanks goes to my family and especially my parents for supporting me throughout my entire life. Papa, I wish that I could share this step in my life with you.

Last but not least, I thank Jonas for more things than I could ever list here.

Contents

CHAPTER 1
Introduction

The connection between graphs and geometry goes beyond their common first letter. Among the first problems that are now regarded as graph theoretic problems are the problem of the bridges of Königsberg [Eul41] and the knights path problem [War23]. Both these problems are motivated by geometric descriptions, the map of Königsberg in the former case and the possible fields on a chess board, a knight can visit in the latter case. The first mention of the term *graph* was in an article connecting the valences of atoms to algebraic structures [Syl78], which can also be seen as a geometrically inspired application. Since its origins, the field of graph theory has developed towards the combinatorics of graphs without considering underlying geometry, however geometric concepts are still central to some aspects of it and there is a flourishing field of geometric graph theory [Pac04]. In this thesis we focus on graphs that are defined by geometry. In these graphs, the vertices are defined by a set of point sites in the Euclidean plane and the edges are defined by the geometrical properties of these sites. The edges can be naturally represented by straight line segments between the corresponding sites and their lengths are determined by the lengths of these line segments.

1.1 Thesis Organization

We consider two main types of graphs that are defined on a set of point sites. These graphs and other concepts that are used throughout the thesis, are formally introduced in Chapter 2.

For the first type of graphs, considered in Part I, we augment the sites with a radius to form a disk. We consider the *disk graphs* and *transmission graphs* defined by these disks. In a disk graph, two sites are connected by an edge if and only if their associated disks intersect. Transmission graphs can be seen as a directed version of disk graphs: there is a directed edge between a site s and a site t if t lies in the disk defined by s. Even though these graph classes are often motivated as a simple model for wireless networks [HS95], the study of their properties has developed into an independent field. For both these graphs we aim to design algorithms and data structures that use the underlying geometry to achieve better time bounds than the best currently known bounds for general graphs. In particular, our running times do not depend on the number of edges. We consider two families of problems: algorithms for finding triangles and short cycles, and data structures to dynamically maintain the connectivity information of a graph. In Chapter 3, we develop an algorithm to find the shortest weighted cycle in a disk graph, while in Chapter 4, we focus on transmission graphs and consider the problems of finding a shortest triangle and a short cycle. In Section 4.5, we develop range searching data structures that are used for the algorithms on transmission graphs. These data structures might also be of independent interest. In Chapter 5 we give two decremental connectivity data structures for disk graphs, one for the case of a bounded maximum radius and one for general disk graphs.

The second type of graphs are spanning trees on a given site set. In Part II, we focus on spanning trees of a site set whose embedded edges do not cross and whose total edge length is maximized. As finding an exact solution to this problem is conjectured to be NP-hard, we focus on approximation algorithms in Chapter 6 and on exact polynomial time algorithms for special cases in Chapter 7. The thesis closes with some concluding remarks in Chapter 8.

1.2 Girth and Triangles in Disk Graphs and Transmission Graphs

Given a graph G with n vertices and m edges, the question if G contains a triangle is one of the most basic algorithmic questions in graph theory, and many other problems reduce to it [IR78; WW18]. The best known algorithms use fast matrix multiplication and run in either $O(n^\omega)$ time or in $O(m^{2\omega/(\omega+1)})$ time, where $\omega < 2.37287$ is the matrix multiplication exponent [AYZ97; IR78; Le 14]. Despite decades of research, the best available "combinatorial" algorithm[1] needs $O(n^3 \operatorname{polyloglog}(n)/\log^4 n)$ time [Yu15], which is only slightly better than checking all vertex triples. This lack of progress can be explained by a connection to Boolean matrix multiplication: if there is a truly subcubic combinatorial algorithm for finding triangles, there is also a truly subcubic combinatorial algorithm for Boolean matrix multiplication [WW18]. Itai and Rodeh [IR78] reduced computing the *girth*, that is the length of the shortest cycle, of an unweighted undirected graph to triangle detection. For integer edge weights, Roditty and Williams [RW11] gave an equivalence between finding a minimum weight cycle (the weighted girth) and finding a minimum weighted triangle.

For the special case of *planar* graphs, significantly better algorithms are known. Itai and Rodeh [IR78] and, independently, Papadimitriou and Yannakakis [PY81] showed that a triangle can be found in $O(n)$ time, if it exists. Chang and Lu [CL13] presented an $O(n)$ time algorithm for computing the unweighted girth. The weighted girth can be found in $O(n \log\log n)$ time in both undirected and directed planar graphs [CL13; ŁS11].

Motivated by the vastly better algorithms for planar graphs, triangle detection and girth computation in disk graphs was considered by Kaplan et al. [Kap+19; Kap+17]. They show that a triangle can be found in $O(n \log n)$ time, using a simple geometric observation to relate disk graphs and planar graphs. By the same geometric observation it

[1] An algorithm is "combinatorial" if it does not need algebraic manipulations to achieve its goal.

is also possible to compute the unweighted girth in a disk graph in $O(n \log n)$ time. Their method generalizes to finding a shortest triangle in a weighted disk graph in $O(n \log n)$ expected time. By a reduction from ε-CLOSENESS [Pol17], all these bounds are optimal in the algebraic decision tree model, a contrast to planar graphs, where $O(n)$ time is possible.

Results In this thesis, we extend the approach of Kaplan et al. to the weighted girth. More specifically, in Section 3.2 we present an algorithm that computes the weighted girth of a disk graph in $O(n \log n)$ expected time. For this result we also describe a method to find a cycle that contains a given vertex in an abstract graph.

Furthermore, we consider the same problems in transmission graphs. In Section 4.1, we first show that the favorable property connecting triangle-free disk graphs to planar graphs does not extend to transmission graphs. Thus we consider the geometric properties of directed triangles in transmission graphs and in Section 4.2 develop an algorithm to identify such triangles in $O(n \log n)$ expected time. We extend this result to the weighted version in Section 4.3, using similar ideas as Kaplan et al. [Kap+19]. In Section 4.4, we give an algorithm that finds a cycle with at most k edges in $O(n \log n) + n \cdot 2^{O(k)}$ expected time. For all these algorithms, we develop several new techniques for batched range searching, using linearized quadtrees and three-dimensional polytopes to test for containment in the union of planar disks. The range query data structures are described in Section 4.5.

1.3 Dynamic connectivity

Preprocessing a graph G in such a way that one can efficiently determine if two vertices lie in the same connected component is another fundamental problem in algorithmic graph theory. If the graph is static, using breadth first search or depth first search to identify all connected components allows to answer these queries in $O(1)$ time after a linear preprocessing time. If the graph is dynamic, the situation is more complicated. In such a dynamic graph the connectivity queries are interleaved with update operations on the graph. These updates can be the insertion or deletion of edges or vertices. When having a fixed set of vertices and allowing the insertion of edges only, a disjoint-set data structure solves the problem. Such a data structure can be implemented efficiently to achieve amortized time of $O(1)$ for updates and $O(\alpha(n))$ for queries, where $\alpha(n)$ is the inverse Ackermann function [SS05].

If both edge insertions and deletions are allowed, Holm et al. [HdLT01] give the fastest currently known data structure. The data structure achieves $O\left(\frac{\log n}{\log \log n}\right)$ query time and $O(\log^2 n)$ amortized time for edge updates. If the underlying graph is planar, there is a data structure by Eppstein et al. [Epp+92] with $O(\log n)$ amortized time for both queries and updates. If the vertex set is considered to be dynamic, this is done by having a fixed base set of m edges and activating or deactivating vertices and their incident edges. In this setting, there is a data structure allowing vertex activation and deactivation with

$O(m^{1/3}\,\mathrm{polylog}(n))$ amortized query and $O(m^{2/3}\,\mathrm{polylog}(n))$ amortized update time by Chan et al. [CPR11].

In Chapter 5, we study the dynamic connectivity problem on disk graphs. Note that while the disk graph can be described by specifying the n sites, it might have $\Theta(n^2)$ edges. We consider two variants of disk graphs, based on the possible values for the radii. In the first variant, the ratio Ψ between the largest and the smallest radius is bounded. The second variant does not impose any restriction on the radii.

We assume that the site set S is dynamic in the sense that sites can be inserted and deleted. At each site insertion or deletion, the edges incident to the updated site appear or disappear in $\mathcal{D}(S)$. As the degree in $\mathcal{D}(S)$ is not bounded, each update can then lead to $O(n)$ edges changing in $\mathcal{D}(S)$. Thus, simply storing $\mathcal{D}(S)$ in a Holm et al. data structure would lead to potentially superlinear update times and would be even slower than recomputing the connectivity information from scratch.

For disk graphs with arbitrary radius ratio, Chan et al. [CPR11] give a data structure with amortized $O(n^{(1/7)+\varepsilon})$ query and $O(n^{(20/21)+\varepsilon})$ update time for solving the dynamic connectivity problem. Their approach uses similar ideas as their vertex update data structure. This is still the best currently known connectivity data structure that allows insertions and deletions for arbitrary disk graphs. However, their data structure handles a more general setting, so there is hope that using the specific geometry of disk graphs will lead to better results.

Indeed, there is a series of results on disk graphs that achieve polylogarithmic update and query times. For unit disk graphs, Chan et al. [CPR11] observe that there is a data structure with $O(\log^{10} n)$ update and $O\left(\frac{\log n}{\log\log n}\right)$ query time. This time bound was then improved by Kaplan et al. [Kap+21a] to $O(\log^2 n)$ amortized update and $O\left(\frac{\log n}{\log\log n}\right)$ amortized query time. In the case of bounded radius ratio, they showed that, using bichromatic maximal matchings, there is a data structure with an expected amortized $O(\Psi^2 2^{\alpha(n)} \log^{10} n)$ update and $O\left(\frac{\log n}{\log\log n}\right)$ query time. They also give an algorithm with an improved expected amortized update time of $O(\Psi 2^{\alpha(n)} \log^{10} n)$, at the cost of an increased $O(\log n)$ query time. All these data structures have in common, that they define a proxy graph with a bounded number of edges. This proxy graph represents the connectivity of the disk graph and can be updated efficiently when sites are inserted and deleted, by updating suitable auxiliary dynamic geometric data structures. To answer queries the graph is additionally stored in a dynamic connectivity data structure for general graphs. The queries are then performed on this connectivity data structure, sometimes encapsulated in a geometric preprocessing step for each query. The time bounds for updates and queries are then a combination of the time bounds for the additional geometric data structures and the dynamic connectivity data structures.

Results In this thesis, we only consider the *decremental setting* in disk graphs. In this setting, sites are only deleted from the set and never inserted. In the setting of bounded radius ratio, in Theorem 5.9 we show that we can obtain a logarithmic dependency on Ψ for the deletion of a site, with $O\left(\frac{\log n}{\log\log n}\right)$ query time. This can be achieved by defining a suitable proxy graph based on a set of quadtrees on the set S. The proxy graph is then stored in a Holm et al. [HdLT01] dynamic connectivity data structure. In order to update the proxy graph in the Holm et al. data structure, the main challenge is to identify the edges that have to be updated because the site was deleted. There is a data structure by Kaplan et al. [Kap+21a; KKM21] that we will use to solve this problem. The exact result presented in this thesis is a data structure on which we can perform m updates in overall $O((n\log^5 n + m\log^9 n)\lambda_6(\log n) + n\log\Psi\log^4 n)$ expected time, with a query time of $O\left(\frac{\log n}{\log\log n}\right)$.

In Section 5.2 we extend this approach to the case of general disk graphs. The dependency on Ψ in the approach for the bounded radius ratio resulted from the height of the quadtrees. By using a compressed quadtree in combination with a heavy path decomposition and binary search trees, we obtain a data structure with a running time independent of Ψ. The data structure described in Theorem 5.16 allows deletions in $O((n\log^6 n + m\log^{10} n)\lambda_6(\log n))$ overall expected time, with the same query time as in the case of bounded radius ratio.

1.4 Long plane spanning trees

Geometric network design is a common and well-studied task in computational geometry and combinatorial optimization [Epp00; Har11; Mit17; MM17]. In this family of problems, we are given a set S of point sites in general position, and our task is to connect S into a geometric graph that has certain favorable properties. Not surprisingly, this general question has captivated the attention of researchers for a long time. We can find countless variants, depending on which restrictions we put on the graph that connects S and which criteria of this graph we would like to optimize. Typical graph classes of interest include matchings, paths, cycles, trees, or general ***plane*** (*noncrossing*) graphs, that is graphs, of which the straight-line embedding on S does not contain any edge crossings. Typical quality criteria include the total edge length [Aro98; dBer+08; Mit99; MR08], the maximum length edge [Bin20a; EIK01], the maximum degree [AC04; Cha04; FH09; PV84], the dilation [Epp00; Mul04; NS07], or the stabbing number [MO20; Wel92] of the graph.

Many famous problems from computational geometry fall into this general setting. For example, if our goal is to minimize the total edge length, while restricting ourselves to paths, trees, or triangulations, respectively, we are faced with the problems of finding an optimum TSP tour [Har11], a Euclidean minimum spanning tree [dBer+08], or a minimum weight triangulation [MR08] for S. These three examples also illustrate the wide variety of complexity aspects that we may encounter in geometric design problems: the Euclidean TSP problem is known to be NP-hard [Pap77], but it admits a PTAS [Aro98; Mit99]. On the other hand, it is possible to find an Euclidean minimum spanning tree for S in polynomial time [dBer+08], even though, the associated decision problem is not known to be solvable by a polynomial-time Turing machine. The minimum weight triangulation problem is also known to be NP-hard [MR08], but the existence of a PTAS is still open; however, a QPTAS is known [RS09].

In Part II of this thesis, we are interested in the interaction of two specific requirements for a geometric design problem, namely the two desires of obtaining a plane graph and of optimizing the total edge length. In case we want to *minimize* the total edge length of the resulting graph, these two goals are often in perfect harmony: the shortest Euclidean TSP tour and the shortest Euclidean minimum spanning tree are automatically plane, as can be seen by a simple application of the triangle inequality. In contrast, if our goal is to *maximize* the total edge length, while obtaining a plane graph, much less is known.

This family of problems was studied by Alon et al. [ARS95], who considered the problems of computing a longest plane matching, a longest plane Hamiltonian path, and a longest plane spanning tree for a planar point set S in general position. They conjectured that these three problems are all NP-hard, but this question is still open. The situation is similar for the problem of finding a *maximum weight triangulation* for S: here, we have neither an NP-hardness proof, nor a polynomial time algorithm [QW06]. If we omit the planarity condition, then the problem of finding a longest Hamiltonian path (the *geometric maximum TSP problem*) is known to be NP-hard in dimension three and above, while the two-dimensional case remains open [Bar+03]. On the other hand, we can find a longest, typically not plane, tree on S in polynomial time, using classic greedy algorithms [Cor+09].

We focus on the specific problem of finding a longest plane (that is non-crossing) tree for a given set S of $n \geq 3$ point sites in the plane. As already mentioned above, the problem is conjectured to be NP-hard for general site sets [ARS95]. For this reason, past research has focused on designing polynomial-time approximation algorithms. Typically, these algorithms proceed by constructing several “simple” spanning trees for S of small diameter and by arguing that at least one of these trees is sufficiently long. One of the first corresponding results was published by Alon et al. [ARS95]. They showed that a longest star, a plane tree with diameter two, on S yields a 0.5-approximation for the longest, not necessarily plane, spanning tree of S. They also argued that this bound is essentially tight for stars, by considering site sets that consist of two large clusters far away from each other. Dumitrescu and Tóth [DT10] refined this algorithm by adding two additional families of candidate trees, now with diameter four. They showed that at least one member of this extended range of candidates constitutes a 0.502-approximation, which was further improved to 0.503 by Biniaz et al. [Bin+19]. In all these results, the approximation factor is analyzed by comparing the tree that results from the algorithm with the length of a longest, typically not plane, spanning tree. Such a tree may be up to $\pi/2 > 1.5$ times longer than a maximum length plane tree [ARS95], as, for instance, witnessed by large point sets spaced uniformly over a circle. By comparing against a longest *plane* tree, and by considering some trees with diameter five in addition to stars, the approximation factor was pushed to 0.512 [Cab+20]. This was subsequently improved even further to 0.519 [Bin20b].

Results In Chapter 6 we focus on approximating the longest plane spanning tree. We give a polynomial-time algorithm that iterates all candidates for the longest edge of a longest plane tree. For each candidate edge it constructs two trees that contain the edges and have diameter at most four. Additionally, it considers all possible stars, that is trees where all sites are connected to a singe site. We argue that when considering the actual longest edge in the optimal plane spanning tree, one of five stars or one of the two special trees is long enough. As a warm up, we show in Theorem 6.2 that the tree returned by this algorithm is a $\frac{2}{3}$-approximation, when considering site sets in which the y-coordinates are neglectable. We also show that the $\frac{2}{3}$ bound for the restricted set of sites is tight, when comparing to the longest, possibly crossing, spanning tree. In Theorem 6.13 we show that on general site sets the algorithm yields a 0.546-approximation. This is a substantial improvement over the previous bounds. As all trees considered for approximation in the literature so far have small diameter, we also consider upper bounds on the quality of such algorithms. More precisely, in Theorem 6.14, we construct a site set such that the longest plane spanning tree of diameter at most three cannot give an approximation factor of more than $\frac{5}{6}$.

Chapter 7 focuses on exact polynomial time algorithms for finding the longest plane spanning tree. A natural way to design an algorithm for the longest plane spanning tree problem is the following local search heuristic [WS11]: start with an arbitrary plane tree T, and as long as possible, exchange an existing edge by a new longer edge such that the resulting graph tree is still a plane spanning tree. Once no further local improvements are possible, output the current tree T. We show in Lemma 7.1 that there is a site set, for that this algorithm fails to compute the optimum answer as it may get stuck in a local optimum. This suggests that a natural local search approach does not yield an optimal algorithm for the problem.

However there are polynomial time algorithms for special cases. If S is in convex position, we show in Section 7.2 that the longest plane tree for S can be found in $O(n^3)$ time by adapting standard dynamic programming methods for plane structures on convex point sets [Gil79; Kli80]. In Theorem 7.5 we present an $O(n^4)$ time dynamic programming algorithm for finding a longest plane tree among those of diameter at most three. In Theorem 7.6 we extend this approach to trees, where all sites are connected to one of three fixed sites on the convex hull. The running time increases to $O(n^6)$.

1.5 Publications

The results presented in this thesis are either novel, or were already published in the following publications:

[Kap+19] Haim Kaplan, Katharina Klost, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. “Triangles and Girth in Disk Graphs and Transmission Graphs”. In: *27th Annual European Symposium on Algorithms (ESA 2019)*. Ed. by Michael A. Bender, Ola Svensson, and Grzegorz Herman. Vol. 144. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 64:1–64:14. 2019

[Cab+21] Sergio Cabello, Michael Hoffmann, Katharina Klost, Wolfgang Mulzer, and Josef Tkadlec. *Long Plane Trees.* arXiv: `2101.00445 [cs]`. URL: `http://arxiv.org/abs/2101.00445`. 2021

[Kap+21a] Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. *Dynamic Connectivity in Disk Graphs.* arXiv: `2106.14935 [cs]`. URL: `http://arxiv.org/abs/2106.14935`. 2021

CHAPTER 2
Preliminaries

In this chapter, we define the notions and concepts used in this thesis. We assume a basic familiarity of the reader with elemental mathematical and in particular geometric concepts. Additionally, we assume knowledge about the analysis of algorithms. However, to introduce the notation used for some elemental concepts, we repeat some of the definitions below.

2.1 Geometry, Graphs and Geometric Graphs

Geometry Most of the geometry considered in this thesis takes place in $\mathbb{R}^2$. Let $p = (p_x, p_y) \in \mathbb{R}^2$ be the point in the plane with x-coordinate p_x and y-coordinate p_y. Given two points p and q in $\mathbb{R}^2$, let $\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$ be the Euclidean distance between p and q. For a subset $Q \subseteq \mathbb{R}^2$, let the diameter be defined as $\operatorname{diam}(Q) = \sup\{\|pq\| \mid p, q \in Q\}$. Given a point p and a radius r we denote by $D(p,r)$ the disk of radius r with center p. When talking explicitly about the circle bounding a disk, we denote it by $\partial D(p,r)$. More formally, we define disks and circles as follows:

$$D(p,r) = \{q \in \mathbb{R}^2 \mid \|pq\| \leq r\} \qquad \partial D(p,r) = \{q \in \mathbb{R}^2 \mid \|pq\| = r\}$$

Given two points p, q we denote the line segment defined by these two points by $\overline{pq} = \{p + \lambda(q-p) \mid 0 \leq \lambda \leq 1\}$. Let $\overrightarrow{pq} = \{p + \lambda(q-p) \mid \lambda \geq 0\}$ be the ray originating in p and going through q. We call the ray $\{p - \lambda(q-p) \mid \lambda \geq 0\}$ starting in p with the same supporting line as $\overrightarrow{pq}$ that only has p in common with $\overrightarrow{pq}$ the ray opposite to $\overrightarrow{pq}$.

When talking about geometry in $\mathbb{R}^3$, we denote a point by $p = (p_x, p_y, p_z)$. Given a point set P in $\mathbb{R}^2$ or $\mathbb{R}^3$, the *convex hull* of this point set is the minimum convex set containing P [dBer+08]. We will also consider *halfplanes* in $\mathbb{R}^3$. For fixed constants $a, b, c \in \mathbb{R}$ let a halfplane be the set $H = \{p \in \mathbb{R}^3 \mid p_z = ap_x + bp_y + c\}$. For a more compact representation, we also write $H : z = ax + by + c$ to define a halfplane H. The closed upper halfspace bounded by a halfplane H is defined as $\{p \in \mathbb{R}^3 \mid p_z \geq ap_x + bp_y + c\}$ where a, b, c are the coefficients defining H. In some proofs we consider *convex polyhedra*. We will consider two different definition of such polyhedra. The first definition is that of the intersection of finitely many closed halfspaces. The resulting polyhedron might be unbounded. The second definition is that of the convex hull of finitely many points. This definition will always result in a bounded polyhedron, which we will also call a *convex polytope.*

Graphs We revisit the definition of a general or abstract *graph.* A graph G is defined as a pair consisting of a *vertex set* $V(G)$ and an *edge set* $E(G)$. We consider both directed and undirected graphs. For an *undirected graph* we have $E(G) \subseteq \binom{V(G)}{2}$, or put differently the members of $E(G)$ are sets of size two. We denote an undirected edge as $e = \{u,v\}$. In the case of *directed graphs*, the members of $E(G)$ are ordered tuples, so $E(G) \subseteq (V(G) \times V(G)) \setminus \{(v,v) \mid v \in V(G)\}$. Deviating from the natural tuple notation, we will denote a directed edge as $e = uv$. These definitions of the edge sets imply that all graphs are *simple* i.e. they do not contain loops or duplicate edges.

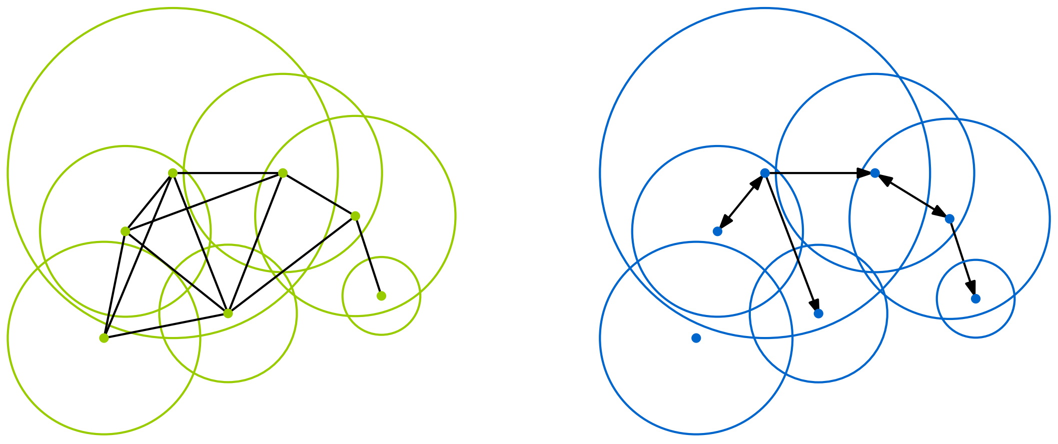
A sequence $\pi = v_0,\dots,v_{k-1}$ of unique vertices is called a *path*, if for $i = \{0,\dots,k-2\}$, v_i can reach v_{i+1} with an edge. A sequence $C = v_0,\dots,v_{k-1}$ where v_i can reach v_{i+1} if the indices are considered modulo k is called a *cycle* in G. Two vertices u and v are *connected*, if there is a path π with $v_0 = u$ and $v_{k-1} = v$.

In a *weighted graph* there is a function $w : E(G) \to \mathbb{R}$ that assigns a weight to each of the edges. For this thesis we will assume all edge weights to be positive. The unweighted length of a path or a cycle is the number of edges on the path or cycle, while the weighted length is the sum of the edge lengths. Let $d(u,v)$ be the (un)weighted length of a shortest path between u and v. We define the (un)weighted diameter $\operatorname{diam}(G)$ of a graph as $\max_{(u,v)\in V\times V} d(u,v)$. Furthermore, let the *unweighted girth* of a graph be the length of the shortest unweighted cycle and the *weighted girth* the length of the shortest weighted cycle.

Geometric Graphs A *geometric graph* is a graph, where each vertex is mapped to a point in $\mathbb{R}^2$. An edge connecting vertices u and v is embedded as the line segment $\overline{uv}$ connecting u and v [Fel04; Pac13]. In the following, we will not distinguish between the vertices and the points they are mapped to. Similarly, we often do not specifically distinguish between an edge $\{u,v\}$ or uv and its associated line segment. In general, these line segments might intersect. If for a given abstract graph, there is an embedding as a geometric graph, where no two line segments intersect, we call the graph *planar.* A geometric graph where no two edges intersect in their interior is called a *plane graph.* Note that being planar is a property of the abstract graph, while being plane is a property of the given embedding.

For the geometric graphs considered in this thesis, we always assume that the set of points that are identified with the vertices is given as part of the input. The graph is then defined based on different geometric properties of these points. In order to distinguish the input points from other points in the plane, we call the set S of n input points *sites*, and only refer to points not in S as *points.* When talking about weighted geometric graphs, the weight of edge connecting sites s and t is the Euclidean length $\|st\|$. We now define the types of geometric graphs considered in this thesis.

Disk Graphs and Transmission Graphs The first two types of geometric graphs are closely related to each other. For both *disk graphs* and *transmission graphs* we associate



(a) The disk graph $\mathcal{D}(S)$ (b) The transmission graph $\mathcal{T}(S)$

Figure 2.1: A set of sites with the associated disks.

a radius $r_s \in \mathbb{R}, r_s > 0$ with each site s. This radius together with the site defines a disk $D_s = D(s, r_s)$ in a natural way. When it is clear from the context, we will use s and D_s interchangeably. We consider both disk graphs and transmission graphs as geometric graphs as defined above. Thus the vertex set of both the disk graph $\mathcal{D}(S)$ and the transmission graph $\mathcal{T}(S)$ is S. In both cases the edges are defined based on the relation of the distance between two sites and their associated radii.

In a disk graph the edges are undirected and there is an edge $\{s,t\}$ in the disk graph, if and only if the associated disks D_s and D_t intersect. Equivalently, we can say that s and t are connected by an edge if and only if $\|st\| \leq r_s + r_t$. The edges in the transmission graph $\mathcal{T}(S)$ are directed. There is an directed edge st in the transmission graph if and only if $t \in D_s$, or equivalently, if $\|st\| \leq r_s$. For disk graphs and transmission graphs, we will not only consider their representation as geometric graphs, but in some cases also the abstract graph defined by the set of sites. See Figure 2.1 for an illustration of these two types of graphs.

For a given disk graph or transmission graph we denote by Ψ the ratio between the largest and the smallest radius of the sites, $\Psi = \frac{\max_{s \in S} r_s}{\min_{t \in S} r_t}$. To ease some of the calculations and arguments in the following, we assume without loss of generality that S satisfies the following properties:

(a) The radius associated to the smallest site $s \in S$ is 1,

(b) S fits into a single axis parallel square with the origin as a vertex and diameter $2^{\lceil \log \operatorname{diam}(S) \rceil} \leq 2 \operatorname{diam}(S)$,

(c) the maximum radius of a site is at most $2 \operatorname{diam}(S)$; and

(d) no two sites have the same radius.

All disk graphs and transmission graphs can be represented by a set of sites satisfying these conditions. The first and second conditions can be achieved by scaling, translating

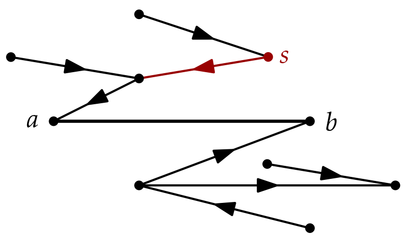


Figure 2.2: All edges are directed towards $\{a, b\}$. The edge associated to s is marked blue.

and rotating the point set and radii. For condition (c), first consider the set of sites after scaling and translating. Setting $r_s = \min\{r_s, 2\operatorname{diam}(S)\}$ satisfies condition (c) and does not remove any possible edges: all disks with radius $2\operatorname{diam}(S)$ contain the whole square defined by the second condition and thus are connected to all other sites. Conditions (a) directly implies $\Psi = \max_{s\in S} r_s$.

The last condition holds without loss of generality by perturbing the radii. To see this, note that each radius can be increased by any amount less than half the smallest distance between its boundary and the boundary of any other disk without introducing additional intersections. As increasing the radius cannot remove existing edges, this does not change the structure of the underlying graph. This construction is similar to the argument of Gräf et al. [GSW98] which allows the radii of the disks in a unit disk graph to be increased by a small ε without changing the underlying graph. We only need this property in some applications, in which it will be made clear that this assumption is used.

Spanning Trees The third type of geometric graphs are *spanning trees* of a set of sites. A spanning tree is a minimal set of undirected edges, where all sites are pairwise connected. It is a well known fact, that the spanning tree on n sites has $n-1$ edges. For spanning trees we assume that the points lie in general position, in the sense that no three points are collinear. The general position assumption was also used in previous work on this problem [ARS95; DT10]. Without it, one should specify whether overlapping edges are allowed, an additional complication that we would like to avoid. Let $T(S)$ be a spanning tree on a given set of sites. If the site set considered is clear from the context, we will simply write T instead of $T(S)$. We then define the length of T as the sum of the weights of the edges T and denote it by $|T|$. A very simple case of a spanning tree is the *star* T_a rooted at the site a. In T_a all sites in $S \setminus \{a\}$ are connected with an edge to a.

Given multiple spanning trees, that all contain a common edge $\{a, b\}$, in Chapter 6, we will need a way to compare the weights of the trees, by considering the edges separately. For this we direct all edges towards the edge $\{a, b\}$ and assign each site its unique outgoing edge. Denote the length of the edge assigned to $s \neq a, b$ in a fixed tree T by $\ell_T(s)$. The edge $\{a, b\}$ remains undirected, see Figure 2.2. Then the length of the total tree can be written as $|T| = \left(\sum_{s\in S, s\neq a,b} \ell_T(s)\right) + \|ab\|$.

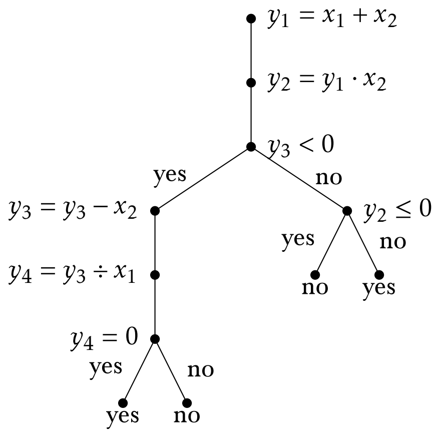


Figure 2.3: Example of an algebraic decision tree of height four.

2.2 Computational Models

Now that we have described the geometric objects considered in this thesis, we describe the computational models used. The main model of computation used in this thesis is the model of the *real RAM* as defined by Preparata and Shamos [PS85]. The real RAM is often seen as the standard model for computation in the setting of computational geometry. It is based on the standard (integer) RAM model but instead of storing only integers, we are allowed to store a single real number in each cell of the RAM. The basic set of operations consists of $\{+, -, \times, \div\}$ combined with comparisons between numbers as well as indirect addressing. Depending on the application, additional operations such as the k-th root, exponential functions and logarithms are added. For this thesis, we add two additional operations which extend the definition of Preparata and Shamos. The first is the floor function $\lfloor \cdot \rfloor$. The second operation is the function $\mathrm{msb}(x, y)$ that yields the most significant bit in which two positive integers x and y differ in binary representation. We assume that all operations take unit cost.

There is a lot of work regarding the power of the unit cost integer RAM when allowing multiplications [Sch79; vEmd91]. Essentially, allowing multiplication in combination with arbitrarily large cells allows to pack multiple values into a single cell and perform operations in parallel. This can then be used to break well-known lower bounds on the complexity of certain problems. Thus, when working in the RAM model while assuming unit cost, one has to be careful not to abuse this power. As an integer RAM can be trivially simulated in a real RAM when using the floor function, all these caveats carry over to our model of computation. However, there are some good practices which allow us to use these additional operations without exploiting the power of this model. The essence of them is to make sure that multiplications, rounding and subsequent bit-operations are only applied to the input or a scaled version of the input. The only place where we use the floor function and $\mathrm{msb}(x, y)$ is in the context of quadtrees. We will not go into detail here, but we only use the floor function to round a scaled version of the input points to the integer grid, and apply the $\mathrm{msb}(x, y)$ function to these rounded points. This way we do not use the power of parallel computing the real RAM with the floor function gives us, and stay within a model of computation that can be considered to be “reasonable”.

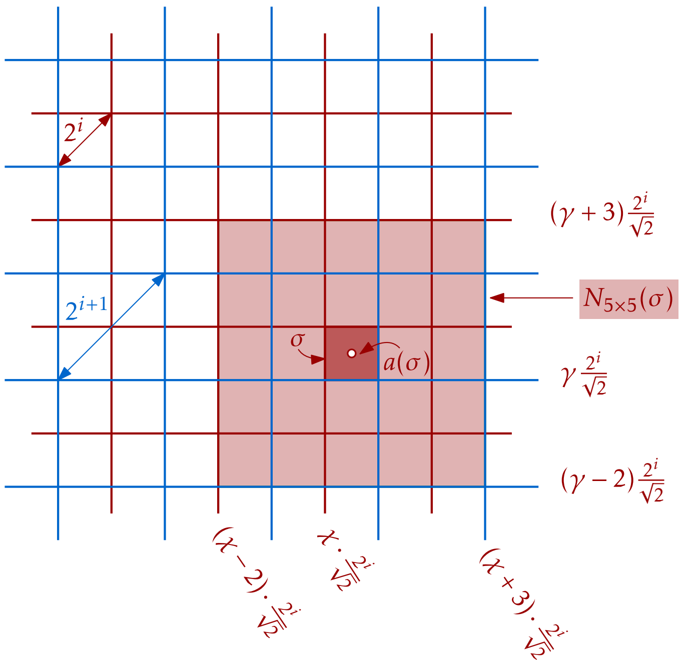
The model used for giving the lower bound in Section 3.1 is the *algebraic decision tree*. This model is closely related to the model of the real RAM in the sense, that every algorithm in the real RAM model can be described as an algebraic decision tree. The following description of this model follows Arora and Barak [AB09, Section 16.2]. An algebraic decision tree is a way of representing a function deciding a decision problem with input $(x_1,\dots,x_n) \in \mathbb{R}^n$ as a binary tree. There are two kinds of inner vertices in the tree. The first kind is a computation node, in which a new variable is defined by applying one operator of $\{+,-,\times,\div,\sqrt{\cdot}\}$ to input values or existing variables. Computation nodes only have one outgoing edge. The second kind of vertices are branching vertices with two children. In those a given variable is compared to zero with one of $\{<,\leq,=\}$, and the edge to each child is associated with a possible outcome. Each leaf is labeled either *yes* or *no* and a given instance of the problem is decided by following the path from the root to a leaf in the obvious way, see Figure 2.3 for an example. The mapping of input instances in $\mathbb{R}^n$ to 0 or 1, depending on whether the answer to the instance is *no* or *yes*, yields a partition of $\mathbb{R}^n$. The complexity of a decision problem in this model is then determined by the minimum height of any algebraic decision tree deciding the problem. For this height, computation nodes that apply a $+$ or $-$ operation are not counted. The lower bounds given in this model are based on a result by Ben-Or [Ben83] which bounds the height of the algebraic decision tree based on the number of connected components defined by either the *yes* or the *no* instances.

2.3 Hierarchical Grids, Z-order and Compressed Quadtrees.

Given some $i \geq 0$ we define the *grid cells* at level i to be all regions, bounded by the halfspaces

$$\begin{aligned} &x \geq \chi \cdot \frac{2^i}{\sqrt{2}}, \qquad && x < (\chi+1)\cdot\frac{2^i}{\sqrt{2}} \qquad && \text{for } \chi \in \mathbb{Z} \text{ and} \\ &y \geq \gamma \cdot \frac{2^i}{\sqrt{2}}, && y < (\gamma+1)\cdot\frac{2^i}{\sqrt{2}} && \text{for } \gamma \in \mathbb{Z}. \end{aligned}$$

Note that the grid cells in each level partition the plane $\mathbb{R}^2$ and that each cell at level i has diameter 2^i. We call the union of all grid cells at level i the grid $\mathcal{G}_i$. On every level, the lower left corner of the cell defined by $\chi = \gamma = 0$ is the origin. Each grid cell in $\mathcal{G}_i$ can be uniquely indexed by χ and γ. Given a point $(p_x, p_y) \in \mathbb{R}^2$, we can determine the grid cell it belongs to by computing the tuple $\left(\left\lfloor \frac{\sqrt{2}\cdot p_x}{2^i} \right\rfloor, \left\lfloor \frac{\sqrt{2}\cdot p_y}{2^i} \right\rfloor\right)$ in $O(1)$ time. As the cells are consistent between the levels, we can scale the indices χ, γ on level i to be consistent with the indices on level 0. This allows us to assume that the cells are defined on the integer grid when doing arithmetics on the grid cells.



2^i
2^{i+1}
$(\gamma+3)\frac{2^i}{\sqrt{2}}$
$N_{5\times5}(\sigma)$
σ
$a(\sigma)$
$\gamma\frac{2^i}{\sqrt{2}}$
$(\gamma-2)\frac{2^i}{\sqrt{2}}$
$(\chi-2)\cdot\frac{2^i}{\sqrt{2}}$
$\chi\cdot\frac{2^i}{\sqrt{2}}$
$(\chi+3)\cdot\frac{2^i}{\sqrt{2}}$

Figure 2.4: Two levels of the hierarchical grid.

The (infinite) *hierarchical grid* $\mathcal{G}$ is then defined as $\bigcup_{i=0}^{\infty}\mathcal{G}_i$. For any cell σ from the hierarchical grid, denote by $|\sigma|$ its diameter and by $a(\sigma)$ its center. Furthermore, for a given cell $\sigma \in \mathcal{G}_i$ identified by a tuple (χ, γ) and an odd number $k \in \mathbb{N}$, let $N_{k\times k}$ be the $k \times k$ subgrid of $\mathcal{G}_i$ which contains the cells (χ', γ') with $\chi - \left\lfloor \frac{k}{2} \right\rfloor \leq \chi' \leq \chi + \left\lceil \frac{k}{2} \right\rceil$ and $\gamma - \left\lfloor \frac{k}{2} \right\rfloor \leq \gamma' \leq \gamma + \left\lceil \frac{k}{2} \right\rceil$. We also call $N_{k\times k}(\sigma)$ a ***neighborhood*** of σ. See Figure 2.4 for a depiction of these concepts.

If a site s is contained in a cell σ of some level of the hierarchical grid, there is a smallest neighborhood of cells around σ, such that a disk with center in s is completely contained in the union of the cells in the neighborhood. The following lemma makes the size of this neighborhood explicit.

Lemma 2.1. *Let $p \in \mathbb{R}^2$ be a point and σ be a cell of some grid with $p \in \sigma$. Then the disk $D(p, r)$ is completely contained in the $\left(2\left\lceil \frac{\sqrt{2}r}{|\sigma|} \right\rceil + 1\right) \times \left(2\left\lceil \frac{\sqrt{2}r}{|\sigma|} \right\rceil + 1\right)$ neighborhood N of σ.*

Proof. We are going to show a slightly stronger result, namely that the enclosing square of $D(p, r)$ is completely contained in N. The diameter of this square is $2\sqrt{2}r$ and the union of all enclosing squares for possible positions of p is another square ρ with center $a(\sigma)$; see Figure 2.5. The size of the neighborhood then is the same as the number of cells intersected by a diagonal of ρ. To bound this number, we divide the diagonal of ρ into the part inside of σ and the two disconnected parts outside of σ. Then the part inside of σ only passes through σ. The two parts on the outside pass through $\left\lceil \frac{\sqrt{2}r}{|\sigma|} \right\rceil$ cells each and thus $D(p, r)$ is completely contained in N. □

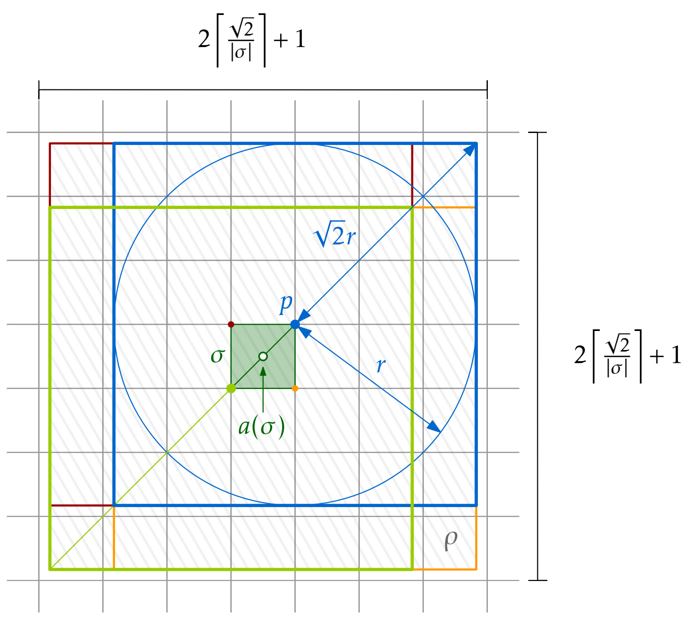
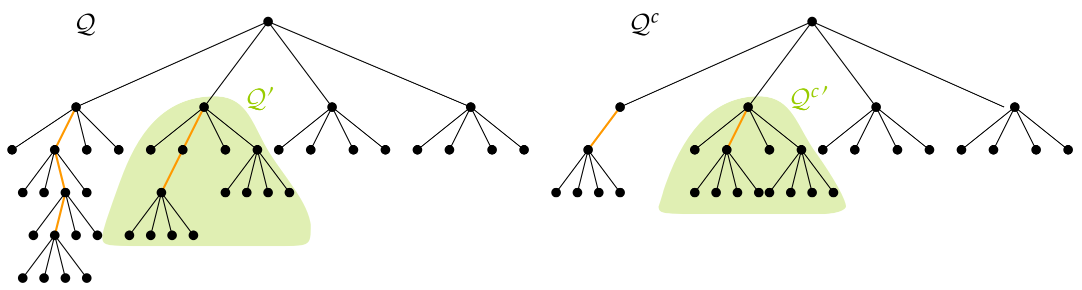

$2\left\lceil \frac{\sqrt{2}}{|\sigma|} \right\rceil + 1$
$\sqrt{2}r$
p
σ
r
$a(\sigma)$
ρ
$2\left\lceil \frac{\sqrt{2}}{|\sigma|} \right\rceil + 1$

Figure 2.5: The disk $D(p, r)$ is contained in N. The square ρ is marked gray and the diagonal is divided into the light green, dark green and blue part.

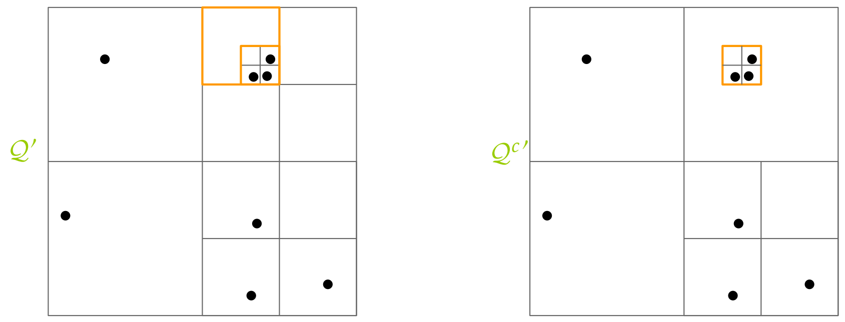
The *quadtree* $\mathcal{Q}$ on S is a rooted tree whose nodes are a subset of cells from the hierarchical grid. The root is the cell with the smallest diameter that contains all sites of S. If a cell σ with $|\sigma| = 2^i$ for $i \geq 0$ contains at least two sites of S, then the children of σ are the four cells τ with $|\tau| = 2^{i-1}$ and $\tau \subseteq \sigma$ that partition σ. If a cell σ contains only one site of S, it does not have any children. By our assumption on S, the root has diameter $\Theta(\text{diam}(S))$. Note that as we do not continue building the quadtree below level 0 the leafs can possibly contain more than one site. In many cases, we will not explicitly distinguish between a cell σ and its associated vertex in the quadtree.

If we define a quadtree in this way, it has $O(n)$ leaves and height at most $\lceil \log(\text{diam}(S)) \rceil$. This height does not depend on n and can be arbitrarily large. However in some applications we would like to have a dependence only on n. To avoid the dependency on the diameter of the point set, we define the *compressed quadtree* $\mathcal{Q}^c$. Let $\sigma_1, \ldots, \sigma_k$ be a maximal path in $\mathcal{Q}$, where all σ_i, $1 \leq i < k$, have only one non-empty child. In the compressed quadtree, this path is replaced by the single edge $\sigma_1\sigma_k$, see Figure 2.6. Such a compressed quadtree has $O(n)$ vertices, height $O(n)$, and it can be constructed in $O(n \log n)$ time (see, e.g., [Buc+11, Appendix A] and [Har11]).

Consider the following order $\leq_Z$ on the cells of $\mathcal{G}$. Let $\sigma, \tau \in \mathcal{G}$. If $\sigma \subseteq \tau$, then $\sigma \leq_Z \tau$ and if $\tau \subseteq \sigma$, then $\tau \leq_Z \sigma$. If $\sigma \cap \tau = \emptyset$, let ρ be the smallest cell such that $\sigma \subseteq \rho$ and $\tau \subseteq \rho$. Furthermore, let σ' be the largest cells such that $\sigma \subseteq \sigma' \subset \rho$, and similarly define τ' as the largest cell with $\tau \subseteq \tau' \subset \rho$. We set $\sigma \leq_Z \tau$ if σ' is before τ' in the order shown in Figure 2.7 and $\tau \leq_Z \sigma$, otherwise. The order defined this way is called the *Z-order*. It is known that the Z-order is a total order on the cells of $\mathcal{G}$, see [BM11] for more details. As restated in the following lemma, given $\sigma, \tau \in \mathcal{G}$, we can decide if $\sigma \leq_Z \tau$ in constant time.



(a) Tree representation of a quadtree $\mathcal{Q}$ and its compressed version $\mathcal{Q}^c$.



(b) Geometric representation of $\mathcal{Q}'$ and $\mathcal{Q}^{c\prime}$ from Figure 2.6(a).

Figure 2.6: Compressed quadtrees

Lemma 2.2 (Chapter 2 in Har-Peled [Har11]). *Suppose the floor function and the first differing bit in the binary representations* $\mathrm{msb}(x, y)$ *of two given real numbers can be computed in* $O(1)$ *time. Then, we can decide in* $O(1)$ *time for two given cells* $\sigma, \tau \in \mathcal{G}$ *whether* $\sigma \leq_Z \tau$ *or* $\tau \leq_Z \sigma$.

The *linearized compressed quadtree* $\mathcal{L}$ for S is the sorted sequence of cells obtained by listing the nodes of the compressed quadtree $\mathcal{Q}^c$ according to a postorder traversal, where the children of a node $\sigma \in \mathcal{Q}^c$ are visited according to the Z-order from Figure 2.7. The cells in $\mathcal{L}$ appear in increasing Z-order, and searching for a given cell $\sigma \in \mathcal{G}$ reduces to a simple predecessor search in $\mathcal{L}$, as is made explicit in the following lemma.

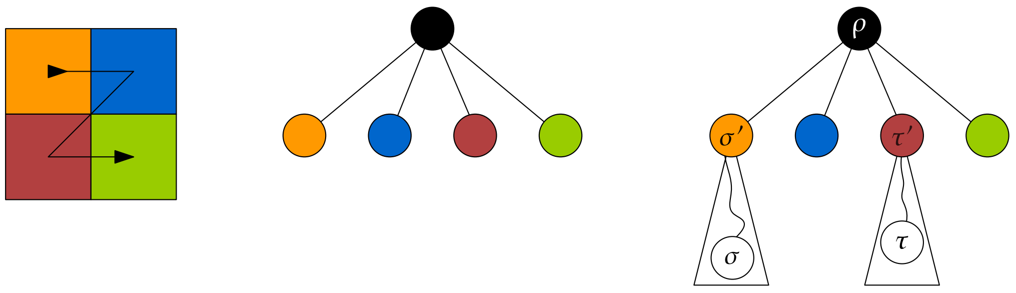


Figure 2.7: We order the children in the quadtree by their Z-Order. On the very right we have $\sigma \leq_Z \sigma' \leq_Z \tau \leq_Z \tau'$.

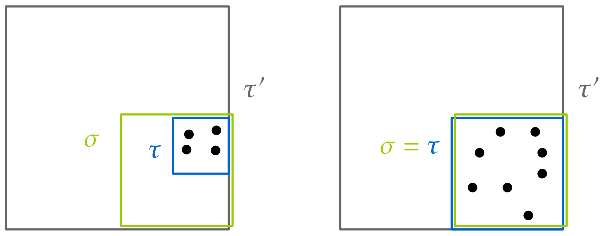


Figure 2.8: If σ is not in $\mathcal{Q}^c$ all sites in σ are contained in two cells of $\mathcal{Q}^c$ containing the same sites. In the other case, $\sigma = \tau$ is the cell in $\mathcal{Q}^c$ containing the same sites.

Lemma 2.3. *Let σ be a cell of $\mathcal{G}$, and let $\mathcal{L}$ be the linearized compressed quadtree on S. Let $\tau = \max_Z\{\rho \in \mathcal{L} \mid \rho \leq_Z \sigma\}$ be the Z-predecessor of σ in $\mathcal{L}$ or $\tau = \emptyset$, if the predecessor does not exist. Then, if $\sigma \cap \tau = \emptyset$, then also $\sigma \cap S = \emptyset$, and if $\sigma \cap \tau \neq \emptyset$, then $\sigma \cap S = \tau \cap S$.*

Proof. Let $\mathcal{Q}^c$ be the compressed quadtree on S, and let $\mathcal{Q}^c_\sigma = \{\tau \in \mathcal{Q}^c \mid \tau \subseteq \sigma\}$ be the cells in $\mathcal{Q}^c$ that are contained in σ. If $\mathcal{Q}^c_\sigma$ is non-empty, then it is a connected subtree of $\mathcal{Q}^c$. Let τ be the root of this subtree. Then, $\tau = \max_Z\{\rho \in \mathcal{Q}^c_\sigma\}$ and $\tau \leq_Z \sigma$. Furthermore, all other cells in $\mathcal{Q}^c \setminus \mathcal{Q}^c_\sigma$ are either smaller than all cells in $\mathcal{Q}^c_\sigma$ or larger than σ with regard to the Z-order. Thus, τ is the Z-predecessor of σ in $\mathcal{L}$, and $\sigma \cap S = \tau \cap S \neq \emptyset$. Otherwise, if $\mathcal{Q}^c_\sigma = \emptyset$, the Z-predecessor of σ in $\mathcal{L}$ either does not exist or is disjoint from σ. Thus, in this case, we have $\emptyset = \sigma \cap \tau = \sigma \cap S$. $\square$

Note that as $\mathcal{Q}^c$ is the compressed quadtree, for each cell σ of $\mathcal{G}$ there are at most two cells $\tau', \tau \in \mathcal{Q}^c$ such that $S \cap \tau = S \cap \tau' = S \cap \sigma$, see Figure 2.8. Without loss of generality it holds that $|\tau'| \geq |\sigma| \geq |\tau|$. By Lemma 2.3 the Z-predecessor is the smaller of these cells.

2.4 Canonical Decomposition

Given a set X of n elements, where each element x is assigned a unique numerical value r_x. We aim to find canonical subsets of these elements, that allow us to efficiently represent all elements of X, having a value in some given interval. There is a standard approach for this, which is often used in range query applications [dBer+08; WL85].

All concepts defined in this section are depicted in Figure 2.9. The elements of X are stored in the leaves of a balanced binary search tree B, sorted by the values r_x. Each inner vertex of B corresponds to a subset of X, consisting of consecutive elements. We call these subsets *canonical subsets* and denote the canonical subset defined by a vertex v by $\mathcal{I}_v$.

Now consider the subset X' of elements defined by a given interval. This interval can be given either by upper and lower bounds on the value of the elements, or by explicitly giving bounding elements. As each element can be represented by its unique value, we will without loss of generality assume the case of numerical bounds. Furthermore, depending on the application, the bounds on the interval can be inclusive or exclusive. Both cases will be handled here.

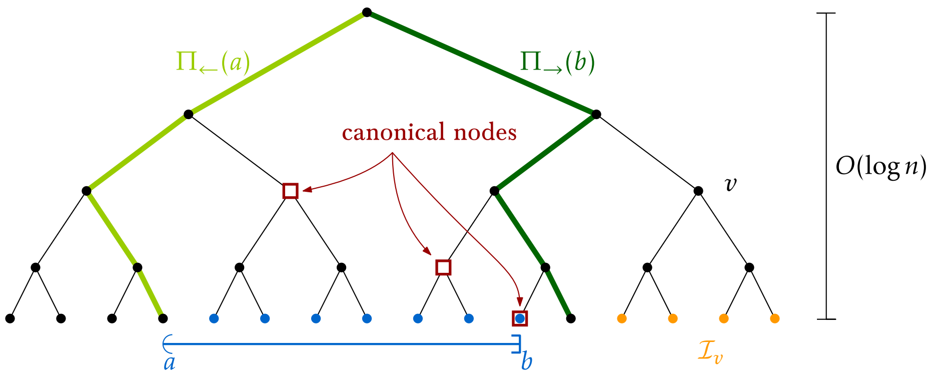


Figure 2.9: Depiction of the various concepts used in the range queries.

Given a value a, let its predecessor be the element $\max\{x \in X \mid r_x \leq a\}$ and its proper predecessor the element $\max\{x \in X \mid r_x < a\}$. Similarly the (proper) successor of a is the element $\min\{x \in X \mid r_x \geq a\}$ ($\min\{x \in X \mid r_x > a\}$). Let the interval considered for the range query be given by a lower bound a and an upper bound b. To answer the range queries we then consider the search path to the (proper) predecessor of a and to the (proper) successor of b. We denote the search path to the predecessor of a by $\Pi^{\bullet}_{\leftarrow}(a)$ and the search path to the proper predecessor of a by $\Pi^{\circ}_{\leftarrow}(a)$. Similarly, we denote the search path to the (proper) successor of b by $\Pi^{\circ}_{\rightarrow}(b)$ and $\Pi^{\bullet}_{\rightarrow}(b)$ respectively. We set these paths to $\emptyset$ if the searched element is not in B. Now we can define paths $\Pi_{\leftarrow}(a)$ and $\Pi_{\rightarrow}(b)$.

$$\Pi_{\leftarrow}(a) = \begin{cases} \Pi^{\bullet}_{\leftarrow}(a) & \text{if } a \text{ is excluded from the bound; and} \\ \Pi^{\circ}_{\leftarrow}(a) & \text{if } a \text{ is included in the bound.} \end{cases}$$

$$\Pi_{\rightarrow}(b) = \begin{cases} \Pi^{\bullet}_{\rightarrow}(b) & \text{if } b \text{ is excluded from the bound; and} \\ \Pi^{\circ}_{\rightarrow}(b) & \text{if } b \text{ is included in the bound.} \end{cases}$$

We call the union of all right siblings of vertices on $\Pi_{\leftarrow}(a)$ with the left siblings of the vertices on $\Pi_{\rightarrow}(b)$ the *canonical nodes* for this interval. Standard arguments, see for example [dBer+08, Section 5.1], yield the following result:

Lemma 2.4. *The total size of the canonical subsets is $O(n \log n)$. The tree B and the canonical subsets can be built in $O(n \log n)$ time. The number of canonical nodes for each interval bounded by a and b is $O(\log n)$. The canonical subsets for some interval form a partition of the subset defined by this interval.*

Proof. Since element $x \in X$ appears in the $O(\log n)$ subsets along its search path, the total size of the subsets is $O(n \log n)$. To construct B we first sort the elements and then construct the binary search tree in a bottom up fashion. As the canonical nodes are the siblings along two root-to-leaf paths in B, there are at most $O(\log n)$ of them. The partition property follows by construction. □

I

Disk Graphs and Transmission Graphs

CHAPTER 3 Computing the Girth in Disk Graphs

In this chapter we consider the problem of computing the girth of a disk graph. As already mentioned in Section 1.2, this problem is closely related to that of finding the shortest triangle. The main result of this chapter is an algorithm that computes the weighted girth of a disk graph in expected time $O(n \log n)$. Recall from Chapter 2 that when we talk about weighted disk graphs, that an edge between sites s and t is weighted by the Euclidean distance $\|st\|$ between the sites. For this chapter, in addition to the properties given in Section 2.1, we will assume that all edge lengths (and more generally shortest path distances) are pairwise distinct and that no site lies on a disk boundary.

The main result described in Section 3.2 extends ideas and approaches used in the algorithm for the weighted triangles of Kaplan et al. [Kap+19]. For convenience, we describe their algorithm and highlight the lemmas and theorems of Kaplan et al. relevant to our result in Section 3.1.[1] Also in Section 3.1, we show that the results given in this chapter and in Sections 4.2 and 4.3 are basically tight. To this end, we explicitly give the reduction from Element-Uniqueness to the problem of finding a triangle in unit disk graphs originally given by Polishchuk [Pol17] which implies a lower bound of $\Omega(n \log n)$ in the algebraic decision tree model.

3.1 Finding a (shortest) Triangle in a Disk Graph

In this section, we describe the algorithms for finding unweighted and weighted triangles by Kaplan et al. [Kap+19]. The central part of testing if a given graph contains a triangle is the following property due to Evans et al. [Eva+16].

[1]The results on disk graphs in the paper presented at ESA'19 [Kap+19] are the combination of related results that were presented before at EuroCG'17 [Kap+17] and EuroCG'18 [Kap+18]. The results presented in Section 3.1 are those presented at EuroCG'17 [Kap+17] which represent the state at affairs, when I started to work on this topic.

Lemma 3.1 (Evans et al. [Eva+16], Lemma 2.1 Kaplan et al. [Kap+19]). *Let $\mathcal{D}(S)$ be a disk graph that is not plane. Then there are three sites whose associated disks intersect in a common point.*

Lemma 3.1 tells us that if $\mathcal{D}(S)$ is not plane, it contains a triangle. Kaplan et al. now give a simple algorithm for deciding if the graph is plane. They do repeated line sweeps over the point set to explicitly construct the disk graph.

They stop this process if at some point they construct more than $6n - 12$ edges, and report that the underlying graph is not planar. This upper bound on the number of edges in a planar graph follows from the well known Euler's formula. In the other case the plane sweeps result in an embedding of the graph. With another plane sweep it can be checked if the embedding is plane. Again, if the answer is no, there is a triangle by Lemma 3.1. If the graph is plane, and therefore also planar, Kaplan et al. employ a linear time algorithm for finding triangles [Cha99; PY81] in a planar graph.

We will need two consequences of this algorithm for the weighted girth. The first is the following lemma:

Lemma 3.2 (Theorem 2.2 [Kap+19]). *Let $\mathcal{D}(S)$ be a disk graph on n sites. A triangle in $\mathcal{D}(S)$ can be found in $O(n \log n)$ worst-case time, if it exists.*

Since we do not only need the information if the triangle exists, but also an explicit representation of the abstract graph, we make the following corollary of Lemma 3.2 explicit:

Lemma 3.3 (Theorem 2.2 (implicit) [Kap+19]). *Let $\mathcal{D}(S)$ be a disk graph on n sites. In time $O(n \log n)$, it can be decided if $\mathcal{D}(S)$ is plane. If the answer is "yes", an explicit representation of the graph can be computed within the same time bound.*

Lemma 3.2 settles the unweighted case. Kaplan et al. then use this result as a subroutine to find the shortest triangle. In the following we repeat their main concepts and results.

The main strategy is to first solve the decision problem: *Given $W > 0$, does $\mathcal{D}(S)$ contain a triangle with perimeter at most W?* After an algorithm for the decision problem is available, Chan's randomized geometric optimization framework [Cha99] yields an optimization algorithm with the same time bound.

The decision algorithm uses four shifted grids $\mathcal{G}_{\mathrm{I}}, \dots, \mathcal{G}_{\mathrm{IV}}$, where $|\sigma| = \frac{W}{3}$ for each cell σ of any of the grids. For some calculations in both the analysis and description of the algorithm, the side length $\ell = \frac{W}{3\sqrt{2}}$ is needed. This definition of the diameter ensures that if all sites of a triangle lie inside of one cell, the total length of this triangle is at most W.

The grids are shifted, such that $\mathcal{G}_{\mathrm{I}}, \dots, \mathcal{G}_{\mathrm{IV}}$ have a vertex at $(0,0), (\frac{\ell}{2}, 0), (0, \frac{\ell}{2})$ and $(\frac{\ell}{2}, \frac{\ell}{2})$, respectively, see Figure 3.1. This shifting makes sure that triangles whose longest edge has length at most $\frac{\ell}{2}$, will be completely contained in a cell of one of the grids (Lemma 2.3 by Kaplan et al. [Kap+19]). Such triangles can be found by using Lemma 3.2 on each cell in overall $O(n \log n)$ time.

If no such triangle is found, each triangle will have at least one edge with side length at least $\frac{\ell}{2}$, and thus at least one of its vertices is a site with radius at least $\frac{\ell}{4}$. The sites are now partitioned into *large sites* with radius at least $\frac{\ell}{4}$ and *small sites* which are all

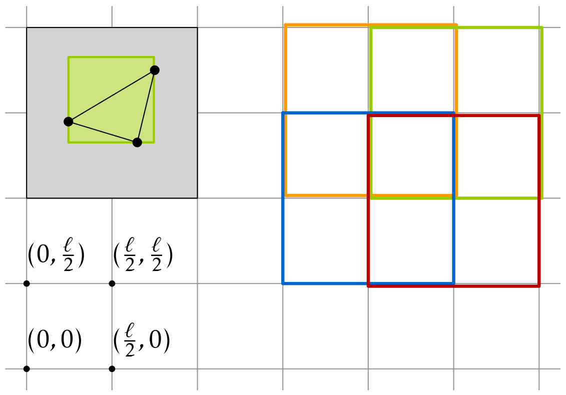


Figure 3.1: The four shifted grids, with a cell from each grid shown in red, orange, green, and blue, respectively. Every square with side length at most $\frac{\ell}{2}$ is completely contained in a single grid cell.

remaining sites. Now Kaplan et al. first note that if any cell contains more than 18 large sites, there is a triangle in this cell. We restate their lemma here:

Lemma 3.4 (Lemma 2.4 [Kap+19]). *Let $\sigma \in \bigcup_{i=\mathrm{I}}^{\mathrm{IV}} \mathcal{G}_i$ be a nonempty grid cell, and suppose that σ does not contain a triangle. Then σ contains at most 18 large sites.*

As triangles consisting only of small sites would have been found before, the only remaining options for triangles is to have zero, one or two small sites. Kaplan et al. consider each of these options separately and give algorithms with $O(n \log n)$ running time for each. The result is summarized in the following lemma:

Lemma 3.5 (Lemma 2.5 [Kap+19]). *Let $\mathcal{D}(S)$ be a disk graph on n sites, and let $W > 0$. We can decide in $O(n \log n)$ worst-case time whether $\mathcal{D}(S)$ contains a triangle of perimeter at most W.*

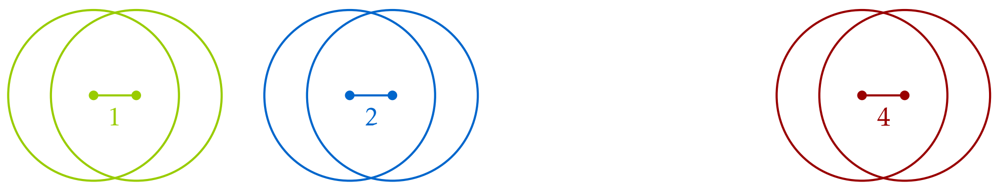
The main result then follows from an application of Chan's randomized optimization framework, restated here for convenience:

Lemma 3.6 (Lemma 2.1 [Cha99]). *Given a* problem space Π, *let $w(P) \in \mathbb{R}$ be the* optimum *and $|P| \in \mathbb{N}$ be the* size *of some problem $P \in \Pi$. Let $\alpha < 1$, $\varepsilon > 0$, and $r \in \mathbb{N}$ be constants, and let $\delta(\cdot)$ be a function such that $\frac{\delta(n)}{n^\varepsilon}$ is monotone increasing in n. Given any optimization problem $P \in \Pi$ with optimum $w(P)$, suppose that within time $\delta(|P|)$,*

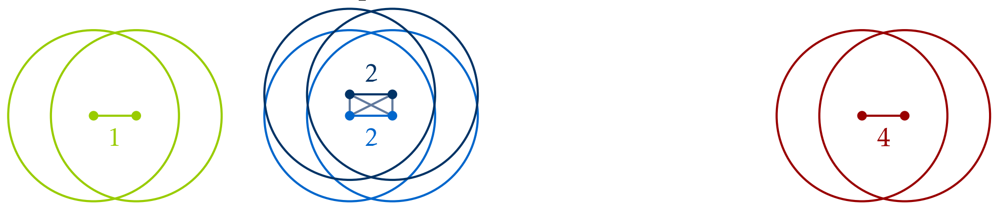
(a) we can decide whether $w(P) < t$, for any given $t \in \mathbb{R}$; and

(b) we can construct r subproblems $P_1, \dots, P_r$, each of size at most $\lceil \alpha |P| \rceil$, so that $w(P) = \min\{w(P_1), \dots, w(P_r)\}$.

Then we can compute $w(P)$ in total expected time $O(\delta(|P|))$.



(a) $T = [1,2,4]$, all elements are unique.



(b) $T = [1,2,2,4]$, the element 2 is contained twice.

Figure 3.2: The reduction from element uniqueness.

Kaplan et al. use Lemma 3.5 to show that (a) holds. For (b), four subsets $S_0, \dots, S_3 \subseteq S = \{s_1, \dots, s_n\}$ are constructed, where $s_i \in S_j$ with $i \not\equiv j \pmod 4$. Then any triple a, b, c of points is contained in at least one set S_i and Lemma 3.6 can be applied with $\alpha = \frac{3}{4}, \varepsilon = 1, r = 4$ and $\delta = O(n \log n)$. Their main theorem follows.

Theorem 3.7 (Theorem 2.6 [Kap+19]). *Let $\mathcal{D}(S)$ be a weighted disk graph on n sites. We can compute a shortest triangle in $\mathcal{D}(S)$ in $O(n \log n)$ expected time, if one exists.*

Next, we proceed to describe the lower bound due to Polishchuk [Pol17, personal communication]. This bound holds in the *algebraic decision tree model.*

Lemma 3.8 (Polishchuk [Pol17]). *In the algebraic decision tree model there is an $\Omega(n \log n)$ lower bound for testing if a given (unit) disk graph contains a triangle.*

Proof. This lower bound follows by a reduction from Element-Uniqueness. The Element-Uniqueness problem asks for a given set T of n integers, whether all of them are unique. It is known, that for Element-Uniqueness there is an $\Omega(n \log n)$ lower bound in the algebraic decision tree model [Ben83].

The reduction from Element-Uniqueness now works as follows: for each $x \in T$ create the sites $(3 \cdot x, 0)$ and $(3 \cdot x + 0.5, 0)$ and call the resulting set S. We now consider the unit disk graph on these sites, or equivalently set $r_s = 1$ for each site created this way. This reduction can be applied in $O(n)$ time. Now if all values $x \in T$ were unique, the graph consists of n connected components, where each of the components consist of a single edge. If on the other hand there are two elements in T that have the same value, this corresponds to a clique of size four, and thus also a triangle in $\mathcal{D}(S)$, see Figure 3.2. □

If the resulting graph in Lemma 3.8 is carefully considered, it can be seen that the result directly carries over to transmission graphs. Also as each triangle produced this way has perimeter at most 1, the decision problem of finding a triangle or cycle of length at most W also inherits this lower bound. Thus, we cannot hope to find deterministic algorithms that are consistent with the algebraic decision tree model and run faster than $O(n \log n)$ for finding (short) triangles and computing the girth in disk graphs and transmission graphs.

3.2 Computing the Weighted Girth of a Disk Graph

In this section, we show how to adapt and extend the ideas of Kaplan et al. to compute the weighted girth of a disk graph. First, we describe how to find the shortest cycle through a given vertex in a weighted graph with certain properties. This is then used as a subroutine to compute the weighted girth of a disk graph.

Let G be an undirected graph with nonnegative edge weights so that all shortest paths and cycles in G have pairwise distinct lengths, and for all edges $\{u,v\}$, the shortest path from u to v is the edge $\{u,v\}$. We present an efficient deterministic algorithm that, given G and a vertex s, computes the shortest cycle in G containing s, if it exists.[2] We remark that there also is a simple randomized algorithm presented by Yuster [Yus11, Section 2].

The next lemma states a structural property of the shortest cycle through s. It resembles Lemma 1 of Roditty and Williams [RW11] that deals with an overall shortest cycle in G. This structural property will then be used in Theorem 3.10 to show the correctness of our algorithm.

Lemma 3.9. *The shortest cycle in G that contains s consists of two paths in the shortest path tree of s, and one additional edge.*

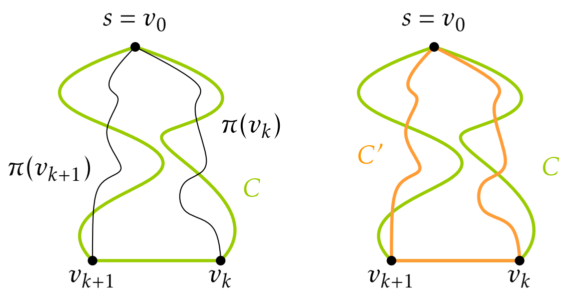
Proof. Let $C = v_0, v_1, v_2, \dots, v_{\ell-1}, v_\ell$ be the shortest cycle in G containing s, where all vertices v_i, $0 \leq i \leq \ell - 1$ are pairwise distinct, $\ell \geq 3$, and $v_0 = v_\ell = s$. For $v_i \in C$, let $d_1(v_i)$ be the length of the path $s, v_1, \dots, v_i$, and let $d_2(v_i)$ be the length of the path $v_i, v_{i+1}, \dots, s$. Let $\pi(v_i)$ denote the shortest path from s to v_i, and let $\|v_i v_{i+1}\|$ be the, length of the edge $\{v_i, v_{i+1}\}$. The weights of the edges can be arbitrary and do not have to be connected to the Euclidean distance between points.

Suppose that C is not of the desired form. Let $\{v_k, v_{k+1}\}$ be the edge on C with $d_1(v_k) < \|v_k v_{k+1}\| + d_2(v_{k+1})$ and $d_2(v_{k+1}) < d_1(v_k) + \|v_k v_{k+1}\|$. By our assumptions on G, the edge $v_k v_{k+1}$ exists and is uniquely defined. In addition we have $k \notin \{0, \ell - 1\}$. We distinguish two cases, illustrated in Figure 3.3.

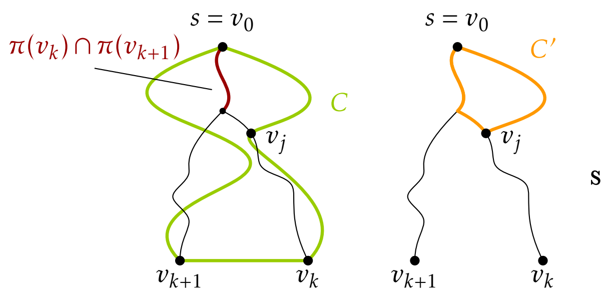
$\pi(v_k) \cap \pi(v_{k+1}) = \{s\}$ (Figure 3.3(a)): Let C' be the cycle defined by $\pi(v_k)$, the edge $\{v_k, v_{k+1}\}$, and $\pi(v_{k+1})$. As $s \neq v_k$ and $s \neq v_{k+1}$ and since the edge $\{v_k, v_{k+1}\}$ neither appears on $\pi(v_k)$ nor on $\pi(v_{k+1})$, it follows that C' is a proper cycle. Furthermore, by assumption, C' is strictly shorter than C, because $\pi(v_k)$ is shorter than $d_1(v_k)$ or $\pi(v_{k+1})$ is shorter than $d_2(v_{k+1})$. This contradicts our assumption on the form of C.

$|\pi(v_k) \cap \pi(v_{k+1})| \geq 2$ (Figure 3.3(b)): Since $\pi(v_k)$ and $\pi(v_{k+1})$ are shortest paths, their intersection is a prefix of both paths. As the shortest path from u to v is the edge $\{u,v\}$ by our assumption on G, at least one of $v_1, v_{\ell-1}$ is not in $\pi(v_k) \cup \pi(v_{k+1})$.

[2]Even though this seems to be a simple fact, we could not locate a previous reference for this.



(a) $\pi(v_k)$ and $\pi(v_{k+1})$ intersect only at s.



(b) $\pi(v_k)$ and $\pi(v_{k+1})$ have a common prefix.

Figure 3.3: The two cases for $\pi(v_k) \cap \pi(v_{k+1})$.

Without loss of generality, let this vertex be v_1. Let j be the smallest index such that $v_j \in \pi(v_k) \cup \pi(v_{k+1})$. We have $j \in \{2, \dots, k\}$.

Consider the cycle C' that starts at s, follows C along $v_1, v_2, \dots$ until it reaches v_j, and then returns along $\pi(v_k)$ or $\pi(v_{k+1})$ to s. By construction, C' is a proper cycle. Furthermore, $C' \neq C$, because even if $j = k$, the path $\pi(v_k)$ cannot contain the part of C from v_{k+1} to s, due to the choice of k. Finally, C' is strictly shorter than C, because the second part of C' from v_j to s follows a shortest path and is thus strictly shorter than $d_2(v_j)$. Again, we reached a contradiction. □

By using this technical insight, we can use an extension of Dijkstra's algorithm to get an efficient algorithm for finding a shortest cycle with a given vertex s.

Theorem 3.10. *Let $G = (V, E)$ be a weighted graph with n vertices and m edges that has the properties given at the beginning of this section. Given a vertex $s \in V$, we can compute the shortest cycle in G that contains s in $O(n \log n + m)$ time, if it exists.*

Proof. We find the shortest path tree T for s in G, and we traverse T to find for each vertex v in $T \setminus \{s\}$ the vertex $b[v]$ that is the successor of s on the shortest path from s to v. Then, we iterate over all edges in E that are not in T. For each such $e = \{u, v\}$, we check if $b[u] \neq b[v]$. If so, e closes a cycle in T that contains s. We determine the length of this cycle in $O(1)$ time and return the shortest cycle found in this way.

The correctness follows from Lemma 3.9. As for the running time, it takes $O(n \log n + m)$ time to find the shortest path tree for s with Dijkstra's algorithm and Fibonacci heaps [Cor+09, Chapter 24.3]. After that, it takes $O(n)$ time to compute the nodes $b[v]$, for $v \in T \setminus \{s\}$, and $O(m)$ time to iterate over the edges not in T. The length of the cycle associated with such an edge e can be computed in $O(1)$ time, using the shortest path distances in T and the length of e. □

Let $\mathcal{D}(S)$ be a weighted disk graph on n sites. In the following we show how a careful combination of the tools by Kaplan et al., described in Section 3.1 and the variant of Dijsktra's algorithm described in Theorem 3.10 gives an efficient algorithm for the weighted girth of $\mathcal{D}(S)$.

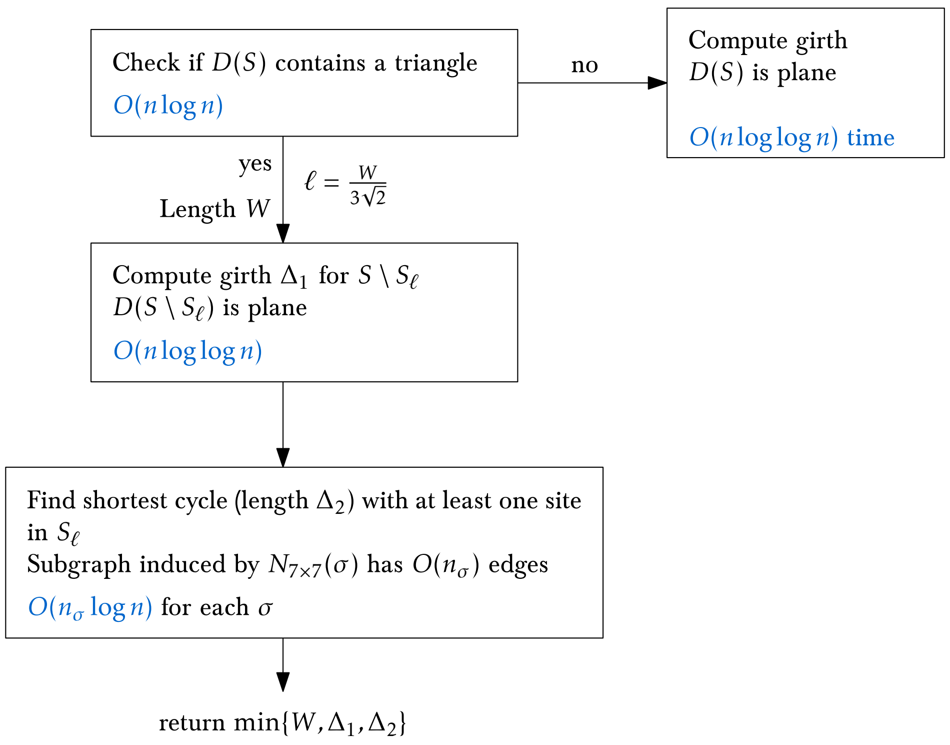

Check if $D(S)$ contains a triangle
$O(n \log n)$
no
Compute girth
$D(S)$ is plane
$O(n \log\log n)$ time
yes
Length W
$\ell = \frac{W}{3\sqrt{2}}$
Compute girth Δ_1 for $S \setminus S_\ell$
$D(S \setminus S_\ell)$ is plane
$O(n \log\log n)$
Find shortest cycle (length Δ_2) with at least one site in S_ℓ
Subgraph induced by $N_{7\times7}(\sigma)$ has $O(n_\sigma)$ edges
$O(n_\sigma \log n)$ for each σ
return $\min\{W, \Delta_1, \Delta_2\}$

Figure 3.4: Overview of the weighted girth algorithm.

Theorem 3.11. *Given a weighted disk graph $\mathcal{D}(S)$ on n sites, we can compute the weighted girth of $\mathcal{D}(S)$ in $O(n \log n)$ expected time.*

Proof. We consider multiple cases to exclude certain types of cycles one by one. See Figure 3.4 for an overview of the algorithm. We first find the shortest triangle in $\mathcal{D}(S)$, if it exists, using Theorem 3.7. If $\mathcal{D}(S)$ has no triangle, it is plane by Lemma 3.1. By Lemma 3.3, it then can be constructed explicitly in $O(n \log n)$ time. The girth of $\mathcal{D}(S)$ can then be found in $O(n \log\log n)$ additional time by using the algorithm of Łącki and Sankowski [ŁS11, Section 5].

Now, suppose $\mathcal{D}(S)$ contains a triangle, and let W be the length of the shortest triangle in $\mathcal{D}(S)$. Clearly W is an upper bound for the girth of $\mathcal{D}(S)$. As for the shortest triangle, set $\ell = \frac{W}{3\sqrt{2}}$, and let $\mathcal{G}_{\mathrm{I}}$ be the grid where each cell has diameter $\frac{W}{3}$, that has the origin $(0,0)$ as a grid point. Note that the side length of each cell in $\mathcal{G}_{\mathrm{I}}$ is ℓ. In contrast to the algorithm for finding the shortest triangle, adding shifted copies of the grid is not necessary here. We call a site $s \in S$ *large* if $r_s \geq \frac{\ell}{4}$, and we let $S_\ell \subseteq S$ be the set of large sites. All other sites are called *small*. As the length of the shortest triangle is already known, we now shift our focus to cycles in $\mathcal{D}(S)$ with more than three vertices and length less than W. If such cycles exist, the shortest of them realizes the girth of $\mathcal{D}(S)$. First, consider cycles in the induced subgraph $\mathcal{D}(S \setminus S_\ell)$. The graph $\mathcal{D}(S \setminus S_\ell)$ has no triangle, as such a triangle would have length less than $3 \cdot \frac{\ell}{2} < W$. Then by the same argument as in the case that $\mathcal{D}(S)$ does not contain a triangle at all, $\mathcal{D}(S \setminus S_\ell)$ is plane and the shortest cycle can be found in $O(n \log n)$ time. Let Δ_1 be the weighted girth of $D(S \setminus S_\ell)$.

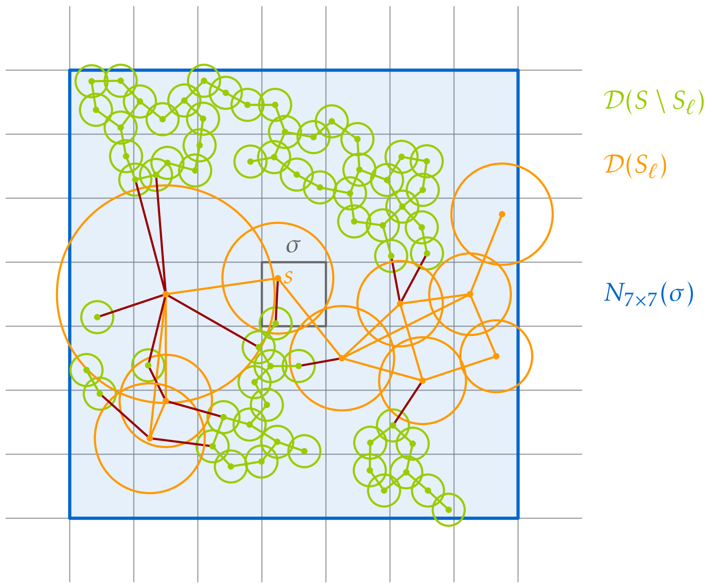


Figure 3.5: The graph $\mathcal{D}(S \setminus S_\ell)$ is plane and the graph $\mathcal{D}(S_\ell)$ is sparse but not necessarily plane. The disk graph $\mathcal{D}(S \cap S_\sigma)$ has $O(n_\sigma)$ edges.

Next, consider cycles that have at least one large site. Let $\sigma \in \mathcal{G}$ be a cell of the grid. Then for every $s \in \sigma$ by triangle inequality every site contained in the same cycle of length at most W as s lies in $D(s, \frac{W}{2})$. Note that Lemma 2.1 is not limited to cells of the hierchical grid. Thus, Lemma 2.1 with $|\sigma| = \frac{W}{3}$ and $r = \frac{W}{2}$ implies that the cycle is contained in $N_{7\times7}(\sigma)$.

Now, for each grid cell σ, consider all large sites $s \in S_\ell \cap \sigma$. For each such site, we want to find the shortest cycle through s in the subgraph $\mathcal{D}(S_\sigma)$ of $\mathcal{D}(S)$ defined on all sites in $N_{7\times7}(\sigma)$. For this the graph $\mathcal{D}(S_\sigma)$ is constructed explicitly and Theorem 3.10 is applied with s as the starting vertex for each $s \in S_\ell \cap \sigma$. Let Δ_2 be the smallest length of such a cycle, over all grid cells σ and all large sites $s \in S_\ell \cap \sigma$. See Figure 3.5 for an illustration of the following argument. As $\mathcal{D}(S \setminus S_\ell)$ has no triangles, the graph induced on the small sites in $\mathcal{D}(S_\sigma)$ is plane. Furthermore, if one cell of $N_{7\times7}(\sigma)$ had more than $O(1)$ large sites, by Lemma 3.4 it would contain a triangle. As all triangles have length at least W and a triangle inside a single grid cell has length less than W this cannot happen. Thus $\mathcal{D}(S_\sigma)$ is the union of a plane graph with a constant number of sites, and therefore has $O(n)$ edges. One run of Theorem 3.10 then takes $O(|S_\sigma| \log n)$ time. As each $\sigma \in \mathcal{G}_\mathrm{I}$ contains $O(1)$ large sites, the algorithm from Theorem 3.10 is executed $O(1)$ times for each S_σ. Furthermore, as each site lies in $O(1)$ neighborhoods, we have $\sum_{\sigma \in G_\mathrm{I}} |S_\sigma| = O(n)$, resulting in an overall running time of $O(n \log n)$ for this step.

The algorithm returns $\min\{W, \Delta_1, \Delta_2\}$. As all steps to determine these values can be implemented in $O(n \log n)$ expected time, the overall running time of the algorithm follows. If the desired output is not only the length of the shortest cycle, but also the cycle itself, appropriate pointers can be maintained during the execution of the algorithm. □

CHAPTER 4
Triangles and Cycles in Transmission Graphs

In this chapter we consider the problems of finding triangles and cycles in transmission graphs. The main insight used in the corresponding algorithms for disk graphs is that the graph is either sparse or contains a triangle. In Section 4.1 we show that this statement is not true for transmission graphs. We give an infinite family of site sets, such that the induced transmission graphs are strongly connected and have $\Theta(n^2)$ edges, while not containing a directed triangle.

Since similar techniques as for the disk graph case are not feasible, we develop a different approach for transmission graphs. In Section 4.2 we describe a simple algorithm to check for the existence of a triangle in a transmission graph, which is then extended to find a shortest such triangle in Section 4.3. The last algorithm of this chapter is then an algorithm to find a cycle of unweighted length at most k.

While the algorithms for the unweighted triangle and the cycles of length k are easy to describe, the main technical challenge is to implement them efficiently. To this end, we define three kinds of batched range queries. In Section 4.5 we show how to efficiently implement these range queries and in consequence also the algorithms of this chapter.

4.1 Dense Transmission Graphs can be Triangle Free

In this section we show that there is an infinite family of strongly connected transmission graphs that do not contain a directed triangle while having $\Theta(n^2)$ edges. We first give a simple family that does not satisfy the strongly connected property. This family already shows that triangle-free transmission graphs do not have to be sparse. However, to find triangles and cycles in transmission graphs, it suffices to consider the strongly connected components and one might still hope that the strongly connected components of a triangle-free transmission graph are sparse. To show that this is also not true, we then extend the family to one where all induced graphs are strongly connected, while the induced transmission graphs are still triangle free and have $\Theta(n^2)$ edges.

Lemma 4.1. *There is an infinite family $\mathcal{A} = \{A_1, A_2, \ldots,\}$ such that $|A_n| = n+1$ and $\mathcal{T}(A_n)$ has $\Theta(n^2)$ edges while containing no triangle.*

Proof. We define the set $A_n = \{a_0, \ldots, a_n\}$ to be $n+1$ sites on the x-axis. Let a_0 be the site in the origin with radius 1 and set the x-coordinate of a_i to $2^{i-1} + (2^i - 1) \cdot \varepsilon$ for the remaining sites, where $\varepsilon > 0$ is a sufficiently small constant. The radii of the sites a_i for $i \geq 1$ are the same as their x-coordinates. A depiction of the set A_n can be found, by only considering the blue and black sites in Figure 4.1.

We now show that the edge set of $\mathcal{T}(A_n)$ is $E(A) = \{a_j a_i \mid i < j\}$. As all points in A_n lie on a horizontal line, we only have to argue about their x-coordinates. By definition, a_i is left of a_j for $i < j$. As the radii are chosen such that the edge $a_j a_0$ exists for $j \geq 1$, all other edges $a_j a_i$ also exist for $i < j$. Given a_i, to show that no edge to a_j with $i < j$ exists, it suffices to show that the edge $a_i a_{i+1}$ does not exist. This can be seen by considering the horizontal distance of a_i and a_{i+1}.

$$\begin{aligned}
\|a_{i+1} a_i\|_x &= 2^i + (2^{i+1} - 1)\varepsilon - (2^{i-1} + (2^i - 1)\varepsilon) \\
&= 2^{i-1} + 2^i \varepsilon \\
&= r_{a_i} + \varepsilon
\end{aligned}$$

Thus, $\mathcal{T}(A_n)$ contains $\Theta(n^2)$ edges. Furthermore the order $a_0, \ldots, a_n$ is a topological sorting of $\mathcal{T}(A_n)$ and thus $\mathcal{T}(A_n)$ is acyclic and in particular does not contain a directed triangle. $\square$

Now, we extend this family such that the resulting graph is strongly connected. We define the family $\mathcal{S} = \{S_5, S_6, \ldots\}$ as follows: Each set S_n consists of $3n$ sites which can be partitioned into three sets $A = \{a_0, \ldots a_n\}$, $B = \{b_1, b_2, b_3\}$ and $C = \{c_1, \ldots c_{2n-4}\}$. An illustration of the disks defined by S_5 can be found in Figure 4.1, the underlying graph structure is depicted in Figure 4.2.

Let $0 < \varepsilon \leq \frac{1}{2^n}$ and $\alpha = \cos^{-1}(\sqrt{2} - 1)$ and define the sets as follows, where the sites a_i are the same as in the set A_n defined in the proof for Lemma 4.1:

$$\begin{aligned}
a_0 &= (0,0) & r_{a_0} &= 1 & \\
a_i &= \left(2^{i-1} + (2^i - 1)\varepsilon, 0\right) & r_{a_i} &= 2^{i-1} + \varepsilon(2^i - 1) & i &= 1, \ldots, n
\end{aligned}$$

Let $r = r_{a_n}$ and $r' = \frac{r+\varepsilon}{2} = 2^{n-2} + 2^{n-1}\varepsilon$, we set

$$\begin{aligned}
b_i &= (r - i\cos(\alpha)r', i\sin(\alpha)r') & r_{b_i} &= r' & i &= 1,2 \\
b_3 &= \left(\sqrt{2}^{2n-4}, \frac{\sqrt{2}^{2n-4} - 1}{\sqrt{2} - 1}\right) & r_{b_3} &= \sqrt{2}^{2n-4} \left(= 2^{n-2}\right) & & \\
c_i &= \left(0, \frac{\sqrt{2}^i - 1}{\sqrt{2} - 1}\right) & r_{c_i} &= \sqrt{2}^i & i &= 1, \ldots, 2n-4
\end{aligned}$$

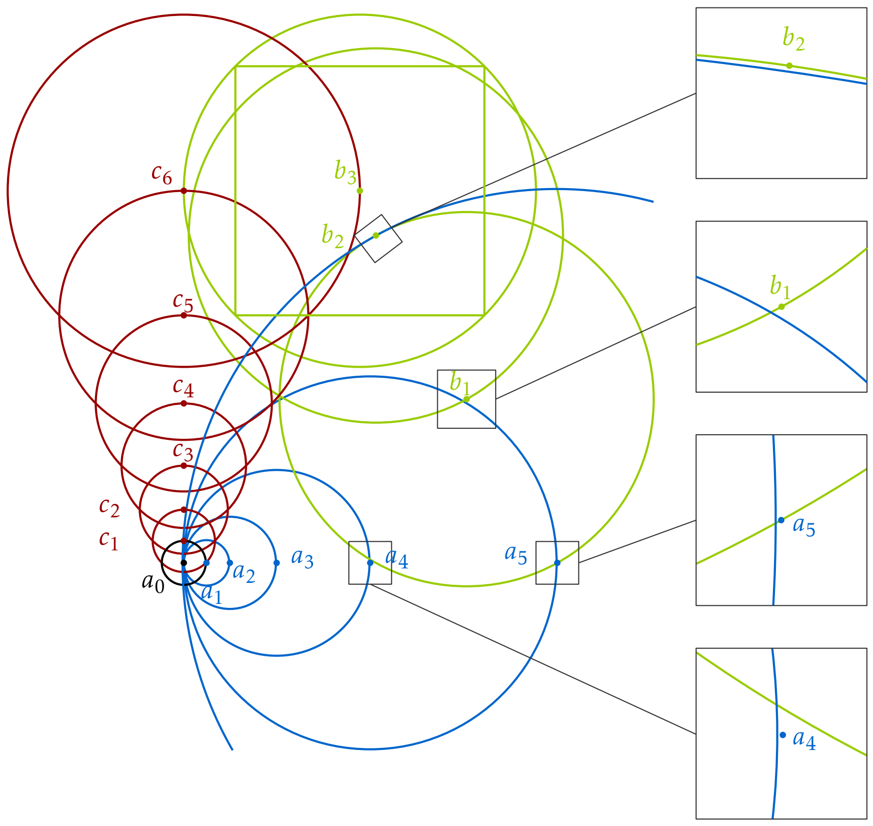


Figure 4.1: The sites and corresponding disks of S_5. The site b_2 lies in the square contained in b_3.

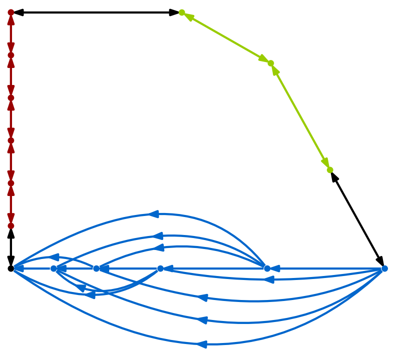


Figure 4.2: The graph structure of S_5, the coordinates are not to scale.

Lemma 4.2. *The graph $\mathcal{T}(S_n)$ consists of the union of the paths $a_n, b_1, b_2, b_3, c_{2n-4}, \dots, c_1, a_0$ and $a_0, c_1, \dots, c_{2n-4}, b_3, b_2, b_1, a_n$ with the set of edges $E(A) = \{a_j a_i \mid i < j\}$.*

Proof. We will first consider the structure in each set separately and then consider the three possible types of edges between the sets. The subgraph induced by A has the edge set $E(A)$ by Lemma 4.1. Now we consider the set B. The definition of b_1 and b_2 implies that $\|b_1 b_2\| = r'$ and thus the edges $b_1 b_2$ and $b_2 b_1$ exist. For the edges $b_2 b_3$ and $b_3 b_2$ we first note that $r_{b_3} = \sqrt{2}^{2n-4} < 2^{n-2} + 2^{n-1}\varepsilon = r'$, so it is enough to show the existence of the edge $b_3 b_2$. We consider the horizontal and vertical distance separately and show that both are at most $\sqrt{2}^{2n-5}$, which implies that b_2 lies in the square with side length $\sqrt{2}^{2n-5} = \frac{2^{n-2}}{\sqrt{2}}$ which is completely contained in b_3, refer again to Figure 4.1. We have:

$$\begin{aligned}
\|b_2 b_3\|_x &= \left| r - 2\cos(\alpha) r' - \sqrt{2}^{2n-4} \right| \\
&= \left| r - (\sqrt{2} - 1) \cdot (r + \varepsilon) - 2^{n-2} \right| \\
&= \left| r(2 - \sqrt{2}) - (\sqrt{2} - 1)\varepsilon - 2^{n-2} \right| \\
&= \left| 2^{n-1}(2 - \sqrt{2}) + \varepsilon \cdot \Big((2^n - 1) \cdot (2 - \sqrt{2}) - \sqrt{2} + 1 \Big) - 2^{n-2} \right| \\
&= \left| 2^{n-2}(3 - 2\sqrt{2}) + \varepsilon \cdot (2^n (2 - \sqrt{2}) - 1) \right| \\
&\leq 2^{n-2}(3 - 2\sqrt{2}) + 1 \\
&< \frac{2^{n-2}}{\sqrt{2}} && \text{for } n \geq 1 \\
\|b_2 b_3\|_y &= \left| 2\sin(\alpha)\frac{r + \varepsilon}{2} - \frac{2^{n-2} - 1}{\sqrt{2} - 1} \right| \\
&\leq \left| 0.95(r + \varepsilon) - (1 + \sqrt{2}) \cdot (2^{n-2} - 1) \right| \\
&= \left| 0.95 \cdot 2^{n-1} + 0.95 \cdot 2^n \varepsilon - (1 + \sqrt{2}) \cdot (2^{n-2} - 1) \right| \\
&= \left| 2^{n-2} \cdot (2 \cdot 0.95) + 0.95 \cdot 2^n \varepsilon - 2^{n-2}(1 + \sqrt{2}) + (1 + \sqrt{2}) \right| \\
&= \left| 2^{n-2}(0.9 - \sqrt{2}) + 0.95 \cdot 2^n \varepsilon + (1 + \sqrt{2}) \right|
\end{aligned}$$

As $0.95 \cdot 2^n \varepsilon \leq 1$, the term in the $|\cdot|$ is negative for $n \geq 5$. As $n \geq 5$ by assumption, we get:

$$\begin{aligned}
\|b_2 b_3\|_y &\leq 2^{n-2}(\sqrt{2} - 1) - 0.95 \cdot 2^n \varepsilon - (1 + \sqrt{2}) \\
&< \frac{2^{n-2}}{\sqrt{2}},
\end{aligned}$$

and therefore the edges $b_2 b_3$ and $b_3 b_2$ exist. Note that these calculations also immediately imply, that b_3 is above and left of b_2.

For the set C we have to show that c_ic_{i+1} and $c_{i+1}c_i$ are edges of $\mathcal{T}(S_n)$, but no other edges exist. As, similarly to the set A, all points in C lie on a vertical line in the order $c_1,\ldots,c_{2n-4}$ and have increasing radii, to show the nonexistence of the other edges, it suffices to show that no edge $c_{i+2}c_i$ exists. The edge set induced by C follows from the following calculations:

$$\begin{aligned}\|c_ic_{i+1}\| &= \frac{\sqrt{2}^{i+1}-1-(\sqrt{2}^{i}-1)}{\sqrt{2}-1}\\ &= \frac{\sqrt{2}^{i}(\sqrt{2}-1)}{\sqrt{2}-1}\\ &= \sqrt{2^i} = r_i\end{aligned}$$

$$\begin{aligned}\|c_ic_{i+2}\| &= \frac{\sqrt{2}^{i+2}-1-(\sqrt{2}^{i}-1)}{\sqrt{2}-1}\\ &= \frac{\sqrt{2}^{i}}{\sqrt{2}-1}\\ &> 2\sqrt{2}^{i} = \sqrt{2}^{i+2} = r_{c_{i+2}}\end{aligned}$$

Finally we consider edges connecting the different sets. First we note, that D_{c_1} intersects the x-axis at $\pm\frac{1}{\sqrt{2}}$ and the other disks in C do not intersect the x-axis at all. Furthermore, a_0 is the only site in A whose corresponding disk intersects the y-axis at more than one point. So c_1a_0 and a_0c_1 are the only edges connecting A and C.

Now we consider possible edges between A and B. By the definition of b_1 it is apparent, that the edges a_nb_1 and b_1a_n are present. To see that no other edges exist, note that D_{b_1} intersects the x-axis at a_n and at the point with distance $2r'\cos(\alpha)$ to the left of a_n. As $\alpha > \frac{\pi}{3}$ we have:

$$\begin{aligned}2\cos(\alpha)r' &< \frac{r+\varepsilon}{2}\\ &= 2^{n-2}+2^{n-1}\varepsilon,\end{aligned}$$

whereas $\|a_{n-1}a_n\| = 2^{n-2}+2^{n-1}\varepsilon$, so no edge (b_1,a_i) exists for $i \le n-1$. As $r_{a_i} \le r'$ for $i \le n-1$ this also implies that no other edges incident to a_i with $i \le n-1$ and b_1 exist. For b_2 and b_3 we consider the vertical distances. We already saw above, that b_3 lies above of b_2 and that $r_{b_2} \ge r_{b_3}$, so it suffices to consider b_2. We have:

$$\begin{aligned}\|a_ib_2\|_y &= 2\sin(\alpha)r'\\ &\ge \frac{\sqrt{3}}{2}(r+\varepsilon)\\ &= \sqrt{3}r'.\end{aligned}$$

Thus considering the vertical distance alone is enough to eliminate the possibility of all edges $b_j a_i$ and $a_i b_j$ for $i = 0, \dots, n-1$ and $j = 2, 3$. As for possible edges between a_n and b_2 or b_3, note that by definition $\|a_n b_2\| = (r + \varepsilon) > r$. Furthermore, as b_2 is left of a_n and b_3 is above and left of b_2, we can derive, that b_3 lies outside of D_{a_n} and thus no edges $a_n b_i$ or $b_i a_n$ for $i = 2, 3$ are possible.

Finally, we have possible connections between B and C. First we consider possible edges incident to b_2. For the x-coordinate of b_2 we have:

$$\begin{aligned} r - \cos(\alpha)(r + \varepsilon) &= r - (\sqrt{2} - 1) \cdot (r + \varepsilon) \\ &\geq r(2 - \sqrt{2}) \geq r' \end{aligned}$$

As b_1 is to the right of b_2 this implies that neither D_{b_1} nor D_{b_2} intersect the y-axis. With $r' \geq 2^{n-2}$ this excludes all edges between B and C incident to b_1 or b_2. Finally, by definition D_{b_3} intersects the y-axis only at the point c_{2n-4}. As $r_{b_3} = r_{c_{2n-4}} > r_{c_i}$ for $i \leq 2n-5$, this implies that the only edges between B and C are $b_3 c_{2n-4}$ and $c_{2n-4} b_3$, concluding the proof. □

Using this result, we can now show the main claim of this section.

Theorem 4.3. *There is an infinite family $\mathcal{S} = \{S_5, \dots\}$ of sets of sites, such that*

(a) $|S_n| = \Theta(n)$

(b) $\mathcal{T}(S_n)$ *has* $\Theta(n^2)$ *edges*

(c) $\mathcal{T}(S_n)$ *forms one strongly connected component; and*

(d) $\mathcal{T}(S_n)$ *does not contain a triangle*

Proof. Let $\mathcal{S}_n$ be the family of sites defined above. By Lemma 4.2, $\mathcal{T}(S_n)$ is the union of the paths $a_n, b_1, b_2, b_3, c_{2n-4}, c_{2n-5}, \dots, c_1, a_0$ and $a_0, c_1, \dots, c_{2n-4}, b_3, b_2, b_1, a_n$ with the set of edges $E(A) = \{a_j a_i \mid i < j\}$. A graph with this structure satisfies the properties of the lemma. Properties (a) and (b) are satisfied, as $3n \in \Theta(n)$ and $|E(A)| = \Theta(n^2)$. The third property is true since the graph contains the Hamiltonian cycle $a_0, c_1, \dots, c_{2n-4}, b_3, b_2, b_3, a_n, \dots a_0$. For (d), we only have to argue that there are no triangles in $E(A)$ as the remaining parts of $\mathcal{T}(S_n)$ are paths. By Lemma 4.1 this is also true. □

4.2 Finding a Triangle in a Transmission Graph

Given a transmission graph $\mathcal{T}(S)$ on n sites, we want to decide if $\mathcal{T}(S)$ contains a directed triangle. We first describe an algorithm for this problem, that is not efficient when implemented naively. Then we explain how to use the underlying geometry of the problem and batched range queries to implement the algorithm in $O(n \log n)$ expected time.

The algorithm iterates over each directed edge $e = st$ with $r_t \geq r_s$, and it performs two tests: first, for each directed edge tu with $r_u \geq \frac{r_t}{2}$, it checks if us is an edge in $\mathcal{T}(S)$, i.e.,

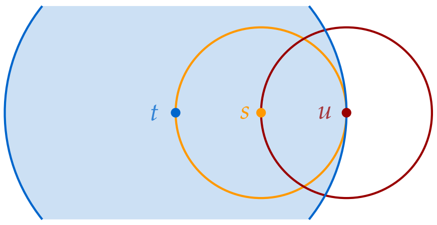


Figure 4.3: We do not need to check whether $u \in D_t$.

if $s \in D_u$. If so, the algorithm reports the triangle stu. Second, the algorithm tests if there is a site u such that $r_u \in [r_s, \frac{r_t}{2})$ and such that us is an edge in $\mathcal{T}(S)$, i.e., such that $s \in D_u$. If such a site u exists, it reports the triangle stu. If both tests fail for each edge e, the algorithm reports that $\mathcal{T}(S)$ contains no triangle. The next lemma shows that the algorithm is correct.

Lemma 4.4. *A triple stu reported by the algorithm is a triangle in $\mathcal{T}(S)$. Furthermore, if $\mathcal{T}(S)$ contains a triangle, the algorithm will find one.*

Proof. Let stu be a triple reported by the algorithm. The algorithm explicitly checks that st and us are edges in $\mathcal{T}(S)$. It remains to consider tu. If $r_u \geq \frac{r_t}{2}$, then stu is reported by the first test, and the algorithm explicitly checks that tu is an edge in $\mathcal{T}(S)$. If $r_u < \frac{r_t}{2}$, then stu is reported by the second test. We have $r_s < \frac{r_t}{2}$, since s and t are chosen so that $r_s \leq r_u < \frac{r_t}{2}$. Furthermore, st and us are edges of $\mathcal{T}(S)$, so $t \in D_s$ and $s \in D_u$. Since the second test ensures that $r_u < \frac{r_t}{2}$, the triangle inequality yields

$$\|tu\| \leq \|ts\| + \|su\| \leq r_s + r_u < \frac{r_t}{2} + \frac{r_t}{2} = r_t.$$

Thus, $u \in D_t$, and tu is an edge in $\mathcal{T}(S)$ and the reported triple stu is a triangle in $\mathcal{T}(S)$. This argument is visualized in Figure 4.3.

Next, suppose that $\mathcal{T}(S)$ contains a triangle stu, labeled such that $r_s \leq \min\{r_t, r_u\}$. If $r_u \geq \frac{r_t}{2}$, then stu is found by the first test for the edge st. If $r_u < \frac{r_t}{2}$, we have $s \in D_u$ and $r_u \in [r_s, \frac{r_t}{2})$. Thus, the second test will be successful for the edge st, and the algorithm will report a triple stu', such that $s \in D_{u'}$ and $r_{u'} \in [r_s, \frac{r_t}{2})$, where the site u' might be different from u. The first part of the proof shows that stu' is a triangle in $\mathcal{T}(S)$. □

There are several challenges for making the algorithm efficient. First of all, there might be many edges st with $r_t \geq r_s$. However, a consequence of the following lemma is, that if there are $\omega(n)$ such edges, the transmission graph $\mathcal{T}(S)$ must contain a triangle.

Lemma 4.5. *There is an absolute constant α_1 so that for any $r > 0$, if there is a square σ of diameter r that contains more than α_1 sites $s \in S$ with $r_s \geq \frac{r}{4\sqrt{2}}$, then $\mathcal{T}(S)$ has a directed triangle.*

Proof. Cover σ with a 6×6 grid of diameter $\frac{r}{6} \leq \frac{r}{4\sqrt{2}}$, see Figure 4.4. For every $s \in S \cap \sigma$ with $r_s \geq \frac{r}{4\sqrt{2}}$, the disk D_s completely covers the grid cell containing s. If σ contains more

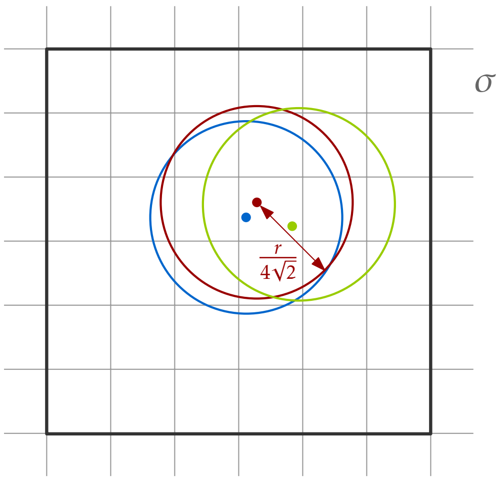


Figure 4.4: Three disks with radius at least $\frac{4}{4\sqrt{2}}$ in the same grid cell form a clique.

than $\alpha_1 = 73$ sites s with $\frac{r}{4\sqrt{2}}$, then a simple volume argument shows that one grid cell contains at least three such sites. These sites form a directed triangle in $\mathcal{T}(S)$. □

Now we define the two range searching problems that are fundamental to efficiently implementing the algorithm.

(R1) Let $\alpha > 0$ be some constant and $r_{\max} \leq 2\sqrt{2} \cdot \max\{r_s \mid s \in S\}$. Then **either** determine that for every site $s \in S$ with $r_s \leq \frac{r_{\max}}{2\sqrt{2}}$, there are at most α outgoing edges st with $r_t \geq \frac{r_s}{2}$ and report all these edges, **or** find a square σ of diameter $0 < r \leq r_{\max}$ that contains more than α sites $s \in S$ with $r_s \geq \frac{r}{4\sqrt{2}}$.

(R2) Given $O(n)$ query triples of the form (p, r_1, r_2) with $p \in \mathbb{R}^2$ and $0 < r_1 < r_2$, find a site $u \in S$ such that there is a query triple (p, r_1, r_2) with $r_1 < r_u < r_2$, and $p \in D_u$; or report that no such site u exists.

The query (R1) indeed always has a valid outcome: suppose there is a site $s \in S$ with $r_s \leq \frac{r_{\max}}{2\sqrt{2}}$ and more than α outgoing edges st with $r_t \geq \frac{r_s}{2}$. Then, all the endpoints t lie in D_s, the bounding square σ of D_s also contains these more than α sites. As σ has diameter $r = 2\sqrt{2}r_s \leq r_{\max}$ these sites have radius at least $\frac{r_s}{2} = \frac{r}{4\sqrt{2}}$ as required by the query.

The next theorem shows that we can detect a triangle in $\mathcal{T}(S)$ with linear overhead in addition to the time needed for answering (R1) and (R2).

Theorem 4.6. *If (R1) and (R2) can be solved in time $R(n)$ for input size n, we can find a directed triangle in a transmission graph $\mathcal{T}(S)$ on n sites in time $R(n) + O(n)$, if it exists.*

Proof. The general idea is to follow the general algorithm described above with some modifications to optimize the running time. Before starting to explicitly search for triangles, we perform a range query (R1) with $\alpha = \alpha_1$ as defined in Lemma 4.5 and $r_{\max} = 2\sqrt{2} \cdot \max\{r_s \mid s \in S\}$. If it reports a square σ of diameter r that contains more than α_1 sites $s \in S$ with $r_s \geq \frac{r}{4\sqrt{2}}$, we scan the sites in that square to find a set S' of $\alpha_1 + 1$ such sites. By Lemma 4.5, $\mathcal{T}(S')$ contains a triangle, which can be found in $O(1)$ time by testing all triples in S'.

Otherwise, (R1) reports the set E' of all edges st in $\mathcal{T}(S)$ with $r_t \geq \frac{r_s}{2}$, where $|E'| = O(n)$. We go through all edges $e = st$ in E' with $r_t \geq r_s$, and we check if there is an edge tu in E' such that us is an edge in $\mathcal{T}(S)$, i.e., such that $s \in D_u$. If so, we report the triangle stu. This takes care of the first test in the algorithm, and we check only $O(n)$ triples, because for each site in S, there are at most α_1 outgoing edges in E'.

If we have not been successful, we again go through all edges $e = st$ in E', and if $r_t > 2r_s$, we create the triple $(s, r_s, \frac{r_t}{2})$. We perform a range query (R2) on the resulting set of $O(n)$ triples. If (R2) finds a site $u \in S$ such that for a query triple $(s, r_s, \frac{r_t}{2})$, we have $r_s \leq r_u \leq \frac{r_t}{2}$, and $s \in D_u$, we report the triangle stu. The site u satisfies all the properties imposed by the second test. Otherwise, we report that $\mathcal{T}(S)$ does not contain a triangle. Note that here we use the fact that all radii are different, thus we can be sure that the $\leq$ condition considered for the edges of the algorithm is actually a $<$ and applying (R2) yields the correct result. By Lemma 4.4, this adapted algorithm correctly reports a triangle in $\mathcal{T}(S)$, if it exists. The time for the additional steps is $O(n)$, so the total running time is $R(n) + O(n)$. □

Using existing methods [WL85], it is easy to solve (R1) and (R2) in $O(n \log^2 n)$ time. However, a better solution is possible. In Lemmas 4.21 and 4.22, we will implement (R1) and (R2) in $O(n \log n)$ expected time, resulting in the main theorem of this section.

Theorem 4.7. *Let $\mathcal{T}(S)$ be a transmission graph on n sites. A directed triangle in $\mathcal{T}(S)$ can be found in expected time $O(n \log n)$, if it exists.*

Proof. The statement follows from combining Theorem 4.6 and Lemmas 4.21 and 4.22. □

4.3 Finding the Shortest Triangle in a Transmission Graph

We extend Theorem 4.7 to find the shortest triangle in $\mathcal{T}(S)$. As in Section 3.1, we solve the decision problem: *given $W > 0$, does $\mathcal{T}(S)$ have a directed triangle of perimeter at most W?* and then use Chan's randomized optimization framework to obtain an algorithm for the optimization problem. We call a site $s \in S$ *large* if $r_s > \frac{W}{4}$. Again let $S_\ell \subseteq S$ be the set of all large sites.

Similarly to the algorithm for the weighted girth in disk graphs, there are several cases to consider, in order to find the shortest triangle. An overview over the complete algorithm can be found in Figure 4.5.

Lemma 4.8. *We can find a triangle in $T(S \setminus S_\ell)$ of perimeter at most W in $O(n \log n)$ time, if it exists.*

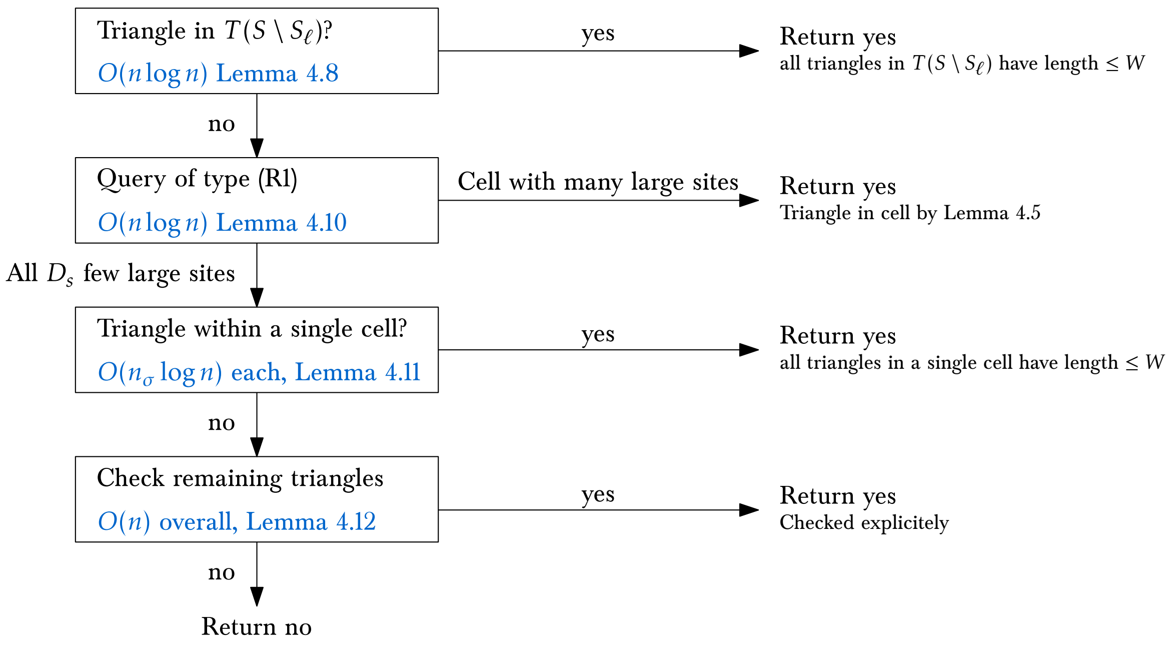

Triangle in $T(S \setminus S_\ell)$?
$O(n \log n)$ Lemma 4.8
yes
Return yes
all triangles in $T(S \setminus S_\ell)$ have length $\leq W$
no
Query of type (R1)
$O(n \log n)$ Lemma 4.10
Cell with many large sites
Return yes
Triangle in cell by Lemma 4.5
All D_s few large sites
Triangle within a single cell?
$O(n_\sigma \log n)$ each, Lemma 4.11
yes
Return yes
all triangles in a single cell have length $\leq W$
no
Check remaining triangles
$O(n)$ overall, Lemma 4.12
yes
Return yes
Checked explicitely
no
Return no

Figure 4.5: An overview of the algorithm to find the shortest triangle in a transmission graph.

Proof. Any triangle in $T(S \setminus S_\ell)$ has perimeter at most W: consider a directed triangle stu in $T(S \setminus S_\ell)$ with $r_s \geq \max\{r_t, r_u\}$. Since s has the largest radius of the sites on the triangle, we have $r, u \in D_s$. This implies that $\|st\|, \|us\| \leq r_s$ and $\|tu\| \leq 2r_s$. Then the perimeter of the triangle is at most $\|st\| + \|tu\| + \|us\| \leq 4r_s \leq W$. We can find a triangle in $T(S \setminus S_\ell)$ in $O(n \log n)$ time by Theorem 4.7. □

The task that remains, is to check for triangles of perimeter at most W with at least one large site. Some of these triangles have to be considered individually, while the others can be handled efficiently in batch mode. The following lemmas show that we either find a triangle or we may assume that there are few edges from $S \setminus S_\ell$ to S_ℓ.

Lemma 4.9. *If $\mathcal{T}(S)$ does not contain any triangle, each site t has at most six incoming edges from sites s with $s \in D_t$.*

Proof. We consider the set of circular sectors with opening angle $\frac{2\pi}{6}$ with common apex t that cover D_t. Assume for the sake of contradiction that one of these six circular sectors contains two sites s, s' such that st and $s't$ are both edges in $\mathcal{T}(S)$. Without loss of generality, we assume that s' is contained in the equilateral triangle defined by the boundary of the circular sector and the line perpendicular to its bisector and through s, see Figure 4.6. Then as this triangle is equilateral we have $\|ss'\| \leq \|st\| \leq r_s$ and thus ss' is an edge in $\mathcal{T}(S)$. This edge closes the triangle $ss't$ where the edge ts exists as $s \in D_t$ by assumption. □

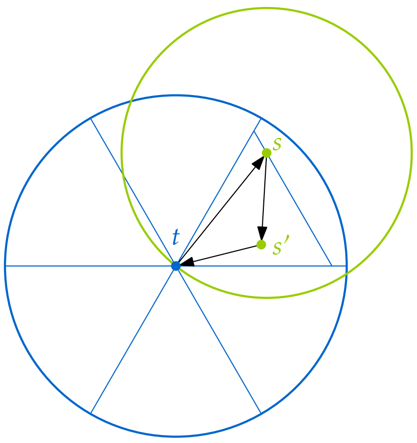


Figure 4.6: If st is an edge in $\mathcal{T}(S)$ then the triangle tss' exists.

Lemma 4.10. *If $\mathcal{T}(S)$ does not have a triangle of perimeter at most W, every site in S_ℓ has at most six incoming edges from $S \setminus S_\ell$. Furthermore, in $O(n \log n)$ time, we can either find a triangle of perimeter at most W in $\mathcal{T}(S)$ or determine for each site in S_ℓ all incoming edges from $S \setminus S_\ell$.*

Proof. Since all sites in S_ℓ have a larger radius than the sites in $S \setminus S_\ell$, Lemma 4.9 implies the first part of the claim.

As stated above, to find the triangles, we will handle some cases efficiently in batch mode. For this we will again use (R1). Suppose there is a square σ in the plane with diameter $0 < r \le \frac{W}{\sqrt{2}}$ such that σ contains more than α_1 sites s of radius $r_s \ge \frac{r}{4\sqrt{2}}$, where α_1 is defined as in Lemma 4.5. Then, Lemma 4.5 shows that $\mathcal{T}(S)$ contains a triangle that lies in σ. In particular by the proof of Lemma 4.5 this triangle lies inside a square with diameter $\frac{r}{6}$ and thus has perimeter at most $3 \cdot \frac{r}{6} < W$.

If there is no such square, it follows that there is no site s with $r_s \le \frac{W}{4}$ such that D_s contains more than α_1 sites of radius at least $\frac{r_s}{2}$, as otherwise s could be enclosed by a square of side length $2\sqrt{2}r_s \le \frac{W}{\sqrt{2}}$ that contains many sites of large radius. Thus, every site s with $r_s \le \frac{W}{4}$ has $O(1)$ outgoing edges to sites with radius at least $\frac{r_s}{2}$. In particular, there are $O(n)$ edges from small sites to large sites in $\mathcal{T}(S)$.

Using (R1) with $\alpha = \alpha_1$ and $r_{\max} = \frac{W}{\sqrt{2}}$ we can distinguish these cases. In the first case the square reported by (R1) contains a triangle of perimeter at most W and we are done. In the second case, we obtain all edges from $S \setminus S_\ell$ to S_ℓ.

Now assume there is a site $s \in S_\ell$ that has indegree more than six from sites in $S \setminus S_\ell$. Then by Lemma 4.9 there is a triangle induced by s and two sites $t, u \in S \setminus S_\ell$. Assume without loss of generality that this triangle has the form sut. Then by the definition of transmission graphs, we have $\|ut\| \le r_u$ and $\|ts\| \le r_t$, and in particular $\|ut\| + \|ts\| \le 2 \cdot \frac{W}{4}$. Since $\|su\| \le \|ut\| + \|ts\| \le 2 \cdot \frac{W}{4}$ by triangle inequality, this implies that the perimeter of the induced triangle is at most W. Checking the $O(1)$ sites which have an incoming edge to s explicitly will either find the at most six sites, or report a triangle. $\square$

Next, we limit the number of relevant edges between large sites. For this, we subdivide the plane with a grid $\mathcal{G}_{\mathrm{I}}$ with cell diameter $\frac{W}{4}$. We get the following lemma:

Lemma 4.11. *A triangle contained in a cell $\sigma \in \mathcal{G}_{\mathrm{I}}$ has perimeter at most W. If there is no triangle in σ, then σ contains $O(1)$ large sites. Such triangles can be found in $O(n \log n)$ overall expected time.*

Proof. An upper bound for the perimeter of a triangle contained in σ is $3 \cdot \frac{W}{4} < W$. Furthermore, if there are at least three large sites in σ, these large sites form a triangle, since the disk of a large site covers σ. By applying Theorem 4.7 to the induced subgraph in each cell of $\mathcal{G}_{\mathrm{I}}$, we can find such a triangle in $O(n \log n)$ total expected time. □

Let t be a site and let σ be the cell containing t, then the neighborhood $N(t)$ of t are all sites contained in $N_{7\times 7}(\sigma)$. By again applying Lemma 2.1, similar to the girth in disk graphs, any triangle of perimeter at most W is contained in $N(t)$.

Lemma 4.12. *If each site in $S \setminus S_\ell$ has at most six incoming edges from S_ℓ and these edges are known, a triangle with at least one large site and perimeter at most W can be found in $O(n)$ overall time.*

Proof. Consider such a triangle tus with $r_s \geq \max\{r_u, t_s\}$. Then, $s \in S_\ell$, and s, t, u all lie in $N(t)$. By Lemma 4.10, there are $O(1)$ small candidates for u, and by Lemma 4.11, there are $O(1)$ large candidates for u in $N(t)$. Having fixed a site t and a possible candidate u, we iterate over all $s \in N(t)$ and check if s, u, and t form a triangle with perimeter at most W. Every site s is contained in $O(1)$ grid neighborhoods, and since there are $O(1)$ candidate pairs in each grid neighborhood, s participates in $O(1)$ explicit checks. The result follows. □

The following theorem summarizes the considerations in this section.

Theorem 4.13. *We can find the shortest triangle in $O(n \log n)$ expected time, if it exists.*

Proof. Combining Lemmas 4.8, 4.10 and 4.12 we get an algorithm for the decision version of the problem. As in Theorem 3.7, the result follows from the same application of Chan's randomized optimization technique [Cha99] (restated in Lemma 3.6). □

4.4 Finding a Cycle of Length k in a Transmission Graph

In this section we aim to find a cycle with at most k sites in a transmission graph. For $k = 3$ the problem can be solved using Theorem 4.7. So from now on, we can focus our attention on the case that $\mathcal{T}(S)$ is triangle-free. Recall from Theorem 4.3 that this graph might still have $\Theta(n^2)$ edges, so simply running the algorithm from Theorem 3.10 with each vertex as a starting vertex is not faster than on general graphs.

We will once again use the geometry of our setting to find a better algorithm. In contrast to the algorithm for the girth in disk graphs, we assume that an upper bound k on the

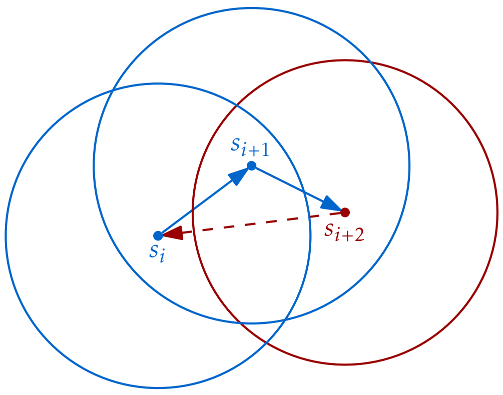


Figure 4.7: As $s_{i+2}s_i$ is not an edge in $\mathcal{T}(S)$, the radius of s_{i+2} is small compared to the sum of the radii of s_i and s_{i+1}.

unweighted length of the cycle is given to the algorithm as a parameter. The running time then has an exponential dependency on k. For sufficiently small values depending on n or constant values of k this is still an improvement over the naive approach.

Before describing the algorithm we make some structural observations that are crucial for showing the correctness of the algorithm. In the following assume, that a cycle $C = s_0, \dots, s_{k-1}$ of length exactly k exists in $\mathcal{T}(S)$ and that the sites are named such that $r_{s_0} = \max\{r_{s_0}, \dots, r_{s_{k-1}}\}$. The results carry over to cycles with length at most k.

Lemma 4.14. *Let $C = s_0, \dots, s_{k-1}$ be a cycle with length $k \geq 4$ and assume that $\mathcal{T}(S)$ does not contain any triangles. Then the following holds, where all indices are interpreted modulo k.*

(a) $r_{s_{i+2}} < r_{s_i} + r_{s_{i+1}}$

(b) $\max\{r_{s_i}, r_{s_{i+1}}\} \geq \frac{r_{s_0}}{2^{k-i-1}}$*; and*

(c) $\max\{r_{s_i}, r_{s_{i+1}}\} \geq \frac{r_{s_0}}{2^k}$.

Proof. The first claim follows from a combination of the triangle inequality with the definition of $\mathcal{T}(S)$. The triangle inequality implies that $\|s_i s_{i+2}\| \leq \|s_i s_{i+1}\| + \|s_{i+1} s_{i+2}\|$. As $s_i s_{i+1}$ and $s_{i+1} s_{i+2}$ are edges in $\mathcal{T}(S)$, this reduces to $\|s_i s_{i+2}\| \leq r_{s_i} + r_{s_{i+1}}$. Finally, as $\mathcal{T}(S)$ contains no triangles, $s_{i+2}s_i$ is not an edge in the transmission graphs, and thus $\|s_i s_{i+2}\| > r_{s_{i+2}}$. Combining these bounds yields $r_{s_{i+2}} < r_{s_i} + r_{s_{i+1}}$ as claimed, see Figure 4.7.

As (c) is a direct consequence of (b), it remains to show (b). For $i = k-1$ we have $\max\{r_{s_{k-1}}, r_{s_0}\} \geq \frac{r_{s_0}}{2^{k-i-1}}$, since $r_{s_0} = \frac{r_{s_0}}{2^0}$. For the remaining terms we use inverse induction on $i = k-2, \dots, 0$. For the base case $i = k-2$ this is the statement $\max\{r_{s_{k-2}}, r_{s_{k-1}}\} \geq \frac{r_{s_0}}{2}$, which follows directly from (a):

$$r_{s_0} < r_{s_{k-2}} + r_{s_{k-1}}$$
$$r_{s_0} \leq 2\max\{r_{s_{k-2}}, r_{s_{k-1}}\}$$

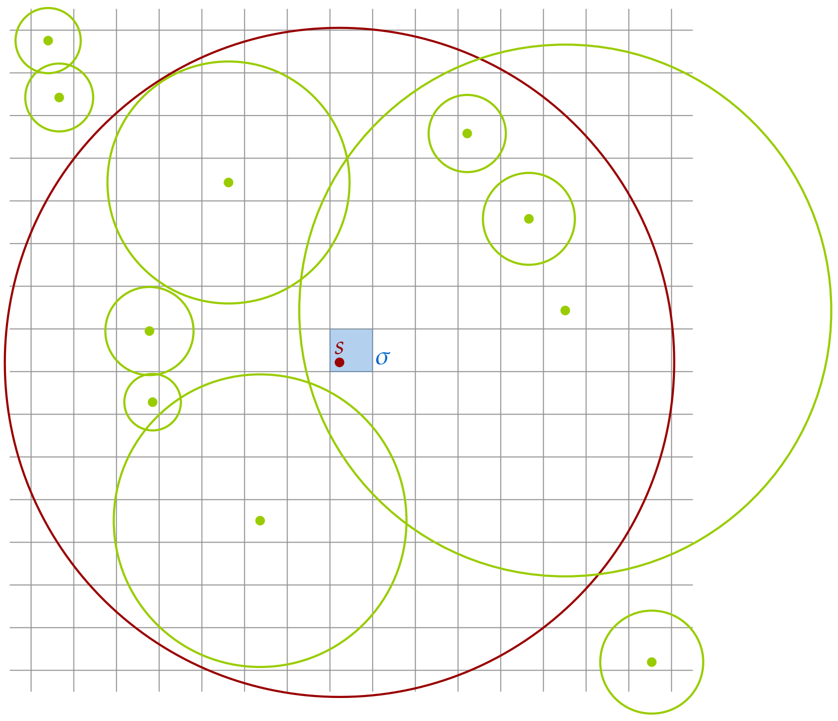


Figure 4.8: (Part of) $N(s)$ together with the corresponding part of $S_1(s)$.

The step now goes from $i+1$ to i. By the hypothesis we have

$$\max\{r_{s_{i+1}}, r_{s_{i+2}}\} \geq \frac{r_{s_0}}{2^{k-i-2}}.$$

Now we distinguish two cases. If $r_{s_{i+1}} \geq \frac{r_{s_0}}{2^{k-i-2}}$ the claim is immediate, as $\frac{r_{s_0}}{2^{k-i-2}} \geq \frac{r_{s_0}}{2^{k-i-1}}$. In the other case $r_{s_{i+2}} \geq \frac{r_{s_0}}{2^{k-i-2}}$ and by using (a) the result again follows:

$$\begin{aligned} \frac{r_{s_0}}{2^{k-i-2}} &\leq r_{s_{i+2}} \\ &< r_{s_i} + r_{s_{i+1}} \\ &\leq 2\max\{r_{s_i}, r_{s_{i+1}}\} \end{aligned}$$ □

Now we are ready to describe the algorithm. Similar to Section 4.2 the algorithm is first described in a general fashion, focusing on a single site s. Then we show that the algorithm can be implemented efficiently using batched range queries.

For each $s \in S$ the goal is to find a cycle C_s with at most k vertices where s is the site with the largest radius among the sites of C_s. The general idea of the algorithm is to collect all sites that can be part of C_s, explicitly compute the transmission graph $\mathcal{T}_s$ on this set and then use an algorithm for general graphs on $\mathcal{T}_s$.

Let $l(s) = \max\{0, \lfloor \log \frac{r_s}{2^k} \rfloor\}$ and consider the hierarchical grid at level $l(s)$. Let σ be the cell of $\mathcal{G}_{l(s)}$ with $s \in \sigma$. Furthermore, let $N(s)$ be the $\left(2 \cdot \left\lceil k \cdot 2^k \cdot \sqrt{2} \right\rceil + 1\right) \times$

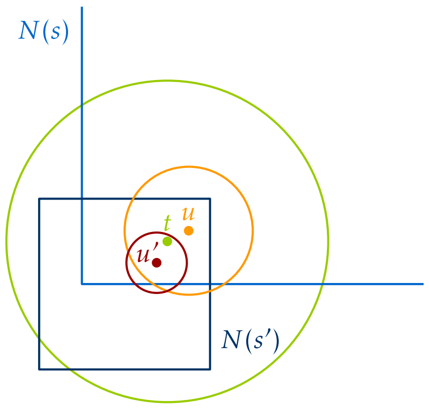


Figure 4.9: The site u' is in both $S_2(t,s)$ and $S_2(t,s')$. The site u is only in $S_2(t,s)$ as it is too large for $S_2(t,s')$. In this case we have $s_{\max}(t) = s$.

$\left(2\cdot\left\lceil k\cdot 2^k\cdot\sqrt{2}\right\rceil+1\right)$ neighborhood of σ. The algorithm now collects a set $S_1(s)$ of sites that contains all sites of radius at least $2^{l(s)}$ in $N(s)$, see Figure 4.8. Furthermore, for a fixed $t\in S_1(s)$ let $S_2(t,s)$ be the set of all sites $u\in S$ with $r_u\leq 2^{l(s)}$ such that the edge ut exists in $\mathcal{T}(S)$. Note that a site t can be in the set $S_1(s)$ of multiple sites s. However as for a fixed t the sites in $S_2(t,s)$ are only determined by r_s, we have $S_2(t,s)\subseteq S_2(t,s')$ for $r_s\leq r_{s'}$. Taking advantage of this observation, we denote by $s_{\max}(t)$ the largest site s such that $t\in S_1(s)$ and by $S_2(t)=S_2(t,s_{max}(t))$ the cardinally largest set defined by t, see Figure 4.9 for an illustration.

After finding all sets $S_1(s)$, we find the set $S_2(t)$ for each $t\in S$. Finally the shortest cycle is found explicitly in the graph $\mathcal{T}_s=\mathcal{T}\left(S_1(s)\cup\bigcup_{t\in S_1(s)}S_2(t)\right)$. If this cycle has length at most k the algorithm stops and reports it.

Lemma 4.15. *The algorithm described above correctly finds a cycle with at most k edges.*

Proof. First of all, if the algorithm reports a cycle, it is immediate that this cycle is a correct output for our algorithm. So the main challenge is to show, that if there is a cycle C_s with at most k edges and largest site s, the algorithm will report it. Let $C_s=(s=s_0),\dots,s_{k'}$ with $k'\leq k-1$ be this cycle. A simple distance argument shows that all sites on C_s are contained in $D(s,2kr_s)$. Furthermore by Lemma 2.1, $D(s,2kr_s)$ is completely contained in $N(s)$ and thus no sites on C_s with radius at least $2^{l(s)}$ are missed by restricting the sites considered for $S_1(s)$ to those in $N(s)$.

What is left to show is that all s_i in C_s are contained in $\mathcal{T}_s$. If $r_{s_i}\geq 2^{l(s)}$ then $s_i\in S_1(s)$ by definition. In the other case, recall that Lemma 4.14 (c) states that $\max\{r_{s_i},r_{s_{i+1}}\}\geq\frac{r_s}{2^k}$. In particular, by the assumption made in Chapter 2 that each radius is at least 1, this implies $\max\{r_{s_i},r_{s_{i+1}}\}\geq 2^{l(s)}$. Thus if $r_{s_i}<2^{l(s)}$ we have $r_{s_{i+1}}\geq 2^{l(s)}$, or equivalently $s_{i+1}\in S_1(s)$. As s_is_{i+1} is an edge of the cycle, by the definition of $S_2(s_{i+1})$ we have $s_i\in S_2(s_{i+1})$ and by consequence also $s_i\in\mathcal{T}_s$. Thus all sites on the cycle are contained in $\mathcal{T}_s$ and we cannot miss C_s. □

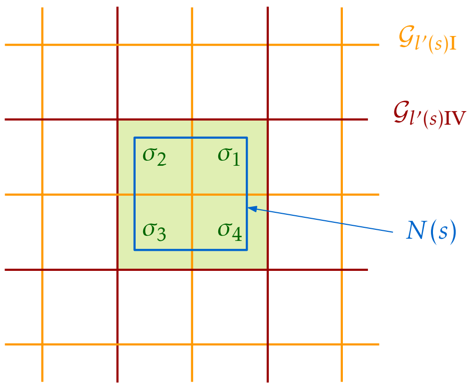

$\mathcal{G}_{l'(s)\mathrm{I}}$
$\mathcal{G}_{l'(s)\mathrm{IV}}$
σ_2
σ_1
σ_3
σ_4
$N(s)$

Figure 4.10: $N(s)$ is contained in $\mathcal{G}_{l'(s)\mathrm{IV}}$.

Now, that we know that the general strategy is correct, we move our focus to efficiently implementing it. Similar to the algorithm described in Section 4.2 the main challenge is to efficiently find the sets $S_1(s)$ and $S_2(t)$ efficiently.

First, observe that if $\mathcal{T}(S)$ contains no triangle the number of sites considered for $S_1(s)$ in each cell of $N(s)$ is at most two. Assume to the contrary that there is a cell $\sigma \in N(s)$ containing at least three sites of radius at least $2^{l(s)}$. Then as $r_s \geq 2^{l(s)} = |\sigma|$ the sites are all pairwise contained in each other, forming a clique and thus also a triangle. So from now on we can assume that each cell in $N(s)$ contributes at most two sites to $S_1(s)$.

Lemma 4.16. *If $\mathcal{T}(S)$ does not contain a triangle, the sets $S_1(s)$ for all $s \in S$ can be found in overall time $O(n \log n) + n \cdot 2^{O(k)}$. Additionally, the values $s_{\max}(t)$ can be determined for all $t \in S$ within the same time bound.*

Proof. Let $\mathcal{Q}^c$ be the compressed quadtree of S and assume without loss of generality that the leaves of $\mathcal{Q}^c$ have diameter 1. Furthermore, augment $\mathcal{Q}^c$ by a bottom-up traversal such that each cell τ is associated with the set $\mathcal{A}_\tau$ of the at most two sites that lie in τ and have radius at least $|\tau|$. Let $\sigma \in N(s)$ be a cell in $N(s)$ and let ρ be the largest cell in $\mathcal{Q}^c$ such that $\rho \subseteq \sigma$. Recall from Lemma 2.3 that this is equivalent to saying that ρ is the Z-predecessor of σ. Denote by R the union of all cells defined this way. By the assumption on the diameter of the leaves, and as $l(s) \geq 0$ for each $s \in S$, we have $S \cap N(s) = S \cap \bigcup_{\rho \in R} \{s \in \rho\}$. The task of computing $S_1(s)$ then reduces to locating the cells R in $\mathcal{Q}^c$ and setting $S_1(s) = \bigcup_{\rho \in R} \{t \in A_\rho \mid r_t \geq 2^{l(s)}\}$.

One easy way of finding the set R would be to collect the cells in $\bigcup_{s \in S} N(s)$, sort them by Z-order and merge them with the linearized quadtree $\mathcal{L}$. By Lemma 2.3 this would yield for each $\sigma \in N(s)$ its Z-predecessor and thus exactly the set R. As each neighborhood contains $2^{O(k)}$ cells the sorting would take an overall of $n2^{O(k)} \log(n2^{O(k)})$ time, which exceeds the claimed time bound. There are however some geometric observations which allow us to reduce the cells which have to be sorted to $O(n)$ and thus replace the $\log(n2^{O(k)})$ factor by an additive $O(n \log n)$ term. To locate the cells in R, note that $N(s)$ can also be seen as a square of diameter $\operatorname{diam}(N(s)) = \left(2\left\lceil k \cdot 2^k \sqrt{2}\right\rceil + 1\right) \cdot 2^{l(s)}$. Let $l'(s)$ be the level of the hierarchical grid, such that $\operatorname{diam}(N(s)) \leq 2^{l'(s)} \leq 2 \operatorname{diam}(N(s))$ and denote this level

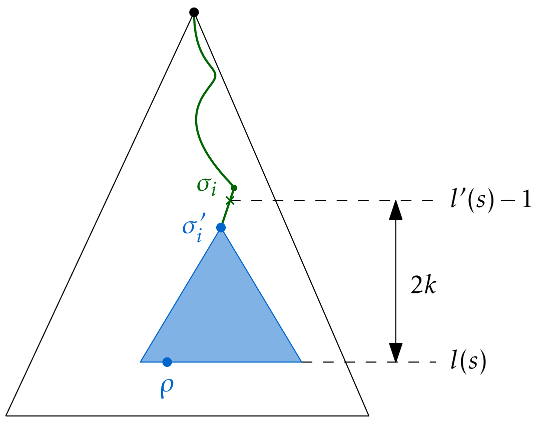


Figure 4.11: The cell σ_i' is the first descendant of σ_i in $\mathcal{Q}^c$. There are $2k$ levels between σ_i and ρ.

of the hierarchical grid by $\mathcal{G}_{l'(s)\mathrm{I}}$. Let ℓ be the side length of a cell in $\mathcal{G}_{l'(s)\mathrm{I}}$ and consider three shifted copies of $\mathcal{G}_{l'(s)\mathrm{I}}$ such that $(\ell/2,0),(0,\ell/2)$ and $(\ell/2,\ell/2)$ are grid points of $\mathcal{G}_{l'(s)\mathrm{II}},\mathcal{G}_{l'(s)\mathrm{III}}$ and $\mathcal{G}_{l'(s)\mathrm{IV}}$, respectively.

Then there is an $i \in \{\mathrm{I},\dots,\mathrm{IV}\}$ such that $N(s)$ is completely contained in a grid cell σ of $\mathcal{G}_{l'(s)i}$. Let $\sigma_1,\dots,\sigma_4$ be the four cells of diameter $2^{l'(s)-1}$ that partition σ. As $\mathcal{G}_{l'(s)i}$ is shifted by $\ell/2$, the cells $\sigma_1,\dots,\sigma_4$ are part of the hierarchical grid in level $l'(s)-1$, see Figure 4.10. The cells $\sigma_1,\dots,\sigma_4$ might not be in $\mathcal{Q}^c$ because they were removed due to compression. However, if they contain at least one site, there are cells $\sigma_1',\dots,\sigma_4' \in \mathcal{Q}^c$ that are the largest cells such that $\sigma_i' \subseteq \sigma_i$, see Figure 4.11. As $s \in N(s)$ and there is a cell in $\mathcal{Q}^c$ containing s with diameter 1, at least one of the cells σ_i' is not empty. Furthermore by the definition of the compressed quadtree, the union of these four cells contains all sites that are contained in $N(s)$. We will use these cells as a starting point of an explicit traversal of $\mathcal{Q}^c$ to find the cells in R.

The cells $\sigma_1',\dots,\sigma_4'$ for each $s \in S$ in $\mathcal{Q}^c$ can be found in a batched fashion. For all $s \in S$, the four tuples (s,σ_i) can be computed in constant time for a fixed s. Collecting these tuples for all $s \in S$ and sorting them in Z-Order of their second component takes overall $O(n\log n)$ time. Merging the linearized quadtree $\mathcal{L}$ of $\mathcal{Q}^c$ with the sorted list of cells, yields for each σ_i in a query tuple its Z-predecessor. By Lemma 2.3 the Z-predecessor of σ_i is exactly σ_i', if σ_i' exists. Thus all tuples (s,σ_i') can be found in $O(n)$ time during the merging process and then be grouped by their first entry in $O(n\log n)$ additional time.

Having the cells σ_i' for each $s \in S$ at hand, the set R and thus the sites in $S_1(s)$ can be found as follows: Starting with each σ_i', the compressed quadtree $\mathcal{Q}^c$ is traversed downwards by visiting all cells with a non-empty intersection with $N(s)$. The downward traversal stops, if the currently visited cell has a diameter smaller than or equal to $2^{l(s)}$ or if it has no children. If the traversal stops in a cell ρ, all sites stored in A_ρ with radius at least $2^{l(s)}$ are added to $S_1(s)$. This mirrors the characterization of $S_1(s)$ given above.

This process visits at most $l'(s) - l(s)$ levels. We can bound this quantity as follows:

$$\begin{aligned} l'(s) - l(s) &= \log\left(\frac{2^{l'(s)}}{2^{l(s)}}\right) \\ &\le \log\left(\frac{(2\lceil k \cdot 2^k \sqrt{2}\rceil + 1) \cdot 2^{l(s)}}{2^{l(s)}}\right) \\ &\le \log\left(2\lceil k \cdot 2^k \sqrt{2}\rceil + 1\right) \\ &\le 2k && \text{for } k \ge 4 \end{aligned}$$

Thus the cells traversed when starting at σ_i' form a subtree of $\mathcal{Q}^c$ of height at most $2k$, and overall at most $2^{O(k)}$ cells are visited, again see Figure 4.11. Traversing all four subtrees defined in this way and reporting all sites with the appropriate radius stored with the leaves of these subtrees takes $2^{O(k)}$ time for each site s.

To summarize, after a preprocessing step of $O(n \log n)$ time, the sets $S_1(s)$ can be reported in $2^{O(k)}$ for each $s \in S$. When doing the traversals in decreasing order of the radii of the sites, we can find the sites $s_{\max}(t)$ for each $t \in S$ on the fly. After computing the set $S_1(s)$ and storing it with s, check for each $t \in S_1(s)$ if it was reported for the first time. If this is the case, set $s_{\max}(t) = s$ and ignore it otherwise. □

Next we focus on the sets $S_2(t)$ and describe an efficient algorithm for finding those. For this we will use the following a variant of (R2):

(R2') Given a set $R \subseteq S \times \mathbb{R}^+$ of $O(n)$ query tuples, report for each $u \in S$ one of the query tuples $(t, r) \in R$ such that $t \neq u$, $t \in D_u$ and $r_u < r$, if it exists.

Lemma 4.17. *Assume that queries of (R2') can be answered in time $R(n)$ and that $\mathcal{T}(S)$ does not contain any triangles. Then the sets $S_2(t)$ for all $t \in S$ can be found in overall time $O(n \log n) + R(n)$.*

Proof. Let $t \in S$ be a fixed site and recall that we defined $s_{\max}(t)$ to be the largest site s such that $t \in S_1(s)$. The set $S_2(t)$ now contains all sites $u \in S$ with $r_u < 2^{l(s_{\max}(t))}$ and $t \in D_u$. All sites in $S_1(s_{\max}(t))$ have radius a least $2^{l(s_{\max}(t))}$, thus we have $r_u \le r_t$. Furthermore, $S_2(t)$ contains at most six sites by Lemma 4.11. We find the sets $S_2(t)$ in two steps. First, we identify for each $t \in S$ the sites $u \in S_2(t)$ with $\frac{1}{2} r_t < r_u$. In a second step, the remaining sites in $S_2(t)$ are found. Both steps are performed in a batched fashion, finding the relevant sites in $S_2(t)$ for all sites $t \in S$ during one computation.

The approach in the first case is similar to that used in the proof of Lemma 4.16. Again we construct a compressed quadtree $\mathcal{Q}^c$ on S, assume that the leaves have diameter 1, and augment each cell $\sigma \in \mathcal{Q}^c$ with the set A_σ of at most two sites with radius at least $|\sigma|$. Recall that if $t \in S_1(s_{\max}(t))$, then $r_t \ge 2^{l(s_{\max})}$ and that all sites in $S_2(t)$ have a radius smaller than $2^{l(s_{\max})}$. Observe that a site $u \in S_2(l)$ with $r_u > \frac{1}{2} r_t$ can only exist, if $r_t \le 2^{l(s_{\max})+1}$. Furthermore as $r_t \ge 2^{l(s_{\max}(t))}$, the assumption $r_u > \frac{1}{2} r_t$ directly implies $r_u \ge 2^{l(s_{\max}(t))-1}$, see Figure 4.12.

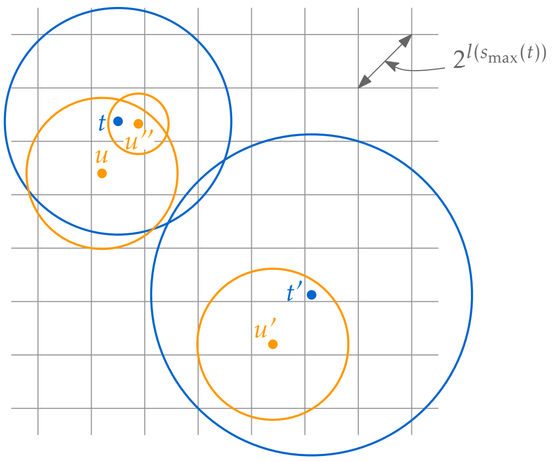


Figure 4.12: The site u falls in the first categories of sites in $S_2(t)$. The radius of u'' is to small to fall in this category. The radius of t' is too large to have any site in the first category in $S_2(t')$.

Consider the hierarchical grid on level $l(s_{\max}(t)) - 1$ and let τ be the cell in this level containing t. By Lemma 2.1 the disk D_t is completely contained in $N_{13\times 13}(\tau)$. Finding the sites u in $S_2(t)$ with $r_u \geq \frac{1}{2} r_t$ now reduces to finding the Z-predecessors of all cells in $N_{13\times 13}(\tau)$ with subsequent reporting of the relevant sites. Finding the Z-predecessors can again be achieved by sorting the $O(n)$ cells from all 13×13 neighborhoods by Z-order and then merging them with the linearized quadtree $\mathcal{L}$. After having the Z-predecessors at hand, we filter each set A_σ and add the sites containing t and having the appropriate radii to $S_2(t)$.

For the second case, we first observe, that there is only one site t such that $u \in S_2(t)$ and $r_u \leq \frac{1}{2} r_t$. This can be seen by the following application of the triangle inequality, see Figure 4.13. Assume that u is both in $S_2(t)$ and in $S_2(t')$. Then the edges ut and ut' exist in $\mathcal{T}(S)$ and therefore also $\|ut'\| \leq r_u$ and $\|ut\| \leq r_u$. As $r_u \leq r_t$ and $r_u \leq r_{t'}$, this also directly implies the existence of the edges tu and $t'u$. Furthermore, the triangle inequality

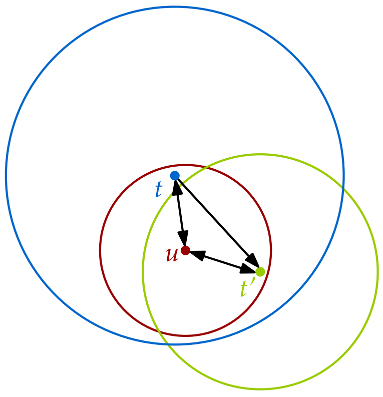


Figure 4.13: The site u is contained in at most one set $S_2(t)$.

gives $\|tt'\| \leq 2r_u \leq r_t$ and thus the edge tt' also exists in $\mathcal{T}(S)$ and closes a triangle, a contradiction to our assumption on $\mathcal{T}(S)$. Thus all that needs to be done is to find for each u the site t with $r_u \leq \frac{r_t}{2}$ and $t \in D_u$, if it exists as in this case $u \in S_2(t)$. This kind of query can be handled in a batched fashion using (R2') by considering a query tuple $(t, \frac{r_t}{2})$ for each $t \in S$.

The time needed in the first step is dominated by the construction of $\mathcal{Q}^c$ and the sorting of the cells. Both steps need $O(n \log n)$ time. The second steps takes time $O(R(n))$ after a linear preprocessing step.

□

In Lemma 4.23 we will see that (R2') can be answered in $O(n \log n)$ expected time. Combining this with Lemmas 4.16 and 4.17, the running time of the algorithm follows.

Theorem 4.18. *A cycle on at most k vertices in a transmission graph can be found in $O(n \log n) + n \cdot 2^{O(k)}$ expected time, if it exists.*

Proof. First we can check if $\mathcal{T}(S)$ contains a triangle in $O(n \log n)$ expected time by Theorem 4.7. If a triangle is found we have found a cycle with at most k vertices and are done.

In the other case, we can use the algorithm described above. As each cell of $N(s)$ contributes $O(1)$ sites to $S_1(s)$ and for each $t \in S_1(s)$ again only $O(1)$ additional sites are added to $\mathcal{T}_s$ because they are in $S_2(t)$, this subgraph of $\mathcal{T}(S)$ consists of $2^{O(k)}$ sites and consequently also $2^{O(k)}$ edges with a slightly larger constant in the exponent. Explicitly constructing $\mathcal{T}_s$ and finding the shortest unweighted cycle takes time proportional to the size of $\mathcal{T}_s$. As $|\mathcal{T}_s| = 2^{O(k)}$ this takes $2^{O(k)}$ time for each $\mathcal{T}_s$ and thus $n2^{O(k)}$ time in total. Combining this bound with the preprocessing time detailed in Lemmas 4.16 and 4.17 the overall time bound follows. □

Corollary 4.19. *Given a transmission graph with unweighted girth k. Then the girth can be computed in time $O(n \log n \log k) + n2^{O(k)}$.*

Proof. Using exponential search, the value k can be determined by $O(\log k)$ applications of Theorem 4.18. □

4.5 Batched Range Searching

After having seen their algorithmic applications, we can now focus on the main technical challenge of this chapter: to solve the range queries described above efficiently. For this we use the general range query data structure described in Section 2.4 as a basis. For all three types of range queries the sites S, sorted in their radii, are stored in the leaves of the binary search tree B. The radii of the sites which are relevant to answer the queries of (R1) and (R2') are only bounded from one side. So the relevant paths in B for (R1) are only the paths $\Pi_{\leftarrow}(\frac{r_s}{2})$ and for (R2') the paths $\Pi_{\leftarrow}(r)$ for the radiii given in the query tuple. On the other hand, the radius range for (R2) is bounded by the interval (r_1, r_2) and thus both paths $\Pi_{\leftarrow}(r_1)$ and $\Pi_{\rightarrow}(r_2)$ are considered.

4.5.1 Queries of Type (R1)

We restate the objective for the queries of type (R1) for convenience.

(R1) Let $\alpha > 0$ be some constant and $r_{\max} \leq 2\sqrt{2}\cdot\max\{r_s \mid s \in S\}$. Then **either** determine that for every site $s \in S$ with $r_s \leq \frac{r_{\max}}{2\sqrt{2}}$, there are at most α outgoing edges st with $r_t \geq \frac{r_s}{2}$ and report all these edges, **or** find a square σ of diameter $0 < r \leq r_{\max}$ that contains more than α sites $s \in S$ with $r_s \geq \frac{r}{4\sqrt{2}}$.

We answer the query for a given set of sites efficiently by checking the properties for various subsets of sites $s \in S$ simultaneously. Let B be the binary search tree on S, sorted by the radii as used in the range query structure introduced in Section 2.4. For this we will use linearized compressed quadtrees $\mathcal{L}_v$ for all canonical subsets of the sites. Each $\mathcal{L}_v$ is the linearized version of the compressed quadtree $\mathcal{Q}_v^c$. The quadtrees $\mathcal{Q}_v^c$ contain all sites in the canonical subset defined by a vertex v of B. We will show in Lemma 4.20 that it is possible to build all these linearized compressed quadtrees without logarithmic overhead.

Since (R1) gives us plenty of freedom in choosing the squares reported by the range queries, we take the enclosing squares of the sites in addition to the cells of the hierarchical grid. The latter allows us to reduce the range searching problem to that of a predecessor search in a linear list. The reduction process, described in detail in Lemma 4.21, can be accomplished by a constant number of tree traversals.

Lemma 4.20. *For each $v \in B$ the linearized quadtree $\mathcal{L}_v$ for the sites in the canonical subset $\mathcal{I}_v$ can be found in $O(n \log n)$ time.*

Proof. For each $v \in B$, the compressed quadtree $\mathcal{Q}_v^c$ for $\mathcal{I}_v$ is build as follows. At the root, the compressed quadtree $\mathcal{Q}^c$ for S is computed in $O(n \log n)$ time [Buc+11; Har11]. Given the compressed quadtree $\mathcal{Q}_v^c$ for a node $v \in B$, we compute $\mathcal{Q}_w^c$ for a child w of v using a postorder traversal of Q_v. In each leaf σ of $\mathcal{Q}_v^c$, check if the sites in σ are in $\mathcal{I}_w$, by looking at their radii. If none of the sites lie in $\mathcal{I}_w$, σ is removed; otherwise, it remains in $\mathcal{Q}_w^c$. If an inner vertex σ of $\mathcal{Q}_v^c$ has no remaining children it is removed. In the other case, if σ has exactly one remaining child that is not a compressed vertex, σ is marked as compressed and the traversal continues. If the only remaining child of σ is compressed, remove this child, connect σ to its grandchild, and mark σ as compressed. This takes $O(|\mathcal{Q}_v^c|)$ time and gives $\mathcal{Q}_w^c$.

Once all the compressed quadtrees $\mathcal{Q}_v^c$ are available, the linearized quadtrees $\mathcal{L}_v$ can be found by traversing B and each $\mathcal{Q}_v^c$ encountered. The total time to find the lists $\mathcal{L}_v$ is $O(n \log n + \sum_{v \in B} |\mathcal{Q}_v^c|) = O(n \log n)$, since $|\mathcal{Q}_v^c| = O(|\mathcal{I}_v|)$, for all $v \in B$, and $\sum_{v \in B} |\mathcal{I}_v| = O(n \log n)$ by Lemma 2.4. □

For a site $s \in S$ let σ be the cell with $s \in \sigma$ and $\frac{1}{2}|\sigma| \leq r_s < |\sigma|$ and define the *neighborhood* of s as $N(s) = N_{5\times 5}(\sigma)$. Using Lemma 2.1 it follows that D_s is completely covered by $N(s)$. By distributing the cells of $N(s)$ to the vertices of B, we can then answer the range query problem (R1) efficiently:

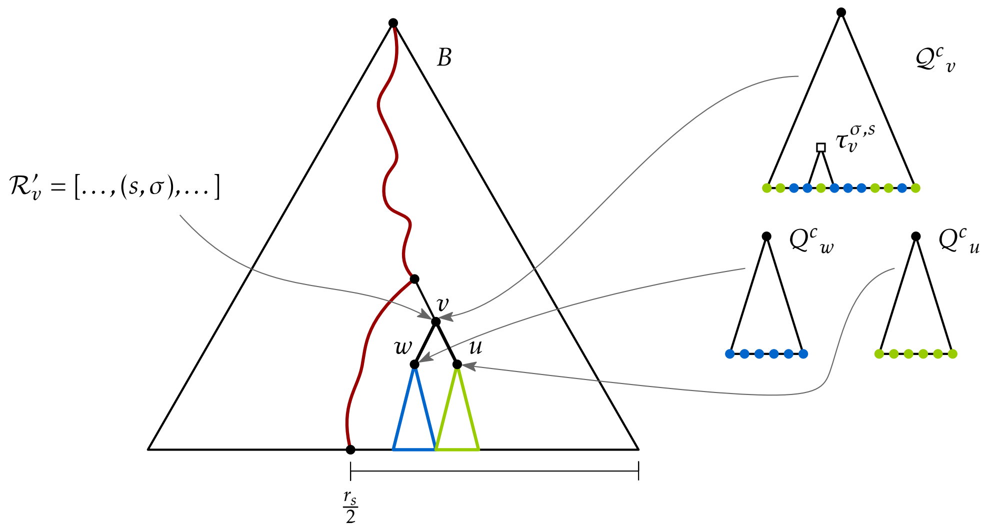


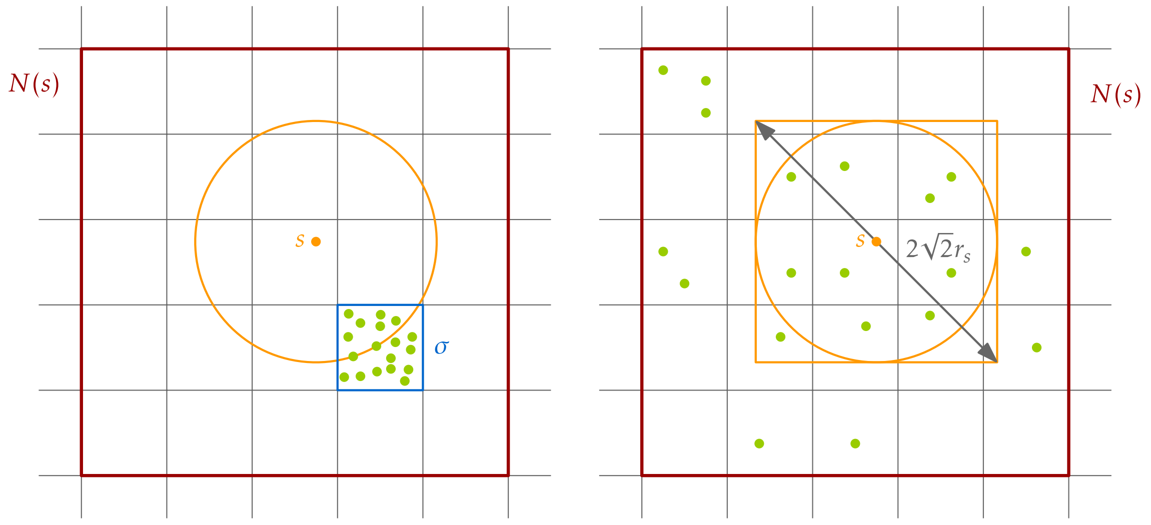
Figure 4.14: The split query (s,σ) is in the list $\mathcal{R}'_v$ of the canonical nodes of the path $\Pi_{\leftarrow}(\frac{r_s}{2})$. The cell $\tau_v^{\sigma,s}$ contains all sites in the canonical subset of v that are also contained in σ.

Lemma 4.21. *The range searching problem (R1) can be solved in $O(n \log n)$ time.*

Proof. We apply Lemma 4.20 to find the linearized quadtree for every canonical subset in B. Remember that the queries in (R1) are all sites $s \in S$ with $r_s \le \frac{r_{\max}}{2\sqrt{2}}$. As in the proof for Lemma 4.16, we split each such query into subqueries, by considering its neighborhood. Define the set of *split queries* as $\mathcal{R} = \bigcup_{s \in S; r_s \le \frac{r_{\max}}{2\sqrt{2}}} \left\{(s,\sigma) \mid \sigma \in N(s)\right\}$. The purpose of the split queries is to approximate the associated disks for the query sites by cells from the hierarchical grid. As each neighborhood contains a constant number of cells, we have $|\mathcal{R}| = O(n)$ and as $r_s \le \frac{r_{\max}}{2\sqrt{2}}$ also $|\sigma| \le \frac{r_{\max}}{\sqrt{2}} \le r_{\max}$ for all $\sigma \in \mathcal{R}$.

We now perform range queries for the cells in the split queries. We do this in a batched fashion. For each $v \in B$ we consider the sorted list $\mathcal{R}'_v$ of split queries that all have v as a canonical node and consider all split queries in $\mathcal{R}'_v$ together. Once these lists are available, we can answer the queries by merging the linearized quadtrees $\mathcal{L}_v$ with the lists $\mathcal{R}'_v$ similar to Lemmas 4.16 and 4.17, details follow.

First we describe how to compute the lists $\mathcal{R}'_v$. For this we first sort all elements of $\mathcal{R}$ in the Z-order of their second components in $O(n \log n)$ time. Next, we distribute the split queries along their associated search paths $\Pi_{\leftarrow}(\frac{r_s}{2}) = \Pi^{\circ}_{\leftarrow}(\frac{r_s}{2})$. For each $v \in B$, let $\mathcal{R}_v$ be the sublist of queries in $\mathcal{R}$, sorted by the Z-order of the second component, that have v on their path. By Lemma 2.4 and as $N(s)$ has constant size, we have $\sum_{v \in B} |\mathcal{R}_v| = O(n \log n)$. To find the lists $\mathcal{R}_v$ for all $v \in B$ in $O(n \log n)$ time, we perform a preorder traversal of B, computing the lists for the children from the lists of the parents. More precisely, given the sorted list $\mathcal{R}_v$ for a node $v \in B$, we can find the sorted list $\mathcal{R}_w$ for a child w of v in time $O(|\mathcal{R}_v|)$ by scanning $\mathcal{R}_v$ from left to right and by copying the elements that also appear in $\mathcal{R}_w$. Finally, we distribute the split queries into their canonical nodes. The canonical nodes of a split query are the right children of the nodes on $\Pi_{\leftarrow}(\frac{r_s}{2})$. Thus, for



(a) A cell of $N(s)$ contains more than α sites. (b) All cells on $N(s)$ contain few sites, we consider the bounding box of D_s.

Figure 4.15: The two cases of finding a square with many sites.

each $v \in B$ we can find the sorted list $\mathcal{R}'_v$ of split queries with v as a canonical node as follows: we iterate over all non-root nodes $v \in B$, and we scan the list $\mathcal{R}_w$ of the parent node w of v. We copy all queries that have v as a canonical node from $\mathcal{R}_w$ into $\mathcal{R}'_v$. This takes $O(n \log n)$ time.

Now we describe how to use this information to answer the query, see Figure 4.14. We iterate over all $v \in B$, and we merge the lists $\mathcal{R}'_v$ with the lists $\mathcal{L}_v$, in Z-order. This takes $O\left(\sum_{v\in B}|\mathcal{L}_v| + |\mathcal{R}'_v|\right) = O(n \log n)$ time and yields for each $(s,\sigma) \in \mathcal{R}'_v$ its Z-predecessor $\tau_v^{\sigma,s}$ in $\mathcal{Q}_v^c$. By Lemma 2.3, we know that if $\sigma \cap \tau_v^{\sigma,s} \neq \emptyset$ then we have $\sigma \cap \mathcal{I}_v = \tau_v^{\sigma,s} \cap \mathcal{I}_v$. Since these sites are all from $\mathcal{I}_v$, they all have radius at least $\frac{r_s}{2}$. We can find all these sites in $O(k)$ time, where k is the output size.

If $k > \alpha$, we stop and report σ as a square with many sites of large radius, see Figure 4.15(a). The diameter of σ is at most r_s and the radii of the reported sites are at least $\frac{r_s}{2} \geq \frac{r_s}{4\sqrt{2}}$. A close look at these bounds shows that the more than α sites that are reported might have a larger radius than required for the cell. This does not interfere with the correctness of our data structure. A cell reported this way is definitely a correct answer for the query. Potential other cells with many large sites will be found in the next step.

Here, we use the sites in each cell σ to accumulate the sites contained in a query disk D_s. This can be done by considering all canonical nodes of a split query of s and iterating over all sites contained in the cell σ. In each such cell there are at most α sites. For each site $t \in \sigma$, we can check in constant time if $t \in D_s$. If we find a query disk D_s with more than α sites of large radius, we stop and report its enclosing square with many sites of large radius. As $r = 2\sqrt{2}r_s$ is the diameter of the enclosing square, the radii are at least

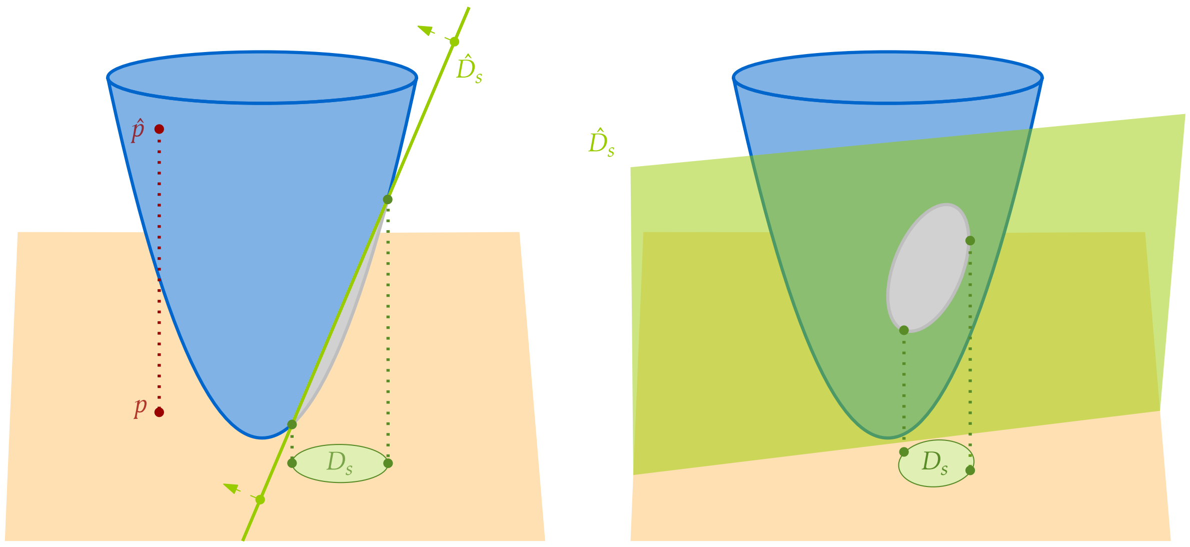


Figure 4.16: Lifting disks and points. In the right picture, $\hat{D}_s$ is only shown in a 2-dimensional representation.

$\frac{r}{4\sqrt{2}}$ as desired. Otherwise, for each $s \in S$, we have found the at most α sites of radius at least $\frac{r_s}{2}$ in D_s, see Figure 4.15(b). The whole algorithm takes $O(n \log n)$ time. □

4.5.2 Queries of Type (R2)

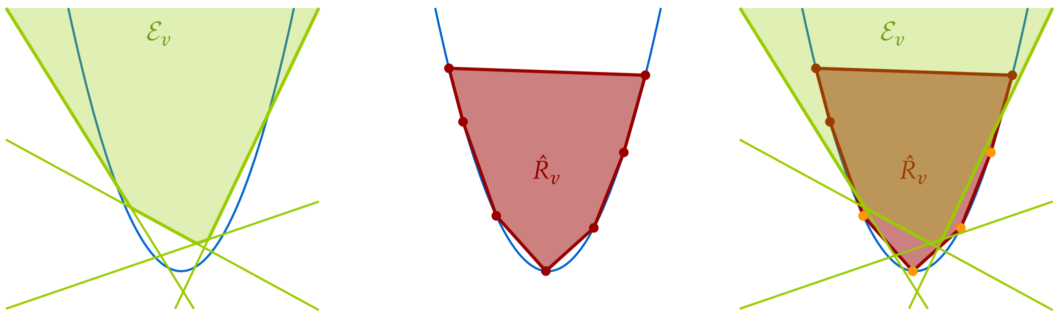
We again restate the kind of queries we consider in this section.

(R2) Given $O(n)$ query triples of the form (p, r_1, r_2) with $p \in \mathbb{R}^2$ and $0 < r_1 < r_2$, find a site $u \in S$ such that there is a query triple (p, r_1, r_2) with $r_1 < r_u < r_2$, and $p \in D_u$; or report that no such site u exists.

We will again make use of the binary search tree B as defined in Section 2.4. The interesting paths are $\Pi^{\bullet}_{\leftarrow}(r_1)$ and $\Pi^{\bullet}_{\rightarrow}(r_2)$ for each query (p, r_1, r_2). As for the queries of (R1) we again answer queries in a batched fashion by storing suitable data structures in each inner vertex v of B. The tree structure of B will again be used to quickly construct the search structures for each canonical subset and then solve all queries for a canonical subset in one batch.

We exploit the connection between planar disks and three-dimensional polytopes to represent the query points and sites of S as objects in $\mathbb{R}^2$. Let $U = \{(x, y, z) \mid z = x^2 + y^2\}$ be the three-dimensional unit paraboloid. For a point $p \in \mathbb{R}^2$, the lifted point $\hat{p}$ is the vertical projection of p onto U. Each disk D_s with $s \in S$ is transformed into an upper halfspace $\widehat{D_s}$, so that the projection of $\widehat{D_s} \cap U$ onto the xy-plane is the set $\mathbb{R}^2 \setminus D_s$;[1] see Figure 4.16. The union of a set of disks in $\mathbb{R}^2$ corresponds to the intersection of the lifted upper halfspaces in $\mathbb{R}^3$.

[1]This halfspace is bounded by the plane $z = 2s_x x + 2s_y y - \left(s_x^2 + s_y^2 - r_s^2\right)$, where $s = (s_x, s_y)$.



(a) Two-dimensional representation of $\mathcal{E}_v$.

(b) Two-dimensional representation of $\hat{\mathcal{R}}_v$.

(c) The orange points answer the query.

Figure 4.17: Using halfspace intersection and convex hull, we can answer queries of type (R2).

Lemma 4.22. *The range searching problem (R2) can be solved in $O(n \log n)$ expected time.*

Proof. For each $v \in B$, we construct a three-dimensional representation of the union of the disks in the canonical subset $\mathcal{I}_v$. As explained above, this is the intersection $\mathcal{E}_v$ of the lifted three-dimensional halfspaces $\widehat{D_s}$, for $s \in \mathcal{I}_v$. The intersection of two three-dimensional convex polyhedra with a total of m vertices can be computed in $O(m)$ time [Cha16; Cha92]. Recall that by Lemma 2.4 the total number of vertices of these polyhedra is $O(n \log n)$. Therefore, we can construct all the polyhedra $\mathcal{E}_v$, for $v \in B$, in overall $O(n \log n)$ time, by a bottom-up traversal of B. See Figure 4.17(a) for a two-dimensional representation of the polyhedron $\mathcal{E}_v$.

To preprocess the query processing, we construct a polytope $\widehat{R_v}$ for each $v \in B$. To define the polytope $\widehat{R_v}$ we consider the points p that appear in a query (p, r_1, r_2) that have v as a canonical node. $\widehat{R_v}$ is then defined as the convex hull of the lifted versions $\hat{p}$ of these points, see Figure 4.17(b). The lifted query points all lie on the unit paraboloid U, so every lifted query point appears as a vertex on $\widehat{R_v}$.

To find all polytopes $\widehat{R_v}$ for $v \in B$ efficiently, we proceed as follows: let A be the three-dimensional point set obtained by taking all points that appear in a query and lifting them onto the unit paraboloid. We compute the convex hull of A in $O(n \log n)$ time and store it in the root of B. Then, for each $v \in B$, we find the convex hull $\mathcal{CH}_v$ of all lifted queries that have v in $\Pi_{\leftarrow}(r_1)$ or $\Pi_{\rightarrow}(r_2)$. This can be done in $O(n \log n)$ total expected time by a top-down traversal of B. We already have the polytope for the root of B. To compute the polytope for a child node, given that the polytope for the parent node is available, we use the fact that for any polytope $\mathcal{R}$ in $\mathbb{R}^3$ with m vertices, we can compute the convex hull of any subset of the vertices of $\mathcal{R}$ in $O(m)$ expected time [CM11]. For $v \in B$, let $\mathcal{CH}_v$ be the convex hull of the lifted query points that have v on $\Pi_{\leftarrow}(r_1)$ or $\Pi_{\rightarrow}(r_2)$. Once we have $\mathcal{CH}_v$ available, we can compute for each $v \in B$ the polytope $\widehat{R_v}$ that is the convex hull of the lifted query points that have v as a canonical node. For this, we consider the convex hull $\mathcal{CH}_w$ stored at the parent node w of v, and again extract the convex hull $\widehat{R_v}$ for the lifted query points that have v as a canonical node.

Now that the polyhedra $\widehat{R_v}$ and the polytopes $\mathcal{E}_v$ are available for all $v \in B$, we can answer the query as follows: for each node $v \in B$, we check for vertices of $\widehat{R_v}$ that do not lie inside of $\mathcal{E}_v$. These are exactly the vertices of $\widehat{R_v}$ that are not vertices of $\widehat{R_v} \cap \mathcal{E}_v$. As mentioned above, the intersections $\widehat{R_v} \cap \mathcal{E}_v$ can be found in linear time for each node $v \in B$, for a total time $O(n \log n)$. Upon a close examination it can be seen that the algorithm by Chan [Cha16] can be extended to also report the points of $\widehat{R_v}$ that are not part of the intersection. The main idea of Chan's algorithm is to recursively find the intersection of polyhedra defined by a suitably chosen subset of the planes defining the original polyhedra, and to then merge the remaining plane with this intersection. For the merging step, each face of the recursively found polyhedron is triangulated and all hyperplanes of either input polyhedron that can intersect the triangle are collected in a *candidate list*. Now if a hyperplane defining a face of $\widehat{R_v}$ is not part of any such candidate list, the vertices that define the face on the hyperplane cannot be vertices of the intersection and can thus be reported. In the other case, the vertices of the face defined by the hyperplane have to be checked explicitly. Chan [Cha16] argues that each candidate list has constant size and finds the intersection explicitly using brute force. During this brute force step, we can also find the vertices of $\widehat{R_v}$ that are not part of the intersection.

Figure 4.17(c) illustrates how these points answer the query. If for any such intersection $\widehat{R_v} \cap \mathcal{E}_v$, there is a lifted site $\hat{s} \in \widehat{R_v}$ that is not a vertex of $\widehat{R_v} \cap \mathcal{E}_v$, we can explicitly find a disk containing it in linear time and report the range query as successful. If the query is unsuccessful for all $v \in B$ we report the complete range query as unsuccessful. □

4.5.3 Queries of Type (R2')

The queries of type (R2'), restated here convenience, are actually very similar to the range queries (R2).

(R2') Given a set $R \subseteq S \times \mathbb{R}^+$ of $O(n)$ query tuples, report for each $u \in S$ one of the query tuples $(t, r) \in R$ such that $t \neq u$, $t \in D_u$ and $r_u < r$, if it exists.

In (R2), for a set of queries of the form (s, r_1, r_2), we wanted to report one query triple such that there is a site u with $r_1 < r_u < r_2$ or to report that there is no such u for any query triple. In (R2'), we want to find one query tuple for each site. However the general task: *find a site within the given radius range, containing a certain point* remains the same. It actually turns out, that we can use geometric duality to adapt the approach for (R2) to the situation for (R2').

Lemma 4.23. *The range searching problem (R2') can be solved in $O(n \log n)$ expected time.*

Proof. Recall that for solving (R2), we represented each disk D_s for $s \in S$ by the hyperplane $\widehat{D_s} : z = 2s_x \cdot x + 2s_y \cdot y - (s_x^2 + s_y^2 - r_s^2)$ and lifted each query site t to $\hat{t} = (t_x, t_y, t_x^2 + t_y^2)$. For the problem at hand, we will use a standard duality transformation [dBer+08] which maps a point $p = (p_x, p_y, p_z)$ to the hyperplane $p^* : z = p_x \cdot x + p_y \cdot y - p_z$ and vice versa. Thus, by first using the lifting described in Section 4.5.2, we represent each disk D_s by the point $\widehat{D_s}^* = (2s_x, 2s_y, (s_x^2 + s_y^2 - r_s^2))$ and each query site t by the hyperplane

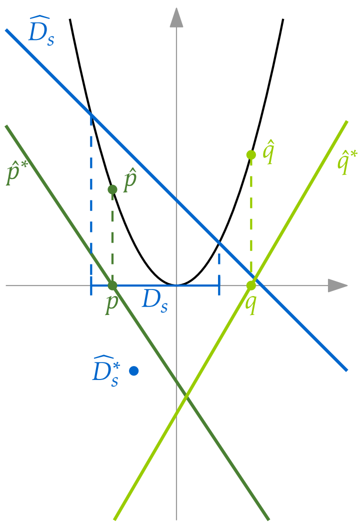


Figure 4.18: The point p lies in D_s so the point $\widehat{D_s}^*$ lies below $\hat{p}^*$.

$\hat{t}^* : z = t_x \cdot x + t_y \cdot y - (t_x^2 + t_y^2)$. For answering a query in (R2), the question if a query point t is contained in a disk D_s was reduced to the question, whether $\hat{t}$ lies below $\widehat{D_s}$. For (R2') we want to test if a site $u \in S$ is contained in the disk defined by a query site t. This reduces to asking: *does* $\widehat{D_u}^*$ *lie below* $\hat{t}^*$*?* A two-dimensional representation of this situation can be seen in Figure 4.18.

To answer the batched range query of type (R2'), we use the same approach as described in Lemma 4.22, but we use the dual structures. To be precise, again a tree B on all sites sorted by radii is build. There is only an upper bound on the radius, so we only have to consider the paths $\Pi^{\bullet}_{\rightarrow}(r)$ for the queries. In each vertex v of B we store a convex polyhedron $\mathcal{E}_v^*$ representing all sites in the canonical subset of v. For (R2) this polyhedron was the intersection of the halfplanes defined by the sites, in the dual this translates to the convex hull of the points $\widehat{D_s}^*$ for each site s in the canonical subset of v. Recall that given the convex hull of m vertices, we can compute the convex hull of a subset of vertices in $O(m)$ expected time [CM11]. Thus we can construct the polyhedra $\mathcal{E}_v^*$ for each vertex $v \in B$ in overall $O(n \log n)$ time.

Furthermore, in each vertex v we want to store a representation of the query sites that have v as a canonical node. For this, we first distribute all queries to the leaves at the end of $\Pi_{\rightarrow}(r)$ and compute the intersection of the collected halfspaces $\hat{t}^*$ in the leaves. Then we compute the intersection of all queries below an inner vertex v in a bottom up fashion. As we can compute the intersection of two polyhedra with a total of m vertices in $O(m)$ time [Cha16; Cha92], this again needs $O(n \log n)$ overall time. Then we can copy these polyhedra which are stored in a right child of an inner node to its left sibling. These copied polyhedra $\widehat{\mathcal{R}}_v^*$ are the desired representation of the sites which have v as a canonical node.

After having all these polyhedra available, the remaining algorithm is mostly identical to that given in Lemma 4.22. For each vertex $v \in B$, we can now compute the intersection of the polyhedra $\mathcal{E}_v^*$ and $\widehat{\mathcal{R}}_v^*$ and find the points on $\mathcal{E}_v^*$ that are not in $\mathcal{E}_v^* \cap \widehat{\mathcal{R}}_v^*$. This can be done as in Lemma 4.22 in linear time for each v and yields a site set U, were all $u \in U$ contain a query site. In contrast to (R2) we do not only need an answer for one site found this way, but rather for all. If there is a subset $U' \subseteq U$ for which we have not already found a tuple, we construct a vertical ray shooting query data structure on $\widehat{\mathcal{R}}_v^*$. Now we can explicitly perform a vertical ray shooting query for each $u \in U'$ which then yields the query tuple for u. There is a data structure due to Dobkin and Kirkpatrick [DK82] that allows ray shooting queries in time $O(\log n)$ after $O(n)$ preprocessing time. Each site is used for one vertical ray shooting query and the ray shooting data structures can be build in $O(n \log n)$ overall time. Thus the running time for this last step is again $O(n \log n)$, resulting in the overall running time of $O(n \log n)$ expected time as claimed. □

CHAPTER 5

Dynamic Connectivity in Disk Graphs

This chapter deals with the dynamic connectivity problem in disk graphs. To be precise, given a disk graph on a site set S we describe two data structures that allow the deletion of sites and queries of the form: *given $s \in S$ and $t \in S$, are s and t connected in $\mathcal{D}(S)$?* The first data structure works for disk graphs with bounded radius ratio Ψ, while the second works for general disk graphs. For small values of Ψ the first data structure is more efficient. However, as the radius ratio can be potentially unbounded, the second data structure has a better running time for sufficiently large values of Ψ.

The overall structure of both data structures is very similar. For both we define a set of *regions* and a *proxy graph* on the set of regions and sites. The proxy graph for general disk graphs can be seen as an extension of the graph for bounded radius ratio. The definition of these sparse proxy graphs, which accurately represent the connectivity, is the main technical challenge in this chapter. We store the proxy graphs in a Holm et al. [HdLT01] data structure for general graphs, that can be queried in $O\left(\frac{\log n}{\log\log n}\right)$ time and allows edge insertions and deletions in $O(\log^2 n)$ amortized time. As the proxy graph is sparse, storing it in a Holm et al. data structure allows us to efficiently query and update the connectivity information. In addition to the connectivity data structure for general graphs, we need other auxiliary geometric data structures.

For the preprocessing step, we need access to an efficient static additively weighted nearest neighbor data structure (AWNN) given in Lemma 5.1. Furthermore, in order to efficiently identify the edges that have to be deleted from the proxy graph at the deletion of a site, we use a data structure given by Kaplan et al. [Kap+21a][1]. For completeness we give the precise results below.

Lemma 5.1 (AWNN, de Berg et al. [dBer+08], Fortune [For87], and Sharir [Sha85]). *Let $P \subseteq \mathbb{R}^2$ be a set of points, each with an associated weight w_p. Then there is a data structure,*

[1]The paper on dynamic disk graph connectivity [Kap+21a] is the combination of a variety of results related to the topic. It mainly consists of the results given in three papers presented at EuroCG [KKM21; Kap+21b; Kap+16]. My contribution to [Kap+21a] were the decremental data structures that were partially presented at EuroCG’21 [Kap+21b]

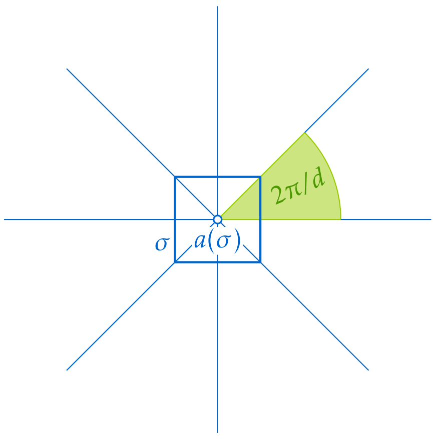


Figure 5.1: The plane is decomposed into d congruent cones with opening angle $\frac{2\pi}{d}$ and apex $a(\sigma)$.

that after $O(n \log n)$ preprocessing time, can report for a query point $q \in \mathbb{R}^2$ the point $p \in P$ that minimizes $\|pq\| + w_p$ in $O(\log n)$ time.

Lemma 5.2 (Reveal data structure (RDS), Kaplan et al. [Kap+21a]). *Let R and B be site sets with $|R| + |B| = n$. There is a data structure that after some preprocessing allows the deletion of sites from R and B. After deleting a site from B, it reports all disks from R now disconnected from $\bigcup_{b \in B} D_b$. Preprocessing the data structure takes $O(|B| \log^5 n \lambda_6(\log n) + |R| \log^3 n)$ time. Deleting m sites from B and an arbitrary number of sites from R takes $O((m \log^9 n)\lambda_6(\log n) + |R| \log^4 n)$ expected time and $O(n \log^3 n)$ expected space, where $\lambda_s(n)$ is the maximum length of a Davenport-Schinzel sequence of order s on n symbols.*

5.1 Logarithmic Dependency on Ψ

This section describes a data structure for the dynamic connectivity problem, whose update time has a logarithmic dependency on Ψ. We first define the proxy graph and show its properties in detail before moving on to describing the actual data structure, using the proxy graph in its core.

The Proxy Graph We start by defining the proxy graph H. The vertex set of H contains one vertex for each site in S, plus additional vertices that represent certain regions in the plane, to be defined below. Each region is defined based on a cell of a quadtree. With each such region A, we associate two site sets. The first set $S_1(A) \subseteq S$ is defined such that all sites $s \in S_1(A)$ lie in A and have a radius r_s comparable to the size of A, for a notion of comparable to be defined below. A site can be assigned to several regions. We will ensure that for each region A, the induced disk graph $\mathcal{D}(S_1(A))$ of the associated sites is a clique. The second set $S_2(A) \subseteq S$, associated to the region A contains a site if it lies in the cell defining the region, its radius is not too large and it intersects

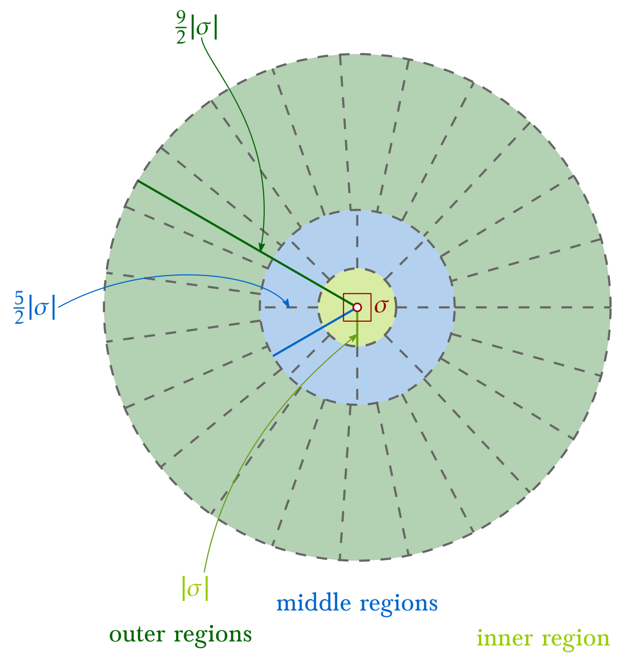


Figure 5.2: The regions defined by a cell σ.

at least one site in $S_1(A)$. Again, the vague notions in this description will be specified below.

The proxy graph H is bipartite, with all edges going between the site-vertices and the region-vertices. The edges of H connect each region A to the sites in $S_1(A)$ or $S_2(A)$. The connections between the sites in $S_1(A)$ with A constitute a sparse representation of the corresponding clique in $\mathcal{D}(S_1(A))$. The edges connecting a site in $S_2(A)$ to A allow us to represent all edges in $\mathcal{D}(S)$ between $S_2(A)$ and $S_1(A)$ by two edges in H. Since $\mathcal{D}(S_1(A))$ is a clique, this sparse representation does not change the connectivity between the sites. Furthermore, we will ensure that the number of regions, and the total size of the associated sets $S_1(A)$ and $S_2(A)$ is small, giving a sparse proxy graph.

We now describe the details. The graph H has vertex set $S \cup \mathcal{A}$, where S are the sites and $\mathcal{A}$ is a set of regions. To define the regions, we first augment the (non-compressed) quadtree $\mathcal{Q}$ as follows. For each site $s \in S$, we consider the cell σ_s in the hierarchical grid with $s \in \sigma_s$ and $|\sigma_s| \leq r_s < 2|\sigma_s|$, and we set $N(s) = N_{15\times 15}(\sigma_s)$. Note that as the smallest radius in S is 1, these cells have diameter at least 1 and are thus contained in the hierarchical grid. We add all cells in $\bigcup_{s\in S} N(s)$ to $\mathcal{Q}$ and, with a slight abuse of notation, still call the resulting tree $\mathcal{Q}$.

The set $\mathcal{A}$ defining the vertex set of H is a subset of the set $\mathcal{A}_\mathcal{Q}$ that contains certain regions for each cell of $\mathcal{Q}$. There are three kinds of regions for a cell σ of $\mathcal{Q}$: the *outer regions*, the *middle regions*, and the *inner region*. To define the outer regions for σ, we consider the set $\mathcal{C}_{d_1}(\sigma)$ of d_1 congruent cones with their common apex at the center $a(\sigma)$ of σ, for some integer parameter d_1 to be determined below, see Figure 5.1. For each cone $C \in \mathcal{C}_{d_1}(\sigma)$, we intersect C with the annulus that is centered at $a(\sigma)$ and that has inner radius $\frac{5}{2}|\sigma|$ and outer radius $\frac{9}{2}|\sigma|$, and we add the resulting region to $\mathcal{A}_\mathcal{Q}$. All these

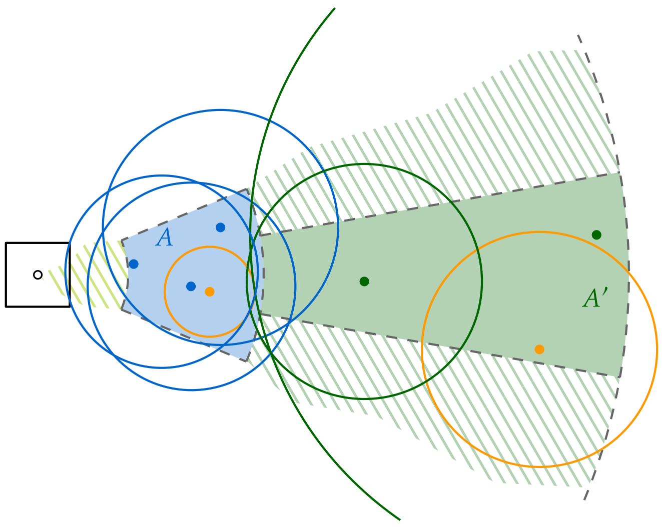


Figure 5.3: The set $S_1(A)$ is marked blue. The orange site in A is not in the set because its radius is too small. The orange site in A' is not in $S_1(A')$: while its radius is in the correct range, it does not touch or intersect the inner boundary.

regions form the *outer regions* of σ. The *middle regions* of σ are defined similarly, but using the set $\mathcal{C}_{d_2}(\sigma)$ of d_2 congruent cones centered at $a(\sigma)$, for another integer parameter d_2 to be determined below, and the annulus that is centered at $a(\sigma)$ and that has inner radius $|\sigma|$ and outer radius $\frac{5}{2}|\sigma|$. Finally, the inner region for σ is the disk with center $a(\sigma)$ and radius $|\sigma|$. See Figure 5.2 for an illustration of the regions for a cell σ.

We associate a set of sites $S_1(A) \subseteq S$ with each region $A \in \mathcal{A}_Q$, depending on the type of the region. This is done as follows: first, suppose that A is an outer region for a cell σ. Then, the set $S_1(A)$ contains all sites t such that

(a) $t \in A$

(b) $|\sigma| \leq r_t \leq 2|\sigma|$; and

(c) $\|a(\sigma)t\| \leq r_t + \frac{5}{2}|\sigma|$.

This means that t represents a disk with a size that is comparable to the diameter of σ, whose center lies in the region A, and that intersects the inner boundary of A. Second, if A is a medium or the central region for a cell σ, then $S_1(A)$ contains all sites t such that

(a) $t \in A$; and

(b) $|\sigma| \leq r_t < 2|\sigma|$.

That is, the site t represents a disk, again with a size is comparable to σ and center in A. See Figure 5.3 for an illustration of these definitions. The only difference between the conditions for the outer and the other region types is the upper bound on the distance $\|a(\sigma)t\|$, which is trivially true for a middle or inner region. We define $\mathcal{A} \subseteq \mathcal{A}_Q$ to be the

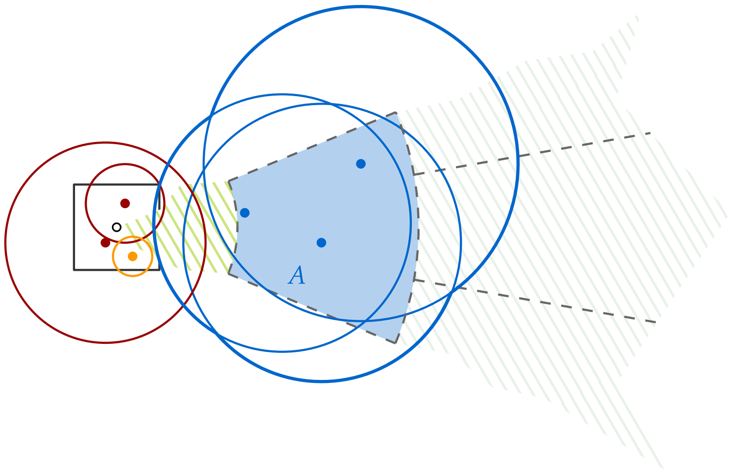


Figure 5.4: The red sites in σ are in $S_2(A)$. The radius of the orange site is in the correct range, but it does not the intersect a site in $S_1(A)$ (marked blue).

set of regions where $S_1(A) \neq \emptyset$. In the following, we will not strictly distinguish between a vertex from $\mathcal{A}$ and the corresponding region, provided that it is clear from the context.

Additionally for each region $A \in \mathcal{A}$ we define a set $S_2(A)$ as follows and illustrated in Figure 5.4. Let A be a region defined by a cell σ, then the set $S_2(A)$ contains all sites s such that:

(a) $s \in |\sigma|$

(b) s is adjacent in $\mathcal{D}(S)$ to at least one site in $S_1(A)$; and

(c) $r_s < 2|\sigma|$

We add an edge $\{s, A\}$ between a site and a region, if $s \in S_1(A) \cup S_2(A)$. Note that the sets $S_1(A)$ and $S_2(A)$ are not necessarily disjoint, as for the center region defined by a cell σ, a site in σ with $|\sigma| \leq r_s < 2|\sigma|$ will be both in $S_1(A)$ and $S_2(A)$. This will however influence neither the preprocessing time nor the correctness in a negative way.

The following structural lemma will help us to show that H accurately represents the connectivity, as well as to bound the size and preprocessing time.

Lemma 5.3. *Let $\{s, t\}$ be an edge in $\mathcal{D}(S)$ with $r_s \leq r_t$, then*

(a) there is a cell $\sigma \in N(t)$ such that $s \in \sigma$ and σ defines a region A with $t \in A$; and

(b) all cells that define a region A with $t \in S_1(A)$ are in $N(t)$.

Proof. First, we show (a). By assumption, $\{s, t\}$ is an edge in $\mathcal{D}(S)$, and thus $\|st\| \leq r_s + r_t$. Let τ be the cell containing t with $|\tau| \leq r_t < 2|\tau|$. By the assumption on the radii, we have $r_s \leq r_t < 2|\tau|$. Now let σ be the cell of the hierarchical grid with $|\sigma| = |\tau|$ and $s \in \sigma$. Using the triangle inequality this yields $\|a(\sigma)t\| \leq \frac{1}{2}|\sigma| + r_s + r_t < \frac{9}{2}|\sigma|$. Thus, the center of σ has distance at most $\frac{9}{2}|\sigma|$ to t. Consider the disk $D = D(t, \frac{9}{2}|\tau|)$. All cells that can contain s

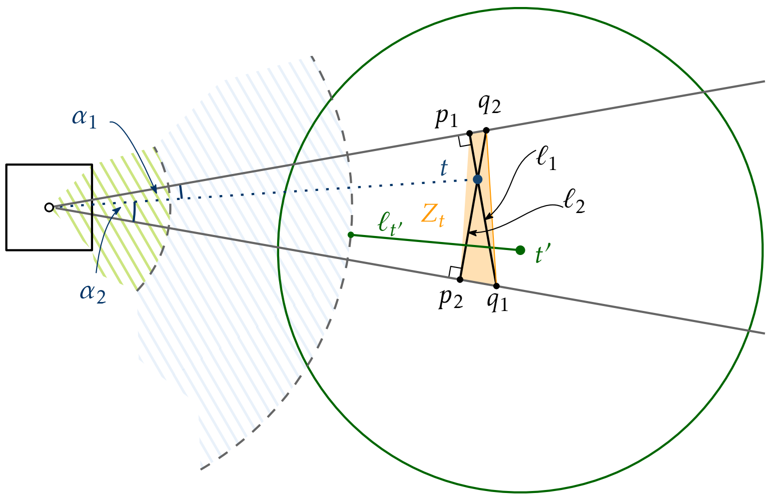


Figure 5.5: The line segment $\ell_{t'}$ intersects $\overline{q_1 q_2}$.

have their center in D. By Lemma 2.1, $N(t)$ contains D and thus $\sigma \in N(t)$. By symmetry we also have $t \in D(a(\sigma), \frac{9}{2}|\sigma|)$. The regions defined by σ partition this disk, and thus t lies in one of the regions.

For (b), note that by the condition on the radius, the only cells defining regions such that $t \in S_1(A)$ lie on the same level of the hierarchical grid as the cells in $N(t)$. Furthermore, by the distance condition, the centers of the cells lie in the disk $D(t, \frac{9}{2}|\sigma|)$. The proof for (a) already showed that this disk is completely contained in $N(t)$. □

Before we argue that our proxy graph H accurately represents the connectivity of $\mathcal{D}(S)$, we first show that the associated sites of a region in $\mathcal{A}$ form a clique in $\mathcal{D}(S)$.

Lemma 5.4. *Suppose that $d_1 \geq 23$ and $d_2 \geq 8$. Then, for any region $A \in \mathcal{A}$, the associated sites in $S_1(A)$ form a clique in $\mathcal{D}(S)$.*

Proof. In the following, let σ be the cell defining A. We handle the three cases where A is an outer, middle or inner region separately.

Case 1: *A* is an outer region First, suppose that A is an outer region. Let t be a site in $S_1(A)$, and let C be the cone that is centered at $a(\sigma)$ and was used to define A. Consider the line segments ℓ_1 and ℓ_2 that go through t, lie in C, and are perpendicular to the upper and the lower boundary of C, respectively. We denote the endpoints of ℓ_1 by p_1 and q_1, and the endpoints of ℓ_2 by p_2 and q_2, where p_1 and p_2 are the endpoints at the right angles. See Figure 5.5 for an illustration.

We claim that both ℓ_1 and ℓ_2 are completely contained in D_t. Let α_i be the acute angle at $a(\sigma)$ defined by the line segments $\overline{a(\sigma)t}$ and $\overline{a(\sigma)p_i}$ for $i = 1, 2$. By our choice of C, we have $\alpha_1 + \alpha_2 = \frac{2\pi}{d_1}$.

Now, we can compute the following lengths:

$$\begin{aligned}
\|a(\sigma)p_1\| &= \cos(\alpha_1)\cdot\|a(\sigma)t\| \\
\|a(\sigma)p_2\| &= \cos(\alpha_2)\cdot\|a(\sigma)t\| \\
\|p_1q_1\| &= \|a(\sigma)p_1\|\cdot\tan\left(\frac{2\pi}{d_1}\right) \\
\|p_2q_2\| &= \|a(\sigma)q_1\|\cdot\tan\left(\frac{2\pi}{d_1}\right).
\end{aligned}$$

Combining these bounds, we get

$$\begin{aligned}
\|\ell_1\| = \|p_1q_1\| &= \tan\left(\frac{2\pi}{d_1}\right)\cdot\cos(\alpha_1)\cdot\|a(\sigma)t\| \\
\|\ell_2\| = \|p_2q_2\| &= \tan\left(\frac{2\pi}{d_1}\right)\cdot\cos(\alpha_2)\cdot\|a(\sigma)t\|
\end{aligned}$$

The length of ℓ_1 and ℓ_2 are maximized for $\alpha_2 = 0$ or $\alpha_1 = 0$, respectively. This yields

$$\begin{aligned}
\|\ell_1\|, \|\ell_2\| &\le \|a(\sigma)t\|\cdot\tan\left(\frac{2\pi}{d_1}\right) \\
&\le \left(r_t + \frac{5}{2}|\sigma|\right)\cdot\tan\left(\frac{2\pi}{d_1}\right) \\
&\le \frac{7}{2}r_t\cdot\tan\left(\frac{2\pi}{d_1}\right) \\
&\le r_t && \text{for } d_1 \ge 23.
\end{aligned}$$

Thus, ℓ_1 and ℓ_2, as well as their convex hull Z_t, are completely contained in D_t.

Now, we take a closer look at the distance condition for the sites $t \in S_1(A)$. Recall that we require $\|a(\sigma)t\| \le r_t + \frac{5}{2}|\sigma|$ or, equivalently, the disk D_t must intersect or touch the disk $D\big(a(\sigma), \frac{5}{2}|\sigma|\big)$. As $\|a(\sigma)t\| \ge \frac{5}{2}|\sigma|$, the line segment $\overline{q_1q_2}$ lies completely outside of $D\big(a(\sigma), \frac{5}{2}|\sigma|\big)$. Now consider the line segment $\ell_t = \overline{a(\sigma)t} \setminus D(a(\sigma), \frac{5}{2}|\sigma|)$. It is the part of the line segment between t and $a(\sigma)$ that lies outside of $D(a(\sigma), \frac{5}{2}|\sigma|)$. This line segment ends with a point on $D(a(\sigma), \frac{5}{2}|\sigma|)$. Now, let t' be another site in $S_1(A)$. If t' lies in the convex hull Z_t, or if t lies in the convex hull $Z_{t'}$, we are done, as this directly implies that t and t' are neighbors in $\mathcal{D}(S)$. Thus, assume without loss of generality that t' is separated from $a(\sigma)$ by the convex hull Z_t. Then, $\ell_{t'}$ intersects $\overline{q_1q_2}$, and as $\ell_{t'} \subseteq D_{t'}$ and $\overline{q_1q_2} \subseteq D_t$ this again induces an edge $\{t, t'\}$.

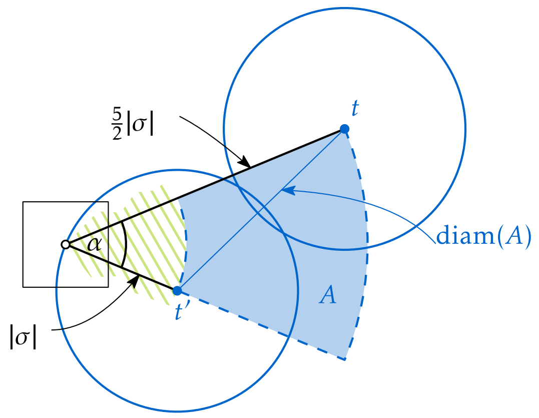


Figure 5.6: The diameter of a middle region is at most $2|\sigma|$.

Case 2: A is a middle region Next, suppose that A is a middle region, and let $t, t' \in S_1(A)$. By the definition of $S_1(A)$ we have $t, t' \in A$. The law of cosines yields for $\operatorname{diam}(A)$

$$\begin{aligned}
\operatorname{diam}(A)^2 &\leq |\sigma|^2 + \left(\frac{5}{2}|\sigma|\right)^2 - \frac{10}{2}|\sigma|^2 \cos\left(\frac{2\pi}{d_2}\right) && \text{for } d_2 \geq 8 \\
&\leq |\sigma|^2 + \left(\frac{5}{2}|\sigma|\right)^2 - \frac{10}{2}|\sigma|^2 \cos\left(\frac{2\pi}{8}\right) \\
\operatorname{diam}(A) &\leq 2|\sigma|
\end{aligned}$$

As we require that $r_t, r_{t'} \geq |\sigma|$, it follows that $\{t, t'\}$ is an edge in $\mathcal{D}(S)$, see Figure 5.6.

Case 3: A is an inner region Finally, suppose that A is an inner region and let $t, t' \in A$. Since t and t' both lie in the disk $D(a(\sigma), |\sigma|)$ of diameter $2|\sigma|$, and since $r_t, r_{t'} \geq |\sigma|$, we again get that the edge $\{t, t'\}$ is present in $\mathcal{D}(S)$. □

Having Lemmas 5.3 and 5.4 at hand, we can now show that H accurately represents the connectivity of $\mathcal{D}(S)$.

Lemma 5.5. *Two sites $s, t \in S$ are connected in H if and only if they are connected in $\mathcal{D}(S)$.*

Proof. First, we show that if s and t are connected in H, they are also connected in $\mathcal{D}(S)$. The path between s and t in H alternates between the site-vertices and the region-vertices. Thus, it suffices to show that if two sites u and u' are connected with the same region $A \in \mathcal{A}$, they are also connected in $\mathcal{D}(S)$. This follows directly from Lemma 5.4: if u and u' both lie in $S_1(A)$, they are part of the same clique and thus adjacent. In the other case we can assume that u lies in $S_2(A)$. Then $S_2(A)$ is non-empty, which also implies that $S_1(A)$ is non-empty and u is connected to u' via the clique induced by $S_1(A)$, as can be seen in Figure 5.7. The claim follows, regardless if u' lies in $S_1(A)$ or in $S_2(A)$.

Now we consider two sites that are connected in $\mathcal{D}(S)$, and we show that they are also connected in H. It suffices to show that if s and t are adjacent in $\mathcal{D}(S)$, they are connected

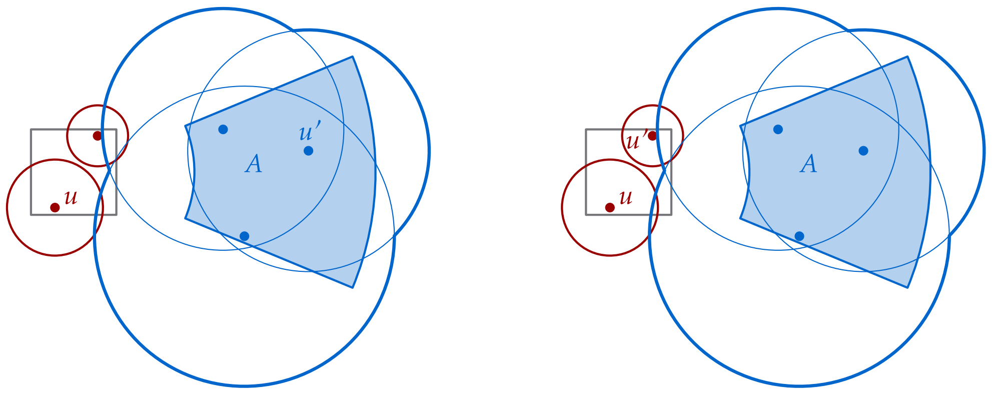


Figure 5.7: The site $u \in S_2(A)$ is connected to the clique by definition. Then u' is connected to u via the clique, no matter if $u' \in S_1(A)$ or $u' \in S_2(A)$.

in H. Assume without loss of generality, that $r_s \leq r_t$ and let σ be the cell in $N(t)$ such that $s \in \sigma$. This cell in $N(t)$ exists by Lemma 5.3 (a) and was explicitly added to $\mathcal{Q}$. We can thus conclude that $t \in S_1(A)$ for some $A \in \mathcal{A}_{\mathcal{Q}}$. As the regions with non-empty sets $S_1(A)$ are in $\mathcal{A}$ by definition, the edge $\{t, A\}$ exists in H.

Now, we argue that for the region A found above, $s \in S_2(A)$ and thus the edge $\{A, s\}$ also exists in H. This follows by straightforward showing that properties of a site in $S_2(A)$ hold for s. We have $s \in \sigma$ by the definition of σ and by assumption the radius of s is bounded by $r_s \leq r_t < 2|\sigma|$. Finally, to see that at least one site in $S_1(A)$ intersects D_s, note that t is in $S_1(A)$ and s and t intersect. □

After we have shown that H accurately represents the connectivity relation in $\mathcal{D}(S)$, we now show that the number of edges in H depends only on n and Ψ and not on the number of edges in $\mathcal{D}(S)$ or the diameter of S. As the size of H only depends on the number of sites in the sets $S_1(A)$ and $S_2(A)$, we first take a look at those sizes.

Lemma 5.6. *Let $\mathcal{A}$, $S_1(A)$ and $S_2(A)$ be defined as above. Then $\sum_{A \in \mathcal{A}} |S_1(A)| = O(n)$ and $\sum_{S \in \mathcal{A}} |S_2(A)| = O(n \log \Psi)$.*

Proof. For the first claim, Lemma 5.3 (b) tells us that t can only lie in regions A defined by cells in $N(t)$. There are $O(1)$ such cells and thus t lies in at most $O(1)$ sets $S_1(A)$. Now each site t contributes only $O(1)$ to the sum $\sum_{A \in \mathcal{A}} |S_1(A)|$ and we get the claimed bound of $O(n)$.

Next, we focus on the total size of all sets $S_2(A)$. A necessary condition for s to lie in $S_2(A)$ is that s lies in the cell defining A, so we focus on the cells containing s. There are potentially $O(\log(\text{diam}(S)))$ cells in $\mathcal{Q}$ that contain s. Recall however, that the set $\mathcal{A}$ only contains those cells, that have a non-empty set $S_1(A)$ and thus a diameter proportional to the radii of the sites in $S_1(A)$. As the maximum radius in S is at most Ψ, the largest cell which is of interest has diameter $2^{\lfloor \log \Psi \rfloor}$. Thus, regions in $\mathcal{A}$ are defined by cells of the quadtree that lie below the level $\lfloor \log(\Psi) \rfloor$. For a fixed site this implies that it can lie in at most $O(\log \Psi)$ sets $S_2(A)$ and thus with a similar argument as above, $\sum_{A \in \mathcal{A}} |S_2(A)| = O(n \log \Psi)$. □

Corollary 5.7. *H has $O(n)$ vertices and $O(n\log\Psi)$ edges.*

Proof. We can have at most $\sum_{A\in\mathcal{A}}|S_1(A)|$ non-empty regions, and thus $|\mathcal{A}| = O(n)$. This constitutes the bound on the number of vertices. Furthermore the number of edges is $\sum_{A\in\mathcal{A}}(|S_1(A)|+|S_2(A)|) = O(n\log\Psi)$. □

The Data Structure Now, we can describe how to use the proxy graph H to build a data structure that allows interleaved deletions and connectivity queries in a disk graph. The data structure will consist of several components: we store a forest of quadtrees that contains all regions $A\in\mathcal{A}$. For each region $A\in\mathcal{A}$ we store the sets $S_1(A)$ and $S_2(A)$ implicitly in a reveal data structure (RDS) as defined in Lemma 5.2 with $S_1(A)$ as the set B and $S_2(A)$ as the set R. Finally, we store the graph H in a Holm et al. [HdLT01] data structure $\mathcal{H}$.

Our data structure now works as follows. The queries are answered using $\mathcal{H}$. Now we focus on the deletion of a site s. First, we delete all edges incident to s from $\mathcal{H}$. Then, we consider all regions A such that $s\in S_1(A)$. We remove s from $S_1(A)$ and the RDS. Let U be the set of revealed sites from $S_2(A)$ reported by the RDS. We delete each site $u\in U$ from $S_2(A)$ and the corresponding RDS. Additionally we delete the edges $\{u,A\}$ for $u\in U$ from $\mathcal{H}$ for all $u\in U\setminus S_1(A)$, these are all sites in U that are not also in $S_1(A)$. Finally, consider a region A such that $s\in S_2(A)$. We simply remove s from the set $S_2(A)$ and the associated RDS. Note that we do not update the forest of quadtrees, as this would not change the asymptotic runtime.

We start the analysis of the data structure, by considering the preprocessing time:

Lemma 5.8. *Given a site set S we can preprocess the data structure described above in $O(n\log^5(n)\lambda_6(\log n)+n\log\Psi\log^3 n)$ expected time.*

Proof. In a first step, we build a forest, containing the lowest $\lfloor\log\Psi\rfloor+1$ levels of the quadtree $\mathcal{Q}$. For this we identify for each site in S the cell of level $\lfloor\log\Psi\rfloor$ containing it. As we can identify the cell in $O(1)$ time, this takes $O(n)$ overall time. The lower $\lfloor\log\Psi\rfloor+1$ levels of $\mathcal{Q}$ can now simply be constructed by building separate quadtrees for all non-empty cells created this way. As the height of each of these separate quadtrees is $O(\log\Psi)$, this takes an overall time of $O(n\log\Psi)$. Following the argument in the proof of Lemma 5.6 these levels are enough to identify all cells in $\mathcal{A}$. We store the roots of the quadtrees in a balance binary search tree. This way, we can access the quadtree containing a site s or a cell σ with $|\sigma|\le\lfloor\log\Psi\rfloor$ in $O(\log n)$ additional time.

Now there might be some cells in $\bigcup_{s\in S}N(s)$ that are not yet added to the forest. For each cell in $\bigcup_{s\in S}N(s)$, we can traverse the matching quadtree of the forest to find the position the cell belongs to. If it is already in the quadtree we are done, in the other case we create a leaf and connect it to the quadtree with a matching path. In the case that a new quadtree containing the cell has to be created, its root is added to the binary search tree in $O(\log n)$ time. This takes $O(\log\Psi+\log n)$ time for each cell, resulting in an $O(n(\log\Psi+\log n))$ overall time.

Now we are equipped to find the sets $S_1(A)$ and $S_2(A)$. Fix a site t, by Lemma 5.3 we only have to consider the regions defined by cells in $N(t)$ to find all sets $S_1(A)$ with

$t \in S_1(A)$. For each cell in $N(t)$, we can iterate all regions defined by it, and find the sets $S_1(A)$ containing t in constant time. Finding the cells in $N(t)$ and iterating the regions takes $O(\log n + \log \Psi)$ time for each cell in $N(t)$ for an $O(n(\log n + \log \Psi))$ overall time.

In order to be able to find the sets $S_2(A)$, we build a static additively weighted nearest neighbor data structure on all sets $S_1(A)$. Each site $t \in S_1(A)$ gets assigned a weight of $-r_t$. By Lemma 5.1, these data structures can be constructed in $O(|S_1(A)| \log n)$ time each, while allowing a query time of $O(\log n)$. As $\sum_{A \in \mathcal{A}} |S_1(A)| = O(n)$ by Lemma 5.6, we need $\sum_{A \in \mathcal{A}} |S_1(A)| \log n = O(n \log n)$ time to compute all data structures in addition to the $O(n \log \Psi)$ time needed to traverse the forest of quadtrees. For a site $s \in S$, let π_s be the path in $\mathcal{Q}$ from the root to the cell σ_s with $s \in \sigma_s$ and $|\sigma_s| \le r_s < 2|\sigma_s|$. For each cell along π_s, we query all nearest neighbor data structures with s. If s intersects the reported weighted nearest neighbor, we add it to $S_2(A)$. As we use the negative additive weight for the nearest neighbor data structure, we do not miss a site this way [Sei16, Lemma 2.8]. Walking along the path for a fixed site s takes $O(\log n + \log \Psi)$ time. We perform $O(1)$ queries to an AWNN in each cell along this path, for a total time of $O(n \log n \log \Psi)$.

The vertices of H are now determined by the sets S and $\mathcal{A}$. We then insert the edges one by one in overall $O(n \log \Psi \log^2 n)$ time into an initially empty Holm et al. data structure and obtain the connectivity data structure $\mathcal{H}$.

Having the sets $S_1(A)$ and $S_2(A)$ at hand, by Lemma 5.2 we can build the reveal data structures with $B = S_1(A)$ and $R = S_2(A)$ in $O\big(|S_1(A)| \log^5(n) \lambda_6(\log(n)) + |S_2(A)| \log^3(n)\big)$ expected time for each region $A \in \mathcal{A}$. The total time is then dominated by summing over all regions and using Lemma 5.6:

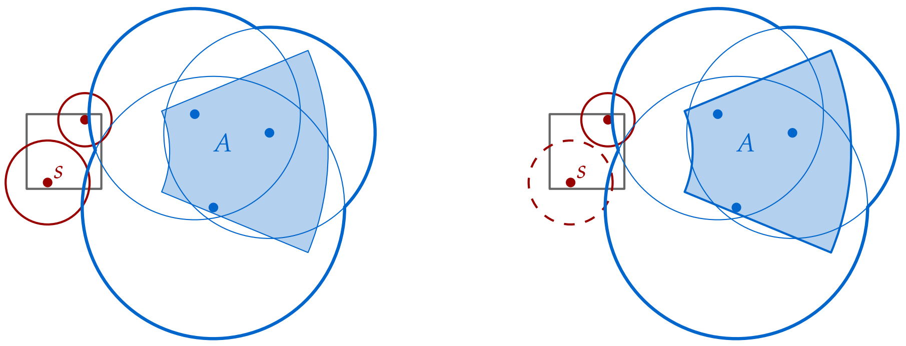
$$
\begin{aligned}
&O\left(\sum_{A \in \mathcal{A}} |S_1(A)| \log^5(n) \lambda_6(\log n) + |S_2(A)| \log^3(n)\right) \\
&= O\left(\left(\sum_{A \in \mathcal{A}} |S_1(A)|\right) \cdot \log^5(n) \lambda_6(\log n) + \left(\sum_{A \in \mathcal{A}} |S_2(A)|\right) \cdot \log^3(n)\right) \\
&= O\big(n \log^5 n \lambda_6(\log n) + n \log \Psi \log^3(n)\big) \qquad \square
\end{aligned}
$$

Now we show that the data structure is correct and efficiently handles the queries.

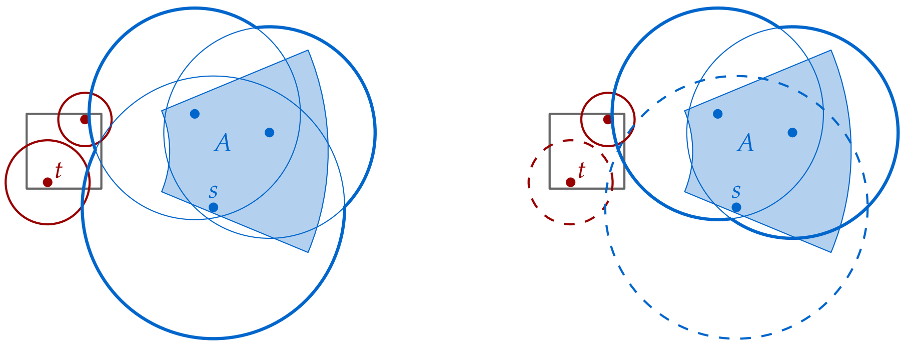
Theorem 5.9. *The data structure described above correctly answers connectivity queries in* $O\left(\frac{\log n}{\log \log n}\right)$ *time and handles* m *site deletions in* $O\big((n \log^5 n + m \log^9 n) \lambda_6(\log n) + n \log \Psi \log^4 n\big)$ *overall expected time.*

Proof. We first show that the answers given by our data structure are indeed correct. During the lifetime of the data structure, we maintain the invariant, that the sets $S_1(A)$ and $S_2(A)$ always contain the sites as defined above, the graph stored in $\mathcal{H}$ is the proxy graph H and each RDS associated with a region A contains the sets $S_1(A)$ and $S_2(A)$. Assuming that this invariant holds, Lemma 5.5 implies that the answers given for each query are correct.

To show that the invariant is maintained, we first note that removing a site from a set $S_2(A)$, does only lead to the deletion of a single edge from H, see Figure 5.8(a). As we



(a) On deleting s from $S_2(A)$, no other sites are affected.



(b) On deleting s from $S_1(A)$, we also have to delete t from $S_2(A)$.

Figure 5.8: The invariant is maintained.

make sure to mirror the removal from $S_2(A)$ in $\mathcal{H}$ and the RDS, removing the site s from all sets $S_2(A)$ containing it, maintains the invariant. Now let A be a region, such that the site s lies in $S_1(A)$. Then for all sites in the matching set $S_2(A)$, it is necessary to intersect at least one site in $S_1(A)$. Furthermore, there is a, possibly empty, subset U' of sites in $S_2(A)$ that only intersect s, see Figure 5.8(b). So to maintain the invariant, we have to delete these sites from $S_2(A)$ and the matching RDS. As these sites are exactly the sites reported in the set U, this set is removed by the data structure and the invariant on $S_2(A)$ and the RDS is maintained. As we do not delete the edges that were present because a site was in $S_1(A) \cap S_2(A)$, the graph stored in $\mathcal{H}$ is still the proxy graph H and the invariant is maintained.

Now we can focus on the analysis of the running time. Queries are performed in $\mathcal{H}$ with a running time of $O\left(\frac{\log n}{\log\log n}\right)$. Each edge is removed exactly once from $\mathcal{H}$, for a total of $O(n \log \Psi \log^2 n)$ time. Locating the sets $S_1(A)$ that have to be changed, can be done by querying all cells of $N(s)$ in the forest of quadtrees. As there are $O(1)$ such cells, this takes an overall of $O(\log \Psi + \log n)$ time. Similarly, all sets $S_2(A)$ containing s lie on one path in a quadtree that can be found with the same running time.

The step dominating the running time are the deletions from the RDS. By Lemma 5.2, the RDS associated to a single region adds at most an expected running time of $O\Big((|S_1(A)| \log^5 n + m_A \log^{11} n)\lambda_6(\log n) + |S_2(A)| \log^4 n\Big)$ to the total running time, where m_A is the number of sites deleted from $S_1(A)$. Summing over all regions, we have $\sum_{A \in \mathcal{A}} m_A = O(m)$ as each site is contained in $O(1)$ sets $S_1(A)$. Furthermore, as we have $\sum_{A \in \mathcal{A}} |S_1(A)| = O(n)$ and $\sum_{A \in \mathcal{A}} |S_2(A)| = O(n \log \Psi)$, the total expected running time of $O\Big((n \log^5 n + m \log^9 n)\lambda_6(\log n) + n \log \Psi \log^4 n\Big)$ for m deletions follows. □

5.2 General Disk Graphs

In this section, we extend the approach from Section 5.1 to give a data structure that allows update times independent of Ψ. The cost for dropping the dependence on Ψ is exchanging the additive $O(n \log \Psi \log^4 n)$ term in the running time in Theorem 5.9 with an additional $O(\log n)$ factor in the first term. The additive $O(n \log \Psi \log^4 n)$ term in Section 5.1 came from the size of the sets $S_2(A)$ and thus from the height of each quadtree in the forest of quadtrees. We can get rid of this dependency by using a compressed quadtree $\mathcal{Q}^c$. Recall from Section 2.1 that the height and size of the compressed quadtree do not depend on the diameter of the point set, but only on n. However the height of $\mathcal{Q}^c$ can still be $O(n)$ which is not favorable for our application. In order to reduce the number of edge in our proxy graph to $O(n \log^2 n)$, we use a heavy path decomposition, similar to the one defined by Sleator and Tarjan [ST83] on $\mathcal{Q}^c$, in combination with the range query framework from Lemma 2.4 for each heavy path.

The Proxy Graph We will often refer back to Section 5.1, as the proxy graph for general graphs is similar to that in the bounded radius ratio case. We still have a bipartite graph with the sites from S on one side and a set of region on the other side. The regions will again be used to define sets $S_1(A)$ and $S_2(A)$ that will in turn define the edges. However, we adapt the definition of the regions and define them based on certain paths of the compressed quadtree instead of single cells. Furthermore, we relax the condition on the radii in the definition of the set $S_1(A)$.

Similar to the non-compressed approach, we consider an augmented version of the compressed quadtree on S. Again let $N(s)$ be the 15×15 neighborhood of the cell σ containing s such that $|\sigma| \leq r_s < 2|\sigma|$. We work on the tree $\mathcal{Q}^c$ that is the union of the compressed quadtree with the neighborhoods of all sites $s \in S$.

On this tree we build a *heavy path decomposition* following Sleator and Tarjan [ST83]. Let T be a rooted ordered tree. An edge $\{u, v\} \in T$ is called *heavy* if v is the first child of u in the given child-order that maximizes the total number of nodes in the subtree rooted at v among all children of u. Otherwise, the edge $\{u, v\}$ is *light*. By definition, every internal node in T has exactly one child that is connected by a heavy edge, see Figure 5.9 for an illustration.

A *heavy path* is a maximal path in T that consists only of heavy edges. The *heavy path decomposition* of T is the set of all the heavy paths in T. The following lemma summarizes a classic result on the properties of heavy path decompositions.

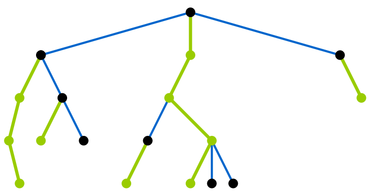


Figure 5.9: A tree with its heavy path decomposition. The heavy edges are marked green.

Lemma 5.10 (Sleator and Tarjan [ST83]). *Let T be a tree with n vertices. Then, the following properties hold:*

(a) Every leaf-root path in T contains $O(\log n)$ light edges,

(b) every vertex of T lies on exactly one heavy path; and

(c) the heavy path decomposition of T can be constructed in $O(n)$ time.

Now let $\mathcal{R}$ be the heavy path decomposition of $\mathcal{Q}^c$ and let $R \in \mathcal{R}$ be a heavy path. When ordering the cells in the heavy path by their diameter, we can apply the canonical decomposition framework given in Lemma 2.4 to this ordered set. Note that each of the $O(n \log n)$ canonical subset of the cells in P defined this way, is a subpath of R in $\mathcal{Q}^c$. As our arguments using this decomposition focus on the property, that each canonical subset corresponds to a path in $\mathcal{Q}^c$, in the following we use the term *canonical path* instead of canonical subset.

Lemma 5.11. *Let σ be a vertex of $\mathcal{Q}^c$ and let π be the path from the root of $\mathcal{Q}^c$ to σ. Then there exists a set $\mathcal{P}_\pi$ of canonical paths such that:*

(a) $|\mathcal{P}_\pi| = O(\log^2 n)$; and

(b) π is the disjoint union of the canonical path in $\mathcal{P}_\pi$.

Proof. Consider the heavy paths $R_1, \dots, R_k$ encountered along π. By Lemma 5.10, we have that $k \in O(\log n)$ and that π is the disjoint union of the intersections $\pi \cap R_i$. Each of these intersections constitutes a subpath of R_i whose largest cell is also the largest cell of R_i. Let σ_i be the smallest cell of $\pi \cap R_i$, then the subpath of R_i consists of all cells in R_i with a diameter in the interval $[|\sigma_i|, \infty)$. Using this interval in the framework given in Lemma 2.4, decomposes $\pi \cap R_i$ into $O(\log n)$ canonical paths. As $k \in O(\log n)$ this results in overall $O(\log^2 n)$ canonical paths needed to represent π. The argument is depicted in Figure 5.10. □

The vertex set of the proxy graph H again consists of the set of sites S and a set of regions $\mathcal{A}$. We define $O(1)$ regions for each canonical path in a similar way as we defined $O(1)$ regions for each cell in Section 5.1. Let σ be the smallest cell and τ be the largest cell of a canonical path. The *inner* and *middle regions* are defined as in the bounded case, using the smallest cell of the path to define the region, instead of the single cell in

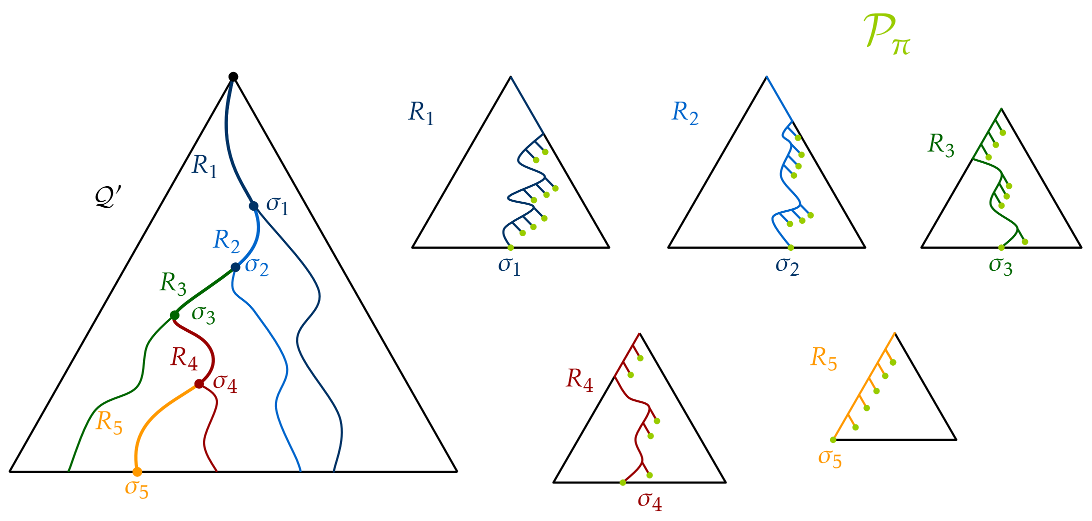


Figure 5.10: Illustration of Lemma 5.11. Left we see the decomposition of R into $R_1, \dots R_k$. On the right the vertices defining $\mathcal{P}_\pi$ are depicted in green.

Section 5.1. More specifically, the inner region is the disk with center $a(\sigma)$ and radius $|\sigma|$. The d_2 middle regions are defined, by intersecting the annulus centered at $a(\sigma)$ with inner radius $|\sigma|$ and outer radius $\frac{5}{2}|\sigma|$ with the cones in $\mathcal{C}_{d_2}$. For the *outer regions*, we extend the outer radius of the annulus. They are defined as the intersections of the annulus of inner radius $\frac{5}{2}|\sigma|$ and outer radius $\frac{5}{2}|\sigma| + 2|\tau|$, again centered at $a(\sigma)$ with the cones in $\mathcal{C}_{d_1}$. The set $\mathcal{A}$ contains the regions defined this way for all canonical paths.

Given a region $A \in \mathcal{A}$ with smallest cell σ and largest cell τ, we define the sets $S_1(A)$ and $S_2(A)$. The set $S_1(A)$ is defined similarly to the set in Section 5.1, again using σ in the role of the single cell for most parts. The difference is that the radius range for t is larger, as its upper bound depends on the diameter of τ. The set $S_1(A)$ contains all sites t such that

(a) $t \in A$

(b) $|\sigma| \leq r_t \leq 2|\tau|$; and

(c) $\|a(\sigma)t\| \leq r_t + \frac{5}{2}|\sigma|$.

The last condition is only relevant, if A is an outer region, as it is trivially true for middle and inner regions.

The intuition behind the definition for the sets $S_2(A)$ is also similar to the one in Section 5.1, but its formal definition does not mirror these similarities directly. Let σ_s be the cell such that $s \in \sigma_s$ and $|\sigma_s| \leq r_s < 2|\sigma_s|$. Let π_s be the path in $\mathcal{Q}^c$ from the root to σ_s and let $\mathcal{P}_{\pi_s}$ be the decomposition of π_s into canonical paths as defined in Lemma 5.11.

Let A be a region, defined by a canonical path P, then $s \in S_2(A)$ if

(a) $P \in \mathcal{P}_{\pi_s}$; and

(b) s is adjacent in $\mathcal{D}(S)$ to at least one site in $S_1(A)$.

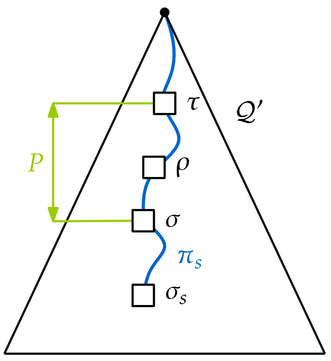


Figure 5.11: σ_s is the smallest cell such that $r_s \leq 2|\sigma_s|$. P contains ρ.

If σ is the smallest cell in a canonical path defining a region A, then each site s in $S_2(A)$ lies in σ, has a radius less than $2|\sigma|$ and intersects at least one cell in $S_2(A)$. These are basically the same conditions we had in Section 5.1. However as the definition is restricted to the canonical paths in $\mathcal{P}_{\pi_s}$, not all cells satisfying these conditions are considered. As we will see below, using only the paths in $\mathcal{P}_{\pi_s}$ suffices to make sure that the proxy graph represents the connectivity, while also ensuring that each site s lies in few sets $S_2(A)$.

The graph H is now again defined by connecting a site $s \in S_1(A) \cup S_2(A)$ to the region A. To show that H accurately represents the connectivity in $\mathcal{D}(S)$, we need the following corollary of Lemma 5.4.

Corollary 5.12. *Suppose that $d_1 \geq 23$ and $d_2 \geq 8$. Then, for any region $A \in \mathcal{A}$, the associated sites in $S_1(A)$ form a clique in $\mathcal{D}(S)$.*

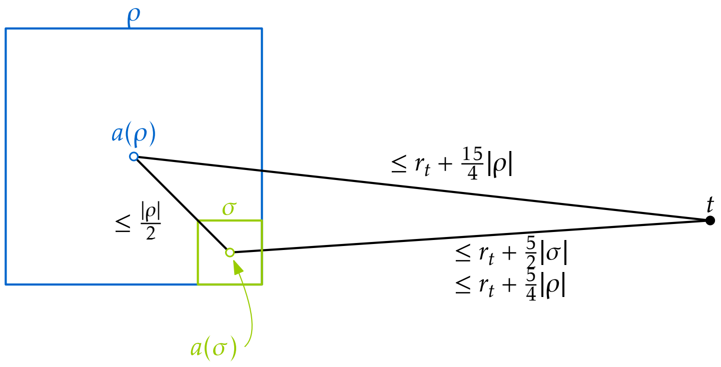
Proof. Recall that the center of the annuli and disks defining A is the center of smallest cell of the associated canonical path. A close inspection of the proof for Lemma 5.4 shows that we only use the lower bound on the radii of the sites in $S_1(A)$. As this lower bound is unchanged, all arguments carry over for sites with larger radii. □

Following the approach in Section 5.1 we now show that H accurately represents the connectivity of $\mathcal{D}(S)$ and that H is sparse.

Lemma 5.13. *Two sites s and t are connected in $\mathcal{D}(S)$ if and only if they are connected in H.*

Proof. If s and t are connected in H, the same argument as in the proof of Lemma 5.5 with Corollary 5.12 instead of Lemma 5.4 holds. The more challenging part is to show that if two sites are connected in $\mathcal{D}(S)$ they are also connected in H.

If suffices to show that if s and t are connected by an edge in $\mathcal{D}(S)$, they are connected to the same region $A \in \mathcal{A}$. Refer to Figure 5.11 for a depiction of the following argument. Assume without loss of generality that $r_s \leq r_t$. Consider the neighborhood $N(t)$ of t. Then by Lemma 5.3 (a) there is a cell $\rho \in N(t)$ that contains s. Consider the path π_s in $\mathcal{Q}^c$ from the root to the cell σ_s such that $s \in \sigma_s$ and $|\sigma_s| \leq r_s < 2|\sigma_s|$. The cell σ_s is in $\mathcal{Q}^c$ as it is part of $N(s)$. Then ρ lies on this path π_s. Let $\mathcal{P}_{\pi_s}$ be the decomposition of π_s into canonical paths as defined in Lemma 5.11 and let P be the path containing ρ. Again let σ and τ be the smallest and largest cell on P respectively. By the definition of P we



ρ
$a(\rho)$
$\leq r_t + \frac{15}{4}|\rho|$
t
$\leq \frac{|\rho|}{2}$
σ
$\leq r_t + \frac{5}{2}|\sigma|$
$\leq r_t + \frac{5}{4}|\rho|$
$a(\sigma)$

Figure 5.12: The cell ρ is in $N(t)$.

have $\sigma_s \subseteq \sigma \subseteq \rho \subseteq \tau$. As $\{s,t\}$ is an edge in $\mathcal{D}(S)$, we have $\|st\| \leq r_s + r_t \leq 2|\sigma| + 2|\tau|$ and thus $\|a(\sigma)t\| \leq \frac{5}{2}|\sigma| + 2|\tau|$. This implies that t lies in a region A defined by P and thus $t \in S_1(A)$ for this region. As s intersects t, it intersects at least one site in $S_1(A)$ for the region A. Furthermore, P is a canonical path in $\mathcal{P}_{\pi_s}$ and thus $s \in S_2(A)$ by definition. As $s, t \in S_1(A) \cup S_2(A)$, they are connected in H. □

Lemma 5.14. *The graph H has $O(n)$ vertices and $O(n \log^2 n)$ edges.*

Proof. As discussed in Section 2.3 the compressed quadtree consists of $O(n)$ cells. Each cell is part of exactly one heavy path, so the total size of the binary search trees that define the canonical paths is $O(n)$. A balanced binary search tree with n leafs has $O(n)$ inner vertices, and thus there is a total of $O(n)$ canonical paths. As each canonical path defines $O(1)$ regions, the number of regions and therefore also the number of vertices in H follows.

To bound the number of edges, we again count the number of sets $S_1(A)$ and $S_2(A)$ a single site can be contained in. Let σ and τ be the smallest and largest cell of a canonical path respectively. A site t can only be in a set $S_1(A)$ if $|\sigma| \leq r_t \leq 2|\tau|$ and σ is contained in a cell of $N(t)$. To see the second part, let P be a canonical path such that $t \in S_1(A)$, with smallest cell σ. If $|\sigma| \leq r_t < 2|\sigma|$, the statement holds by Lemma 5.3 (b). In the other case, let ρ be the cell of the hierarchical grid with $2|\sigma| \leq |\rho| \leq r_t < 2|\rho|$ and $\sigma \subset \rho$. See Figure 5.12 for an illustration of the following argument. As $t \in S_1(A)$, we have

$$\begin{aligned} \|ta(\sigma)\| &\leq r_t + \frac{5}{2}|\sigma| \\ &\leq 2|\rho| + \frac{5}{4}|\rho| && \text{since } r_t < 2|\rho| \text{ and } |\sigma| \leq \frac{|\rho|}{2} \\ &\leq \frac{13}{4}|\rho| \\ \|ta(\rho)\| &\leq \frac{15}{4}|\rho| && \text{by triangle inequality as } \sigma \subseteq \rho \end{aligned}$$

Lemma 2.1 then implies that $\rho \in N(t)$ and thus ρ is in $\mathcal{Q}^c$ and also in P.

Now let ρ be a cell of $N(t)$, and let R be the heavy path containing ρ. Then t is considered for the set $S_1(A)$ for all regions which are defined by a canonical path

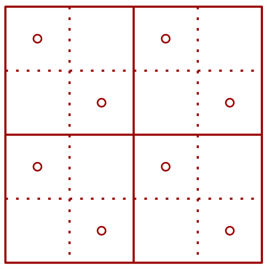


Figure 5.13: Four cells from $N_{15\times15}(\sigma)$ with the virtual sites.

containing ρ. These, are the $O(\log n)$ canonical paths defined by the canonical vertices of the search path $\Pi^{\circ}_{\leftarrow}(\rho)$ for ρ in the binary search tree on the cells of R. Thus each site can be part of at most $O(\log n)$ sets $S_1(A)$.

The number of sets $S_2(A)$, a fixed site s can be part of is at most the number of canonical paths decomposing the path from the root to the smallest cell containing s. By Lemma 5.11 there are $O(\log^2 n)$ such paths. This yields $\sum_{A\in\mathcal{A}} |S_1(A)|+|S_2(A)| = O(n\log^2 n)$ as an upper bound for the number of edges. □

The Data Structure The structure of the decremental data structure is the same as in Section 5.1. We again store the sets $S_1(A)$ and $S_2(A)$ together with a RDS for each region A. The set B for the RDS is again $S_1(A)$ and the set R is $S_2(A)$. Furthermore, we store the graph H defined by $S_1(A)$ and $S_2(A)$ in a Holm et al. [HdLT01] data structure $\mathcal{H}$.

Both queries and deletions work exactly as in Section 5.1, however we repeat them here for completeness. Queries are performed on $\mathcal{H}$. On the deletion of a site s, we first remove all edges incident to s from $\mathcal{H}$. Then the site is removed from all sets $S_1(A)$ containing it, as well as the associated RDS. The sites U reported as revealed by the RDS are then removed from the matching $S_2(A)$ and the edges $\{u, A\}$ for $u \in U \setminus S_1(A)$ are removed from $\mathcal{H}$. Finally the site s is removed from all sets $S_2(A)$ and the matching RDS.

Lemma 5.15. *The data structure described above can be preprocessed in $O(n\log^6(n)\lambda_6(\log n))$ expected time.*

Proof. To find the regions $\mathcal{A}$ we first compute the extended compressed quadtree $\mathcal{Q}^c$. This can be done by adding $O(n)$ virtual sites to our site set, similar to a construction of Har-Peled [Har11]. For each site $s \in S$ and each cell $\sigma \in N(s)$ we add two virtual sites. The virtual sites are added at the center of two of the cells one level below σ that are contained in σ, see Figure 5.13 for an illustration. Now all cells in $N(s)$ have at least two children in the non-compressed quadtree and thus the cells are also present in the compressed quadtree. When having $\mathcal{Q}^c$ at hand, we can, with additional $O(n)$ time, find the heavy paths by Lemma 5.10 (c) and the binary search trees on the heavy paths by standard techniques. This gives us the set of regions $\mathcal{A}$.

To find the set $S_1(A)$, recall from the proof of Lemma 5.14 that a site t can only lie in the set $S_1(A)$ for a region defined by a canonical path that contains a cell in $N(t)$. The sets $S_1(S)$ can now be found as follows. For each site $t \in S$ compute the cells in $N(t)$ and for each cell ρ find the heavy path R containing it. In the binary search tree defined on R, follow the search path for ρ and for each canonical path defined along this search path explicitly find the region containing t and check if the distance condition holds.

When implementing this step naively it takes $O(n\log^2 n)$ time which is fast enough for our purposes. We could however reduce this time to $O(n\log n)$ if we added an additional preprocessing step on $\mathcal{Q}^c$ and distributed the sites to the corresponding paths in a batched fashion. As in Lemma 5.8, we construct a static additively weighted nearest neighbor data structure on each set $S_1(A)$, again assigning the weight $-r_s$ to each site $s \in S$. Recall from Lemma 5.1, that the time needed to construct a single nearest neighbor data structure is $O(|S_1(A)|\log n)$. As $\sum_{A\in\mathcal{A}}|S_1(A)| = O(n\log n)$, the construction of all data structures takes $O(n\log^2 n)$ time. The query time remains $O(\log n)$ in each data structure, as each data structure contains at most $O(n)$ sites.

Recall from the definition of the sets $S_2(A)$, that π_s is the path in $\mathcal{Q}^c$ from the root to the cell σ_s that contains s and has diameter $|\sigma| \le r_s < 2|\sigma|$. To find all sets $S_2(A)$ containing s, we simply follow the decomposition of π_s into canonical paths and query the nearest neighbor data structure with s for all regions defined by these paths. As there are $O(\log^2 n)$ canonical paths in the decomposition, this takes an additional $O(n\log^3 n)$ time for all sites. Inserting the $O(n\log^2 n)$ edges into $\mathcal{H}$ takes $O(\log^2 n)$ amortized time each, for a total of $O(n\log^4 n)$.

Again the step dominating the preprocessing time is the construction of the RDS. For a single region the expected time is $O(|S_1(A)|\log^5 n\lambda_6(\log n)+|S_2(A)|\log^3(n))$ by Lemma 5.2. We have $\sum_{A\in\mathcal{A}}|S_1(A)| = O(n\log n)$ and $\sum_{A\in\mathcal{A}}|S_2(A)| = O(n\log^2 n)$ and thus the claimed preprocessing time follows. □

Theorem 5.16. *The data structure described above correctly answers connectivity queries in $O\left(\frac{\log n}{\log\log n}\right)$ time with $O\Big((n\log^6 n + m\log^{10} n)\lambda_6(\log n)\Big)$ overall expected update time for m deletions.*

Proof. As the only difference in the definition of the data structure is the definition of the sets $S_1(A)$ and $S_2(A)$, the correctness follows from Theorem 5.9. Also the $O\left(\frac{\log n}{\log\log n}\right)$ bound for the queries in $\mathcal{H}$ again carries over. The preprocessing of the data structure takes $O(n\log^6 \lambda_6(\log n))$ time by Lemma 5.15. Deletions from $\mathcal{H}$ take $O(\log^2 n)$ amortized time for each of the $O(n\log^2 n)$ edges for an overall time of $O(n\log^4 n)$. The paths defining a region with a set $S_1(A)$ or $S_2(A)$ that have to be updated can be found in $O(\log^2 n)$ time, by traversing the underlying tree structure.

The time for the sequence of deletions is again dominated by the time needed for the updates in the RDS. Let m_A be the number of sites deleted from $S_1(A)$ for a region A. Then the time needed by the RDS associated with A is $O\Big((|S_1(A)|\log^5 n + m_A\log^9 n)\lambda_6(\log n) + |S_2(A)|\log^4 n\Big)$ by Lemma 5.2. We have $\sum_{A\in\mathcal{A}} m_A = O(m\log n)$, $\sum_{A\in\mathcal{A}}|S_1(A)| = O(n\log n)$ and $\sum_{A\in\mathcal{A}}|S_2(A)| = O(n\log^2 n)$. Summing up the time needed for the RDS over all $A \in \mathcal{A}$, we get a running time of $O\Big((n\log^6 n + m\log^{10} n)\lambda_6(\log n)\Big)$ as claimed. □

II

Long Plane Trees

CHAPTER 6

Approximating the Longest Tree

This chapter considers problems related to approximating the longest plane spanning tree. Let T_{OPT} be a plane spanning tree of maximum length for a site set S. We call a tree T with $|T| \geq \delta \cdot |T_{\text{OPT}}|$ a δ-approximation. In Sections 6.2 and 6.4 we also deal with *flat site sets*. A set S is flat if $\text{diam}(S) \geq 1$ and the y-coordinates of all its points are essentially negligible, meaning that their absolute values are bounded by an infinitesimal $\varepsilon > 0$. For flat site sets, we can estimate the length of each edge as the difference between the x-coordinates of its endpoints, because the error can be made arbitrarily small by taking $\varepsilon \to 0$.

In Section 6.1, we describe a simple approximation algorithm. In Section 6.2 we show that this algorithm yields a $\frac{2}{3}$-approximation when applied to a flat set. Furthermore, we show that the analysis for this algorithm is tight when comparing to the longest crossing tree. In Section 6.3, we then show that the same algorithm yields a $\delta \doteq 0.5467$ approximation when applied to general site sets.

The trees returned by algorithms found in the literature [ARS95; Bin20b; Bin+19; DT10] as well as our algorithm, have a small diameter. In Section 6.4, we study the relation between longest plane spanning trees of small diameter and those of large diameter. We show that there is a flat point set S for that a longest spanning tree of diameter at most three can only approximate a longest plane spanning tree up to a factor of $\frac{5}{6}$.

6.1 A Simple Approximation Algorithm

The algorithm described in this section will be analyzed in detail in Sections 6.2 and 6.3. For the algorithm, we consider two kinds of trees. The first kind are stars T_a as defined in Section 2.1 Additionally, the algorithm considers trees $T_{a,b}$, for $a, b \in S$, that are defined as follows: Let S_a be the sites of S closer to a than to b and let $S_b = S \setminus S_a$. First, connect a to every site in S_b. Second, connect each site of $S_a \setminus \{a\}$ to some site of S_b without introducing crossings. Note that as we did not fix a systematic way to connect the sites in S_a, the tree $T_{a,b}$ is not uniquely determined. However, after the deterministic first step, we can already see that $T_{a,b}$ and $T_{b,a}$ are different in general. Also, if $S_a = \{a\}$, then $T_{a,b}$ and $T_{b,a}$ are the same tree that degenerates to S_a. In the following lemma, we give a systematic way to efficiently construct such a tree $T_{a,b}$.

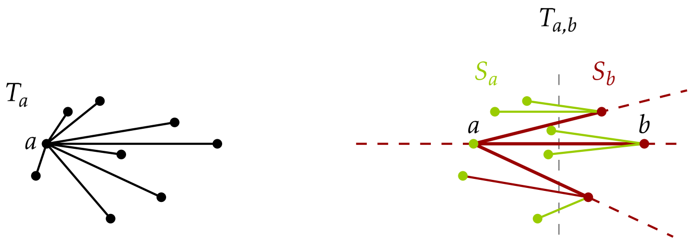


Figure 6.1: A star T_a and a tree $T_{a,b}$, the rays bounding the cones are drawn dashed.

Lemma 6.1. *Given two sites $a, b \in S$, a plane tree $T_{a,b}$ can be constructed in $O(n \log n)$ time.*

Proof. Recall that S_a contains all sites closer to a than to b and $S_b = S \setminus S_a$. The procedure above already states that in $T_{a,b}$, the site a is connected to all sites in S_b. In order to make the procedure above deterministic, we describe a systematic way to connect the sites in S_b to the sites in $S_a \setminus \{a\}$. The rays $\overrightarrow{as}$ for $s \in S_b$ together with the ray opposite to $\overrightarrow{ab}$ partition the plane into convex cones with common apex a. Each such convex cone C contains a site $b_C \in S_b$ on the bounding ray that forms the smaller (convex) angle with $\overrightarrow{ab}$. Within each cone C we then connect all sites of $C \cap (S_a \setminus \{a\})$ to b_C. A tree constructed this way can be seen in Figure 6.1.

This systematic approach can be implemented efficiently by first sorting the points by their angular order around a. After the sorting, we do a traversal of the points above $\{a, b\}$ in counterclockwise order, starting with b. During this traversal, the last site t in S_b that we encountered, is stored. We initialize $t = b$. When we are at a site $s \in S_a$ we add the edge $\{s, t\}$ to $T_{a,b}$. If we encounter a site $s \in S_b$, we update t and add the edge $\{a, t\}$. To also connect the remaining sites, the process is repeated in clockwise order for the points below the line through ab. This takes $O(n \log n)$ overall time and yields the tree $T_{a,b}$. The resulting tree is a plane tree because we add stars within each convex wedge and the interiors of the wedges are pairwise disjoint. □

Given the above definition of the trees $T_{a,b}$, consider the following algorithm that constructs a tree T_{ALG}:

```
1: Algorithm AlgSimple(S)
2:     $a' \leftarrow \arg\max_{a \in S} |T_a|$
3:     $(a^*, b^*) \leftarrow \arg\max_{(a,b) \in S \times S} |T_{a,b}|$
4:     $T_{\text{ALG}} \leftarrow \arg\max(|T_{a'}|, |T_{a^*,b^*}|)$
5:     return $T_{\text{ALG}}$
```

The algorithm simply enumerates all possible stars, and trees $T_{a,b}$ and $T_{b,a}$, and returns the longest such tree. As all those trees are plane, the resulting tree is guaranteed to also be plane. Furthermore, there are $O(n^2)$ candidate trees. Each can be found in $O(n)$ time in the case of stars, or in $O(n \log n)$ time for the remaining trees by Lemma 6.1. This results in an overall running time of $O(n^3 \log n)$.

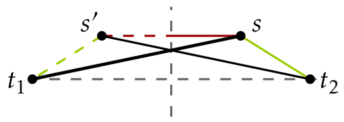


Figure 6.2: By triangle inequality and symmetry we have $\|s's\| + \|st\| \geq \|s't\| = \|sr\|$.

6.2 A $\frac{2}{3}$-Approximation for Flat Sets

In this section we show that `AlgSimple(S)` as described in Section 6.1 yields a $\frac{2}{3}$-approximation of the longest plane spanning tree. Furthermore, we give a site set that shows that this analysis is tight.

Theorem 6.2. *Let S be a flat site set, let T_{ALG} be the result of* `AlgSimple` *and T_{cr} be a fixed longest, possibly crossing spanning tree. Then $|T_{\mathrm{ALG}}| \geq \frac{2}{3}|T_{\mathrm{cr}}| \geq \frac{2}{3}|T_{\mathrm{OPT}}|$.*

Proof. Since $|T_{\mathrm{cr}}| \geq |T_{\mathrm{OPT}}|$, it suffices to prove the first inequality. Let t_1 and t_2 be the sites realizing the diameter of S, see Figure 6.2. Consider the four trees T_{t_1}, T_{t_1,t_2}, T_{t_2,t_1}, T_{t_2} considered by `AlgSimple`. By using a weighted average, it suffices to show that there is a $\beta \in (0, \frac{1}{2}]$ such that

$$\left(\frac{1}{2} - \beta\right) \cdot |T_{t_1}| + \beta \cdot |T_{t_1,t_2}| + \beta \cdot |T_{t_2,t_1}| + \left(\frac{1}{2} - \beta\right) \cdot |T_{t_2}| \geq \frac{2}{3} \cdot |T_{\mathrm{cr}}|.$$

If this inequality holds, then as

$$\max\left\{|T_{t_1}|, |T_{t_1,t_2}|, |T_{t_2,t_1}|, |T_{t_2}|\right\} \geq \left(\frac{1}{2} - \beta\right) \cdot |T_{t_1}| + \beta \cdot |T_{t_1,t_2}| + \beta \cdot |T_{t_2,t_1}| + \left(\frac{1}{2} - \beta\right) \cdot |T_{t_2}|$$

we have shown the desired statement. We fix $\beta = \frac{1}{3}$ and equivalently show:

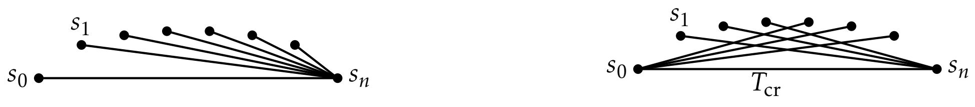
$$\frac{|T_{t_1}| + 2|T_{t_1,t_2}| + 2|T_{t_2,t_1}| + |T_{t_2}|}{6} \geq \frac{2}{3} \cdot |T_{\mathrm{cr}}|.$$

Note that all four trees on the left-hand side include the edge $\{t_1, t_2\}$. Since t_1 and t_2 realize the diameter and T_{cr} allows crossings, we can without loss of generality assume that T_{cr} contains that edge too. Thus we can direct all edges towards $\{t_1, t_2\}$ and use the notation $\ell_T(s)$ as defined in Section 2.1. Assume the following holds for all $s \in S \setminus \{t_1, t_2\}$:

$$\frac{\ell_{T_{t_1}}(s) + 2\ell_{T_{t_1,t_2}}(s) + 2\ell_{T_{t_2,t_1}}(s) + \ell_{T_{t_2}}(s)}{6} \geq \frac{2}{3} \cdot \ell_{T_{\mathrm{cr}}}(s), \tag{6.1}$$

then summing over all points $s \in S \setminus \{t_1, t_2\}$ and adding $\|t_1 t_2\|$ to both sides yields the desired result.

Fix a site s and assume without loss of generality, that $\|st_1\| \geq \|st_2\|$. Let s' be the reflection of s at the perpendicular bisector of $\{t_1, t_2\}$. Since S is flat, we have $\ell_{T_{\mathrm{cr}}}(s) \leq \max\{\|st_1\|, \|st_2\|\} = \|st_1\|$. For the terms on the left hand side, we have $\ell_{T_{t_1}} = \ell_{T_{t_1,t_2}} = \|st_1\|$



(a) The best plane spanning tree of S_n. (b) The best crossing spanning tree of S_n.

Figure 6.3: The point set S_n consisting of $n+1$ points with equally spaced x-coordinates $0, 1, \dots, n$.

and $\ell_{T_{t_2}} = \|st_2\|$ by the definition of the trees. Furthermore, as the edge incident to s in T_{t_2,t_1} at least crosses the bisecting line between t_1 and t_2, we have $\ell_{T_{t_2,t_1}}(s) \le \frac{\|ss'\|}{2}$.

Using the triangle inequality in $\triangle t_2 s s'$, yields the desired result for the left-hand side of (6.1), see Figure 6.2.

$$\begin{aligned}\frac{\|st_1\| + 2\|st_1\| + \|s's\| + \|st_2\|}{6} &\ge \frac{\|s't_2\| + 3\|st_1\|}{6} \\ &= \frac{2}{3} \cdot \|st_1\| \\ &\ge \frac{2}{3} \cdot \ell_{T_{\mathrm{cr}}}(s)\end{aligned}$$ □

In the following lemma we give a family of site sets that shows that the constant $\frac{2}{3}$ is asymptotically tight when comparing the result to the best crossing spanning tree.

Lemma 6.3. *There is an infinite family of sets of sites $S_1, S_2, \dots$ where S_n contains $n+1$ sites and*

$$\lim_{n\to\infty} \frac{|T_{\mathrm{OPT}}|}{|T_{\mathrm{cr}}|} \le \frac{2}{3}$$

Proof. The set $S_n = s_0, \dots, s_n$ is a flat set of sites where all sites lie evenly spaced on a convex arc with x-coordinates $0, \dots, n$, as shown in Figure 6.3(a). As the point set is symmetric, we have $|T_{s_0}| = |T_{s_n}|$. We argue that the star rooted at s_0 is a longest plane spanning tree of this site set.

First we observe that $|T_{\mathrm{OPT}}| \le |T_{\mathrm{OPT}}(\{s_0, \dots, s_{n-1}\})| + \max_{i=0,\dots n-1} \|s_n s_i\|$. This can be seen as $|T_{\mathrm{OPT}}(\{s_0, \dots, s_{n-1}\})|$ is at least as large as the subtree of T_{OPT} induced by $s_0, \dots, s_{n-1}$ and the edge connecting s_n to this subtree cannot be larger than $\max_{i=0,\dots n-1} \|s_n s_i\|$.

Now we can inductively show that $|T_{\mathrm{OPT}}| = |T_{s_0}|$. For the base case we consider S_1 and as there is only one possible edge, the claim holds. Now we consider the step from S_{n-1} to S_n. The star on S_{n-1} rooted in s_0 is a longest plane spanning tree of S_{n-1} by induction. As the sites $s_0, \dots, s_{n-1}$ in S_n have the same pairwise distances and relative positions as the sites in S_{n-1}, this directly implies that $|T_{\mathrm{OPT}}(\{s_0, \dots, s_{n-1}\})| = |T_{\mathrm{OPT}}(S_{n-1})|$. As s_0 maximizes the distance to s_n among all sites, we have $|T_{\mathrm{OPT}}| \le |T_{\mathrm{OPT}}(s_0, \dots, s_{n-1})| + \|s_n s_0\| = |T_{s_0}|$ as claimed.

The length of T_{OPT} now follows from simply summing up the length of the edges in T_{s_0}.

$$|T_{\text{OPT}}| = \sum_{i=1}^{n} i = \frac{(n+1)n}{2}$$

Let T be the tree so that the sites $s_1, \ldots, s_{\lfloor n/2 \rfloor}$ are connected to s_n and the rest is connected to s_0. This tree is illustrated in Figure 6.3(b). We claim that $|T| = |T_{\text{cr}}|$ and thus its length gives an exact bound on the length of the longest spanning tree. Consider a longest crossing spanning tree. As crossings are allowed $\{s_0, s_n\}$ is an edge in every longest spanning tree. So we can use the notation from above and direct the remaining edges towards $\{s_0, s_n\}$. This gives

$$\begin{aligned} |T_{\text{cr}}| &= \|s_0 s_n\| + \sum_{1 \le i \le n-1} \ell_{T_{\text{cr}}}(s_i) \\ &\le \|s_0 s_n\| + \sum_{1 \le i \le n-1} \max_{j=0,\ldots,n} \|s_i s_j\|. \end{aligned}$$

By the definition of T, we have $\ell_T(s_i) = \max_{j=0,\ldots,n} \|s_i s_j\|$ and thus $|T| \ge |T_{\text{cr}}|$. A straightforward summation gives the precise bound for $|T_{\text{cr}}|$.

$$\begin{aligned} |T_{\text{cr}}| &\ge 2 \sum_{i=\lceil n/2 \rceil}^{n-1} i + n - \left\lceil \frac{n}{2} \right\rceil \\ &\ge n^2 - \left\lceil \frac{n}{2} \right\rceil^2 \\ &\ge \frac{3n^2}{4} - \frac{1}{2}n - \frac{1}{4} \end{aligned}$$

Combining the values for $|T_{\text{OPT}}|$ and $|T_{\text{cr}}|$ we get the desired result:

$$\lim_{n\to\infty} \frac{|T_{\text{OPT}}|}{|T_{\text{cr}}|} \le \lim_{n\to\infty} \frac{\frac{n^2}{2} + \frac{n}{2}}{\frac{3n^2}{4} - \frac{1}{2}n - \frac{1}{4}} = \frac{2}{3} \qquad \square$$

6.3 A $\delta \doteq 0.5467$-approximation for General Sets

This section considers the performance of `AlgSimple` on general site sets. To be precise we want to give a lower bound on δ, such that `AlgSimple` returns a δ-approximation of the longest plane spanning tree. The core part of the proof follows a similar idea as in the proof of Theorem 6.2 in the sense that we use a weighted average of the lengths of four trees $T_a, T_b, T_{a,b}$ and $T_{b,a}$. However, here we do not define those trees via the diameter of the site set, but rather pick the sites that define the longest edge in a fixed optimal tree T_{OPT}. In contrast to the flat case, we consider some other cases first, before arguing about the weighted average length of the trees. We first sketch the overall proof strategy and define some terminology, before going to the technical lemmas.

In the following, assume without loss of generality that $\operatorname{diam}(S) = 2$. Fix an optimal tree T_{OPT} and let $2d = \|ab\|$ be the length of the longest edge $\{a, b\}$ in T_{OPT}. As $\operatorname{diam}(S) = 2$, we get $d \leq 1$. Let t_1 and t_2 be the sites realizing the diameter. If $2d \leq 1/\delta$, then in Lemma 6.4 we show that either T_{t_1} or T_{t_2} is long enough. Hence, from then on we can assume that $2d\delta > 1$.

By the bound on the diameter, the set S is completely contained in the lens $D(a, 2) \cap D(b, 2)$. We further subdivide the lens into a region that is close to $\{a, b\}$ and two regions that are far away from this edge. If there is a site c in one of the far away regions, we show in Lemma 6.6 that one of the stars T_a, T_b and T_c is long enough.

In the final case, all sites are close to $\{a, b\}$. Let $\beta \in (0, \frac{1}{2}]$ and consider the weighted average

$$\left(\frac{1}{2} - \beta\right) \cdot |T_a| + \beta \cdot |T_{a,b}| + \beta \cdot |T_{b,a}| + \left(\frac{1}{2} - \beta\right) \cdot |T_b|$$

of T_a, T_b, $T_{a,b}$ and $T_{b,a}$. We will show that it is at least $\delta \cdot |T_{\mathrm{OPT}}|$. This directly implies that the longest of these four trees is the desired δ-approximation. As our fixed tree T_{OPT} and these four trees all contain the edge $\{a, b\}$, we can again use the notation from Section 2.1 to focus on the length of single edges. We define

$$\operatorname{avg}(s, \beta) = \left(\frac{1}{2} - \beta\right) \cdot \ell_{T_a}(s) + \beta \cdot \ell_{T_{a,b}}(s) + \beta \cdot \ell_{T_{b,a}}(s) + \left(\frac{1}{2} - \beta\right) \cdot \ell_{T_b}(s).$$

This reduces the task to showing $\operatorname{avg}(s, \beta) \geq \delta \cdot \ell_{T_{\mathrm{OPT}}}(s)$. In order to get meaningful lower bounds on $\ell_{T_{\mathrm{OPT}}}(s)$, we define three special points. We show in Lemma 6.7 that it suffices to consider the distance between s and these special points to get an upper bound on $\ell_{T_{\mathrm{OPT}}}(s)$. In Lemma 6.8 we then show that one of the three points can actually be ignored. Lemma 6.9 gives a very useful lower bound on the weighted average $\operatorname{avg}(s, \beta)$. Using this bound, we find two sets of upper and lower bounds on the weighting factor β, which only depend on δ. One of these sets turns out to be tighter than the other, both for the upper and the lower bound. By setting the upper and lower bound equal and solving the resulting equation for δ in Lemma 6.12, the approximation factor δ follows. Next, we describe the details of the argument.

6.3.1 $|T_{\mathrm{OPT}}|$ is small or there are sites far away

First, we show that if the longest edge in T_{OPT} is relatively small, the best star rooted at one of the points realizing the diameter yields a good approximation ratio. This can be seen by the following slight generalization of a lemma by Alon et al. [ARS95].

Lemma 6.4. *Let S be a site set and the sites t_1 and t_2 realize its diameter. Suppose that $\|t_1 t_2\| = 2$ and that each edge of the optimal tree T_{OPT} has length at most $1/\delta$. Then* $\max\{|T_{t_1}|, |T_{t_2}|\} \geq \delta \cdot |T_{\mathrm{OPT}}|$.

Proof. By the triangle inequality, for any site $s \in S$ we have $\|st_1\| + \|st_2\| \geq \|t_1 t_2\|$. Hence

$$|T_{t_1}| + |T_{t_2}| = \sum_{s \in S} (\|st_1\| + \|st_2\|) \geq n \cdot \|t_1 t_2\| = 2n,$$

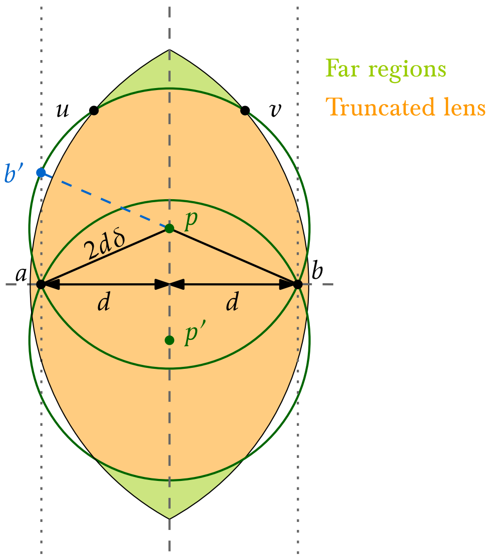


Figure 6.4: Decomposition of the lens into the far regions and the truncated lens.

implying that $\max\{|T_{t_1}|, |T_{t_2}|\} \geq n$. On the other hand, since each of the $n-1$ edges in T_{OPT} has length at most $1/\delta$, we get

$$\delta \cdot |T_{\mathrm{OPT}}| \leq \delta \cdot (n-1) \cdot \frac{1}{\delta} \leq n$$

and we are done. □

Let $\{a, b\}$ be the longest edge in T_{OPT} and let $\|ab\| = 2d \leq 2$. By Lemma 6.4 we can in the following assume that $2d\delta > 1$. We already established that S lies in the lens $D(a,2) \cap D(b,2)$ and that the sites a and b are in the interior of the lens. Without loss of generality, suppose that $a = (-d, 0)$ and $b = (d, 0)$. We split the lens into the *far region* and the *truncated lens*, as follows and depicted in Figure 6.4.

Let p be the point on the positive y-axis such that $\|pa\| = \|pb\| = 2d\delta$ and let p' be the point on the negative y-axis that is defined analogously. As $2d\delta > 1$ the circles $k = \partial D(p, \|pa\|)$ and $k' = \partial D(p', \|p'a\|)$ intersect the boundary of the lens. The *far regions* are defined as the regions in the lens above k and below k'. The remaining region of the lens is called the *truncated lens*. Let u and v be the two intersection points of k and the truncated lens with largest y-coordinate. We have the following lemma:

Lemma 6.5. *For $\delta \geq 0.5$, we have $a_x \leq u_x \leq 0 \leq v_x \leq b_x$.*

Proof. The statement follows directly from the definition. Let b' be the polar opposite point of b on the circle k. Then $x_{b'} = -x_b = x_a$. Now as $2 \cdot 2d\delta \geq 2d$ for all $\delta \geq 0.5$, the point b' lies outside of the lens or on its boundary. Since u is the higher of the intersection points of $D(b, 2d)$ and k, it lies to the right of b' and thus also to the right of a. By a symmetric argument, v lies to the right of b. □

We can now show that if there is a site c in one of the far regions, the tree T_{ALG} returned by `AlgSimple` is a δ-approximation.

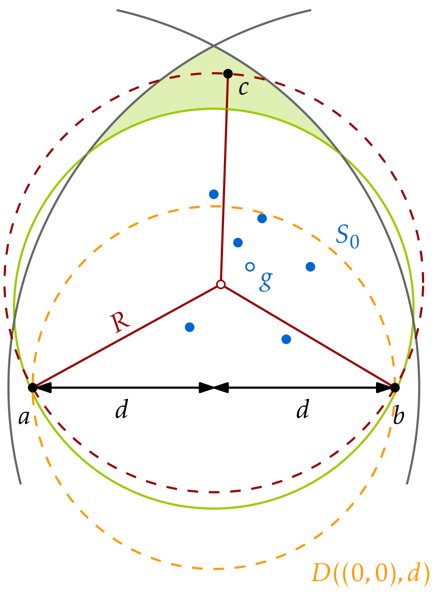


Figure 6.5: We can chose $v = a$. The orange disk does not intersect the far region above the x-axis. The circumradius of the triangle abc is marked red.

Lemma 6.6. *Let* $\{a, b\}$ *be the longest edge of* T_{OPT}. *If* S *contains a point* c *in one of the far regions, then* $\max\{|T_a|, |T_b|, |T_c|\} \geq \delta \cdot |T_{\text{OPT}}|$.

Proof. Assume without loss of generality that y lies in the far region above the x-axis. Let $S_0 = S \setminus \{a, b, c\}$ and let $g = \frac{1}{|S_0|} \sum_{s \in S_0} s$ be the centroid of this set, see Figure 6.5. Now we can show for all $q \in \mathbb{R}^2$:

$$\begin{aligned}\sum_{s\in S_0} \|qs\| = \sum_{s\in S_0} \|q - s\| &\geq \|\sum_{s\in S_0} (q - s)\| \\ &= \|(|S_0| \cdot q - \sum_{s\in S_0} s)\| \\ &= |S_0| \cdot \|q - g\| = |S_0| \cdot \|qg\|\end{aligned}$$

Consider the triangle $\triangle abc$. By Lemma 6.5, the angles at a and b are at most $\frac{\pi}{4}$. Furthermore, the part of the disk $D((0,0), d)$ above the x-axis is completely contained in the disk defined by k and c lies above this disk. Thus by Thale's theorem, the angle at c is also at most $\frac{\pi}{4}$. By these considerations, the triangle abc is acute-angled and its circumradius R satisfies $R \geq 2d\delta$. By the properties of acute-angled triangles, there exists a site $v \in \{a, b, c\}$ such that $\|vg\| \geq R$. Combining this with the inequality from above, gives

$$\sum_{s\in S_0} \|vs\| \geq |S_0| \cdot \|vg\| \geq (n-3) \cdot R.$$

Together with $\|va\| + \|vb\| + \|vc\| \geq 2R$ which holds for any acute-angled triangle, we get

$$|T_v| \geq (n-3)R + 2R = (n-1) \cdot R \geq (n-1) \cdot 2d\delta \geq \delta \cdot |T_{\text{OPT}}|,$$

where in the last inequality we used that each edge of T_{OPT} has length at most $2d$. □

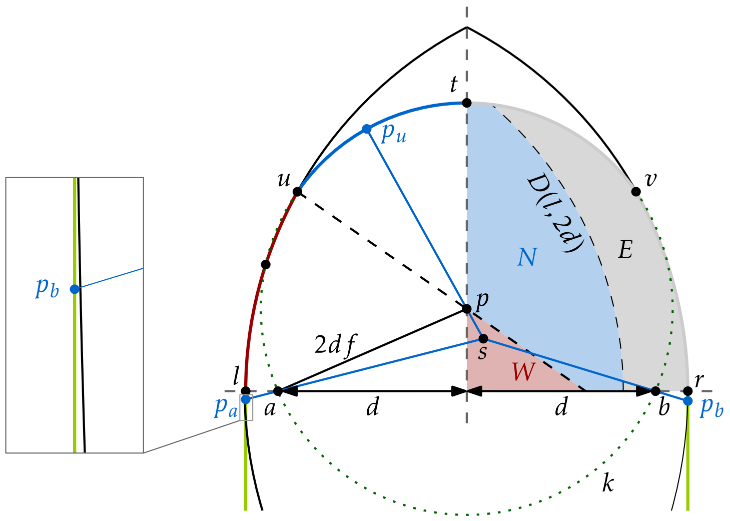


Figure 6.6: For a site $s \in E$ we have $\ell_{T_{\mathrm{OPT}}}(s) \le 2d$. If the site s lies in N we have $\ell_{T_{\mathrm{OPT}}}(s) \le \max\{\|sp_a\|, \|su\|\}$. We have $\ell_{T_{\mathrm{OPT}}}(s) \le \max\{\|sp_a\|, \|sp_u\|\}$ if $s \in W$.

6.3.2 All Sites Lie in the Truncated Lens

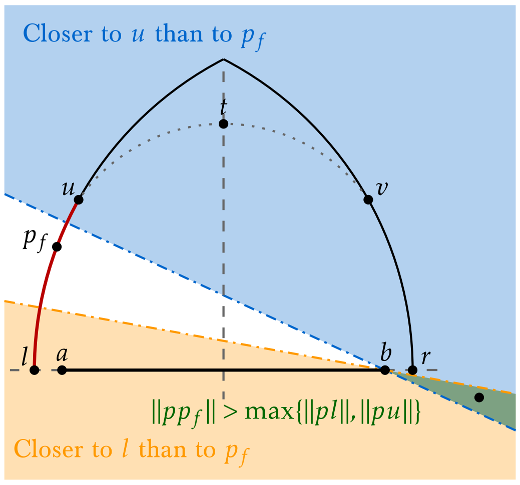
From now on we assume that no point lies in a far region and that the length of the longest edge in T_{OPT} is at least $1/\delta$. We recall the definition of the point p, which is the point on the positive y-axis with $\|pa\| = \|pb\| = 2d\delta$. The point p defined like this has coordinates $p = \left(0, \sqrt{(2d\delta)^2 - d^2}\right)$. Recall that the circle $k = \partial D(p, \|pa\|)$ always intersects the lens $D(a,2) \cap D(b,2)$. Also u and v are the intersection points of the lens with k with the largest y-coordinates, where u is to the left of v. We fix a site $s = (s_x, s_y)$ in the truncated lens and assume without loss of generality that $s_x, s_y \ge 0$. This is equivalent to saying that we only consider sites in the first quadrant of the coordinate system. We further subdivide the truncated lens into regions, see Figure 6.6 for an illustration. First of all we denote the region inside the truncated lens but outside of $D(l, 2d)$ by E. We further subdivide the remainder of the truncated lens into the part N above the line defined by u and p and the part W below that line. Based on the position of s we now define three special points, again refer to Figure 6.6 for an illustration.

(a) p_a is the point on the ray $\overrightarrow{sa}$ whose x-coordinate equals $-(2-d)$;

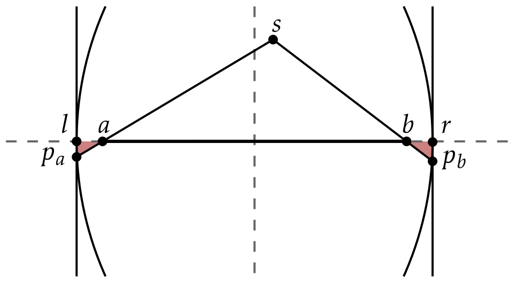
(b) p_b is the point $\overrightarrow{sb}$ whose x-coordinate equals $2-d$, if $s_x < d$, and $p_b = b$ if $s_x \ge d$.

(c) p_u is the the point on the arc of k from u to v that is furthest from s. If $s \in W$, the ray $\overrightarrow{sp}$ intersects k on the arc from u to v. By triangle inequality this point is the point on k with the largest distance to s and we pick it as p_u. Otherwise, the furthest point on k lies below u and we set $p_u = u$.

Having these definitions at hand, we can now show that we can bound $\ell_{T_{\mathrm{OPT}}}(s)$ by only considering the three points p_a, p_b and p_u.


Closer to u than to p_f
t
u
v
p_f
l
a
b
r
$\|pp_f\| > \max\{\|pl\|, \|pu\|\}$
Closer to l than to p_f

(a) The point p_f lies on the arc ul. The points with $\|sp_f\| > \max\{\|sl\|, \|su\|\}$ form a convex wedge with apex b fully contained in the fourth quadrant.



(b) The point p_f lies in the third of fourth quadrant. As $\{s, p_f\}$ cannot be blocked by a or b, p_f lies in one of the red triangles.

Figure 6.7: Cases 2(c), 3 and 4 in Lemma 6.7.

Lemma 6.7. *For each site $s = (s_x, s_y)$ in the truncated lens with $s_x, s_y \geq 0$ we have*

$$\ell_{T_{\mathrm{OPT}}}(s) \leq \min\{2d, \max\{\|sp_a\|, \|sp_b\|, \|sp_u\|\}\}.$$

Proof. Let l and r be the left- and rightmost points of the lens with $r = (2-d, 0)$ and $l = (d-2, 0)$. Assume that s lies in E. Then $\|sp_a\| \geq \|sl\| \geq 2d$, and the right-hand side equals $2d$. Since the left-hand side is at most $2d$ by assumption, the claim is true in this case.

So from now on we can assume that $s \in N \cup W$. This directly implies that $p_b \neq b$ and $(p_b)_x = 2-d$. Let p_f be the point within the truncated lens furthest from s such that the line segment $\overline{p_f s}$ does not intersect the edge $\{a, b\}$. Clearly, p_f lies on the boundary of the truncated lens. Since $\ell_{T_{\mathrm{OPT}}}(s) \leq 2d$ by assumption, it suffices to show that $\|sp_f\| \leq \max\{\|sp_a\|, \|sp_b\|, \|sp_u\|\}$. Let t be the top-most point of the truncated lens. We distinguish four cases, based on the position of p_f relative to the origin:

Case 1: p_f lies in the first quadrant Consider the reflection p'_f of p_f along the y-axis. Since $s_x \geq 0$, we have $\|sp'_f\| \geq \|sp_f\|$, and the inequality is strict when $s_x > 0$. Thus, when $s_x > 0$, this case cannot occur, and when $s_x = 0$, it reduces to the case where p_f belongs to the second quadrant.

Case 2: p_f lies in the second quadrant This case is split into three subcases.

(a) p_f lies on the arc tu and $s \in N$ By definition, we have $p_u = u$, which implies that there is no point on the arc tu with larger distance to s than u. In particular, this means that $\|sp_f\| \leq \|sp_u\|$ which is sufficient to show the claim.

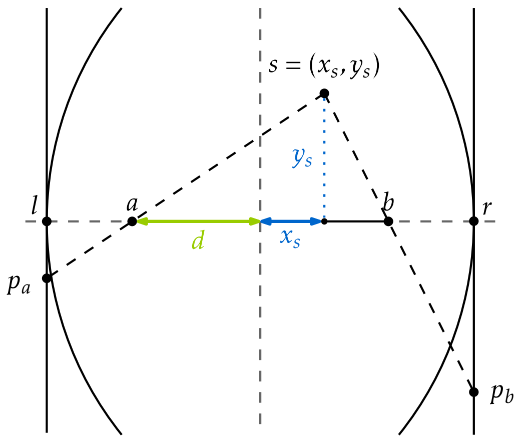


Figure 6.8: The situation for Lemma 6.8 (not to scale).

(b) p_f lies on the arc tu and $s \in W$ By definition, if $s \in W$, the point p_u is the furthest point from s of the arc tu and thus $\|sp_f\| = \|sp_u\|$ and we are done.

(c) p_f lies on the arc ul We claim that $\|sp_f\| \leq \max\{\|sl\|, \|su\|\}$. Indeed, the perpendicular bisectors of segments $\overline{p_f l}$ and $\overline{p_f u}$ intersect at b and thus the points for which the claim fails all lie in a convex cone with apex b that is fully contained in the fourth quadrant, see Figure 6.7(a). Since $\|sl\| \leq \|sp_a\|$ and $\|su\| \leq \|sp_u\|$, we get $\|sp_f\| \leq \max\{\|sp_a\|, \|sp_u\|\}$.

Case 3: p_f lies in the third quadrant Then p_f lies in the triangle slp_a which is shown in Figure 6.7(b). Thus $\|sp_f\| \leq \|sp_a\|$.

Case 4: p_f lies in the fourth quadrant Then p_f lies in the triangle srp_b and $\|sp_f\| \leq \|sp_b\|$, again see Figure 6.7(b). □

The next lemma shows that we can disregard $\|sp_b\|$ in our considerations, as it will always be bounded from above by $\|sp_a\|$.

Lemma 6.8. *Let $s = (s_x, s_y)$ be any site in the truncated lens with $s_x, s_y \geq 0$. If $\|sp_a\| \leq 2d$, then $\|sp_b\| \leq \|sp_a\|$.*

Proof. We present a straightforward algebraic proof, as we are not aware of a purely geometric proof. Even though we show most of the algebraic manipulation steps, one could also verify the claim by using a computer algebra system.

The case $s_x = 0$ is trivial as it yields $\|pp_b\| = \|pp_a\|$. Thus, we only consider the case $s_x > 0$. Since $\|sp_a\| \leq 2d$, we must have $s_x < d$, and therefore $p_b \neq b$ has x-coordinate $(p_b)_x = 2 - d$, see Figure 6.8.

From similar triangles and the Pythagorean theorem we get $\|sp_a\| = \|sa\| \cdot \frac{2-d+s_x}{d+s_x}$ and $\|sa\|^2 = (d+s_x)^2 + s_y^2$. Therefore, the assumption $\|sp_a\|^2 \leq (2d)^2$ can be equivalently rewritten as:

$$\begin{aligned}\left(s_y^2 + (d+s_x)^2\right) \cdot \frac{(2-d+s_x)^2}{(d+s_x)^2} &\leq 4d^2 \\ s_y^2 &\leq \frac{\left(4d^2 - (2-d+s_x)^2\right)(d+s_x)^2}{(2-d+s_x)^2}. \qquad (6.2)\end{aligned}$$

Similarly, as with $\|sp_a\|^2$, we express $\|sp_b\|^2$ as

$$\|sp_b\|^2 = \left(s_y^2 + (d-s_x)^2\right) \cdot \frac{(2-d-s_x)^2}{(d+s_x)^2} \qquad (6.3)$$

and rewrite our goal $\|sp_b\|^2 \leq \|sp_a\|^2$ as

$$\begin{aligned}\left(s_y^2 + (d-s_x)^2\right) \cdot \frac{(2-d-s_x)^2}{(d-s_x)^2} &\leq \left(s_y^2 + (d+s_x)^2\right) \cdot \frac{(2-d+s_x)^2}{(d+s_x)^2} \\ s_y^2 \cdot \frac{(2-d-s_x)^2(d+s_x)^2 - (2-d+s_x)^2(d-s_x)^2}{(d+s_x)^2(d-s_x)^2} &\leq (2-d+s_x)^2 - (2-d-s_x)^2,\end{aligned}$$

which, upon expanding the parentheses and dividing by $4s_x > 0$ transforms into the goal

$$s_y^2 \cdot \frac{2(1-d)(2d-d^2-s_x^2)}{(d+s_x)^2(d-s_x)^2} \leq 2-d.$$

We plug in the upper bound on s_y^2 from (6.2), cancel the term $(d+s_x)^2$, and clear the denominators. This leaves us with proving

$$\left(4d^2 - (2-d+s_x)^2\right) \cdot 2\left(1-d\right)\left(2d-d^2-s_x^2\right) \leq (2-d+s_x)^2 \cdot (2-d)(d-s_x)^2,$$

which, upon expanding the parentheses, reduces to

$$0 \leq \left(16d - 32d^2 + 8d^3 + 16d^4 - 7d^5\right) + s_x^2\left(-8d + 16d^2 - 10d^3\right) + ds_x^4. \qquad (6.4)$$

For any fixed $d > 0$, the right-hand side $Q(d, s_x^2)$ is a quadratic function of s_x^2 with positive coefficient $d > 0$ by the leading term $(s_x^2)^2$. Hence, the minimum of $Q(d, s_x^2)$ is attained when

$$s_x^2 = \frac{8d - 16d^2 + 10d^3}{2d} = 4 - 8d + 5d^2.$$

Plugging $s_x^2 = 4 - 8d + 5d^2$ into (6.4) and expanding the parentheses for the one last time we are left to prove

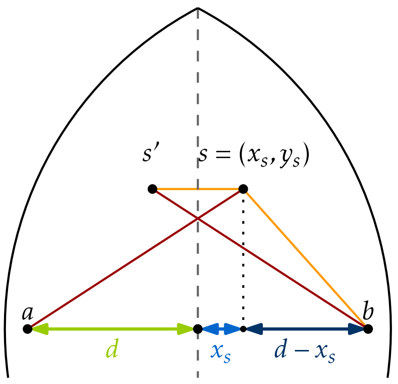


Figure 6.9: Mirroring s along the y-axis in Lemma 6.9. By triangle inequality $\|ss'\|+\|sb\|\le\|s'b\|=\|sa\|$.

$$\begin{aligned}0 &\le 32d^2-96d^3+96d^4-32d^5\\ 0 &\le 32d^2(1-d)^3,\end{aligned}$$

which is true since $d\le 1$. □

With Lemmas 6.7 and 6.8 we have reduced the task of showing $\operatorname{avg}(s,\beta)\le\delta\cdot\ell_{T_{\mathrm{OPT}}}(s)$ to that of showing $\operatorname{avg}(s,\beta)\le\delta\cdot\min\{2d,\max\{\|sp_a\|,\|sp_u\|\}$. We aim to find two sets of upper and lower bounds on β for which $\operatorname{avg}(s,\beta)\le\|sp_a\|$ and $\operatorname{avg}(s,\beta)\le\|sp_u\|$, respectively, hold and which only depend on δ. However, first we give a general lower bound on $\operatorname{avg}(s,\beta)$ which will be useful for the more specific bounds.

Lemma 6.9. *Let $s=(s_x,s_y)$ be any site from our set with $s_x,s_y\ge 0$ and let $\beta\in(0,\frac{1}{2}]$ be a real number. Then*

$$\operatorname{avg}(s,\beta)\ge\frac{d\cdot(1-\beta)+s_x\cdot 2\beta}{d+s_x}\cdot\|sa\|.$$

Proof. Similar to the proof of Theorem 6.2, we have $\ell_{T_a}(s)=\ell_{T_{a,b}}(s)=\|as\|$, $\ell_{T_b}(s)=\|sb\|$, and $\ell_{T_{b,a}}\ge s_x$. Unpacking the definition of $\operatorname{avg}(s,\beta)$, this yields

$$\begin{aligned}\operatorname{avg}(s,\beta)&=\left(\frac{1}{2}-\beta\right)\cdot\ell_{T_a}(s)+\beta\cdot\ell_{T_{a,b}}(s)+\beta\cdot\ell_{T_{b,a}}(s)+\left(\frac{1}{2}-\beta\right)\cdot\ell_{T_b}(s)\\ &\ge\left(\frac{1}{2}-\beta\right)\cdot\|sa\|\quad+\beta\cdot\|sa\|\quad+\beta\cdot s_x\qquad+\left(\frac{1}{2}-\beta\right)\cdot\|sb\|.\end{aligned}$$

Let $s'=(-s_x,s_y)$ be the reflection of s along the y-axis, see Figure 6.9. The triangle inequality $\|s's\|+\|sb\|\ge\|s'b\|=\|sa\|$ leads to $\beta\cdot s_x=\frac{1}{2}\beta\cdot\|s's\|\ge\frac{1}{2}\beta\cdot(\|sa\|-\|sb\|)$, and we obtain

$$\begin{aligned}\operatorname{avg}(s,\beta)&\ge(1/2-\beta)\cdot\|sa\|+\beta\cdot\|sa\|+\frac{1}{2}\beta\cdot(\|sa\|-\|sb\|)+(1/2-\beta)\cdot\|sb\|\\ &\ge\frac{(1+\beta)\cdot\|sa\|+(1-3\beta)\cdot\|sb\|}{2}.\end{aligned}$$

Next, we claim that $||sb|| \geq \frac{d-s_x}{d+s_x} \cdot ||sa||$. The claim can be seen as follows: upon squaring, using the Pythagorean theorem and clearing the denominators by the following calculations, this becomes $s_y^2 \cdot \frac{4ds_x}{(d+s_x)^2} \geq 0$, which is true.

$$\begin{aligned}
||sb||^2 &\geq \frac{(d-s_x)^2}{(d+s_x)^2} \cdot ||sa||^2 \\
(d-s_x)^2 + s_y^2 &\geq \frac{(d-s_x)^2}{(d+s_x)^2} \cdot ((d+s_x)^2 + s_y^2) \\
s_y^2 \cdot \left(1 - \frac{(d-s_x)^2}{(d+s_x)^2}\right) &\geq 0 \\
s_y^2 \cdot \frac{4ds_x}{(d+s_x)^2} &\geq 0
\end{aligned}$$

Using this bound on the term containing $||sb||$, we finally get the desired

$$\begin{aligned}
\operatorname{avg}(s,\beta) &\geq \frac{(1+\beta)(d+s_x) + (1-3\beta)(d-s_x)}{2(d+s_x)} \cdot ||sa|| \\
&= d \cdot \frac{(1-\beta) + 2\beta \cdot s_x}{d+s_x} \cdot ||sa||.
\end{aligned}$$

□

Now we are equipped to show the two sets of bounds whose combination will give us a good lower bound on the approximation factor δ.

Lemma 6.10. *Let $s = (s_x, s_y)$ be a site in the lens with $s_x, s_y \geq 0$. Suppose $\delta \leq \frac{5}{8}$ and*

$$\frac{2\delta - 1}{5 - 8\delta} \leq \beta \leq \frac{1}{2} \cdot \delta.$$

Then $\operatorname{avg}(s,\beta) \geq \delta \cdot \min\{2d, ||sp_a||\}$.

Proof. We distinguish the cases $s_x \geq 3d - 2$ and $s_x \leq 3d - 2$. In the first case we show that $\operatorname{avg}(s,\beta) \geq \delta \cdot 2d$, while in the second case we show $\operatorname{avg}(s,\beta) \geq \delta \cdot ||sp_a||$.

Case 1: $s_x \geq 3d - 2$ Using Lemma 6.9 and the inequalities $||sa|| \geq d + s_x$ and $s_x \geq 3d - 2$, we rewrite

$$\begin{aligned}
\operatorname{avg}(s,\beta) &\geq \frac{d \cdot (1-\beta) + s_x \cdot 2\beta}{d + s_x} \cdot ||sa|| \\
&\geq d - d\beta + (3d-2) \cdot 2\beta \\
&= \beta(5d-4) + d.
\end{aligned}$$

Hence, it suffices to prove $\beta \cdot (5d-4) \geq d(2\delta - 1)$. Using the lower bound on β and $4 \leq 8d\delta$, we get

$$\beta \cdot (5d-4) \geq \beta \cdot (5d - 8d\delta) \geq \frac{2\delta-1}{5-8\delta} \cdot d \cdot (5 - 8\delta) = d(2\delta - 1)$$

as desired. Note that $2\delta - 1$ and $5 - 8\delta$ are both positive.

Case 2: $s_x \le 3d - 2$ We have $\|sp_a\| = \|sa\| \cdot \frac{2-d+s_x}{d+s_x}$. Using Lemma 6.9, it suffices to prove

$$\frac{d \cdot (1-\beta) + s_x \cdot 2\beta}{d + s_x} \cdot \|sa\| \ge \delta \cdot \|sa\| \frac{2 - d + s_x}{d + s_x}$$
$$d(1-\beta) + s_x \cdot 2\beta \ge \delta(2 - d + s_x)$$
$$d(1-\beta) - \delta(2-d) \ge s_x(\delta - 2\beta).$$

Since $\delta \ge 2\beta$ by assumption, the right-hand side is increasing in s_x and we can plug in $3d - 2$ for s_x. This leaves us with proving the inequality

$$d(1-\beta) - \delta(2-d) \ge (3d-2)(\delta - 2\beta)$$
$$\beta \cdot (5d - 4) \ge d(2\delta - 1),$$

which is the same inequality as in the first case. □

Lemma 6.11. *Let $s = (s_x, s_y)$ be a site in the truncated lens with $s_x, s_y \ge 0$. Suppose that $\beta < \frac{151}{304} \cdot \delta$, that $\frac{1}{2} \le \delta \le \frac{19}{32}$ and that*

$$\frac{2\delta - 1}{2\sqrt{5 - 8\delta} - 1} \le \beta \le 1 - \delta\sqrt{4\delta^2 - 1} - 2\delta^2. \tag{6.5}$$

Then $\operatorname{avg}(s, \beta) \ge \delta \cdot \min\{2d, \|sp_u\|\}$.

Proof. Because of the lower bound for $\operatorname{avg}(s, \beta)$ in Lemma 6.9, to show the statement it suffices to show that

$$\frac{d \cdot (1-\beta) + s_x \cdot 2\beta}{d + s_x} \cdot \|sa\| \ge \delta \cdot \min\{2d, \|sp_u\|\}. \tag{6.6}$$

We denote by $(p_u)_y$ the y-coordinate of p_u and we consider the two cases $s_y \le (p_u)_y$ and $s_y > (p_u)_y$ separately.

Case 1: $s_y \le (p_u)_y$ Define $\lambda = \frac{d \cdot (1-\beta) + s_x \cdot 2\beta}{d + s_x} \cdot \|sa\|$. To show the claim, we have to show that $\lambda \ge \delta \cdot \min\{2d, \|sp_u\|\}$. Note that λ is an increasing function in s_y, because $\|sa\|$ is also increasing in s_y. On the other hand, $\|sp_u\|$ is a decreasing function in s_y for $s_y \le (p_u)_y$: if $p_u \ne u$, then $s \in W$. This also implies that $s_y \le p_y$ and $\|sp_u\| = \|ps\| + 2d\delta$ is decreasing in s_y. If $p_u = u$ then $\|sp_u\| = \|su\|$ decreases when s_y increases for $s_y \le (p_u)_y$.

As $\min\{2d, \|sp_u\|\}$ is decreasing and λ is increasing in s_y for $s_y \le (p_u)_y$, it suffices to show (6.6) for $s_y = 0$ to handle our current case . Now we have $\|sa\| = d + s_x$, so we can rewrite λ as $\lambda_0 = d \cdot (1-\beta) + s_x \cdot 2\beta$, which is positive.

Let $q = (q_x, 0)$, $q_x \ge 0$ be the point on the positive x-axis such that $\|qp\| = 2d(1-\delta)$. This means that for $s = q$ we have $\|sp_u\| \le \|sp\| + \|pp_u\| = 2d$, see Figure 6.10. We further distinguish two subcases, depending on whether $0 \le s_x \le q_x$ or $s_x > q_x$.

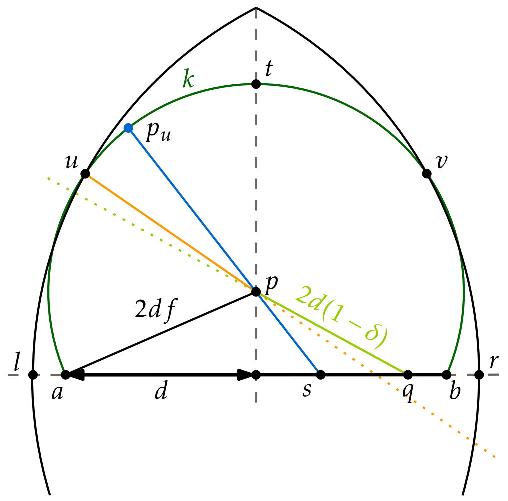


Figure 6.10: The situation for the first case of Lemma 6.11.

(a) $0 \le s_x \le q_x$ We will show that in this case $\lambda_0 \ge \delta \cdot \|sp_u\|$. The Pythagorean theorem gives

$$\delta \cdot \|sp_u\| \le \delta \cdot \|sp\| + \delta \cdot \|pp_u\| = \delta\sqrt{s_x^2 + (2d\delta)^2 - d^2} + 2d\delta^2. \tag{6.7}$$

Therefore, substituting λ_0, it suffices to show

$$d \cdot (1-\beta) + s_x \cdot 2\beta \ge \delta\sqrt{s_x^2 + (2d\delta)^2 - d^2} + 2d\delta^2. \tag{6.8}$$

When $\beta < \frac{151}{304} \cdot \delta$ and $\delta \le \frac{19}{32}$, the term $d \cdot (1-\beta) - 2d\delta^2$ is positive and (6.8) is equivalent to

$$0 \ge \delta^2 \cdot \left(s_x^2 + (2d\delta)^2 - d^2\right) - \left(d \cdot (1-\beta) + s_x \cdot 2\beta - 2d\delta^2\right)^2.$$

The right-hand side is a quadratic function in s_x whose coefficient of the leading term is $\delta^2 - 4\beta^2$, which is positive. Hence to show that the right hand side is smaller than zero in the interval $0 \le s_x \le x_p$, it suffices to check the inequality (6.8) for $s_x \in \{0, q_x\}$.

For $s_x = 0$, we need to check that $d(1-\beta) \ge \delta\left(\sqrt{d^2(4\delta^2 - 1)} + 2d\delta\right)$, which reduces precisely to the assumption

$$\beta \le 1 - \delta\sqrt{4\delta^2 - 1} - 2\delta^2.$$

For $s_x = q_x$, the Pythagorean theorem gives

$$\begin{aligned}(q_x)^2 &= (2d(1-\delta))^2 + (x_p)^2 \\ &= (2d(1-\delta))^2 + d^2 - (2d\delta)^2 \\ &= d^2 \cdot (5 - 8\delta),\end{aligned}$$

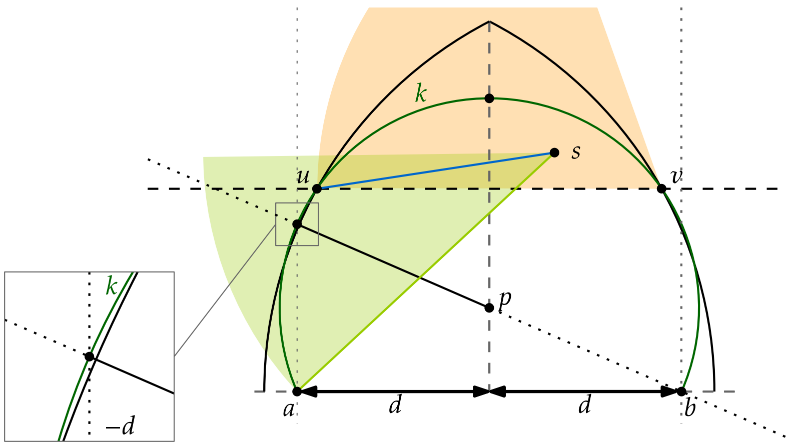


Figure 6.11: The situation for case 2. We have $\|su\| \le 2d$ and $\|su\| \le \|sa\|$.

hence $q_x = d\sqrt{5-8\delta}$.

The point q has been selected such that $\|sp\| + \|pp_u\| = 2d$, and therefore the middle expression of (6.7) and also the right hand side of (6.8) are $2d\delta$. We thus have to verify that

$$\begin{aligned} d(1-\beta) + d\sqrt{5-8\delta}\cdot 2\beta &\ge \delta \cdot 2d \qquad (6.9)\\ \beta\cdot\left(2\sqrt{5-8\delta}-1\right) &\ge 2\delta - 1. \end{aligned}$$

Since $\delta \le 19/32$ by the assumptions of the lemma, the term in the parentheses on the left-hand side is positive, and after dividing we obtain precisely the assumption.

(b) $s_x > q_x$ In this case we show that $\lambda_0 \ge \delta \cdot 2d$. Since the term λ_0 is increasing in s_x and $2d$ is constant, we only need to show that $\lambda_0 \ge \delta \cdot 2d$ for $s_x = q_x$. However, this was already shown in the previous case; see (6.9).

Case 2: $s_y > (p_u)_y$: See Figure 6.11 for a sketch of the situation in this case. As the part above u is completely contained in $N \cup E$, we have $\|sp_u\| = \|su\| \le \|uv\|$. Furthermore, by Lemma 6.5, we have $u_x \ge -d$ as well as $s_x \le d$. So we have $\min\{2d, \|sp_u\|\} = \|sp_u\|$. This means that to show (6.6) we have to show

$$\frac{d(1-\beta) + 2x\beta}{d + s_x} \cdot \|sa\| \ge \delta \cdot \|sp_u\|.$$

As s lies above the horizontal line through u, we get $\|sa\| \geq \|sp_u\|$, thus it suffices to show

$$\begin{aligned}
\frac{d(1-\beta)+2x\beta}{d+s_x} &\geq \delta \\
d - d\beta + 2s_x\beta &\geq \delta d + \delta s_x \\
d(1-(\beta+\delta)) &\geq s_x(\delta - 2\beta) \\
s_x &\leq d \cdot \frac{1-(\beta+\delta)}{\delta-2\beta}.
\end{aligned}$$

As we know that $s_x \leq d$ this is true for

$$\begin{aligned}
\frac{1-(\beta+\delta)}{\delta-2\beta} &\geq 1 \\
1-(\beta+\delta) &\geq \delta - 2\beta \\
\beta &\geq 2\delta - 1,
\end{aligned}$$

where in the last step we used that $\delta > 2\beta$. The last condition is a looser bound than the left hand side of (6.5) as $\delta \geq \frac{1}{2}$ and therefore $2 \cdot \sqrt{5-8\delta} - 1 \leq 1$. □

As a close examination shows that the bounds of Lemma 6.11 are stricter than the ones of Lemma 6.10, we now want to chose δ in such a way that the left and the right side of Lemma 6.11 are equal. The choice of δ following from this is justified in the following lemma.

Lemma 6.12. *The positive solutions of*

$$\frac{2x-1}{2\sqrt{5-8x}-1} = 1 - x\sqrt{4x^2-1} - 2x^2$$

are $x = \frac{5}{8}$ and the fourth smallest root of

$$-80 + 128x + 504x^2 - 768x^3 - 845x^4 + 1096x^5 + 256x^6.$$

Proof. We provide a sketch of how to solve it "by hand". One can also use advanced software for algebraic manipulation. Setting the polynomials $q_1(x) = 4x^2 - 1$ and $q_2(x) = 5 - 8x$, and multiplying both sides of the equation by the denominator on the left-hand side, we are left with the equation

$$\begin{aligned}
2x - 2x^2 &= x\sqrt{q_1(x)} + (2-4x^2)\sqrt{q_2(x)} - 2x\sqrt{q_1(x)q_2(x)} \\
2x - 2x^2 + 2x\sqrt{q_1(x)q_2(x)} &= x\sqrt{q_1(x)} + (2-4x^2)\sqrt{q_2(x)}.
\end{aligned}$$

Squaring both sides, which may introduce additional roots, we get the following equation for some polynomials $q_3(\cdot), \ldots, q_6(\cdot)$:

$$q_3(x) + q_4(x)\sqrt{q_1(x)q_2(x)} = q_5(x) + q_6(x)\sqrt{q_1(x)q_2(x)}$$
$$q_3(x) - q_5(x) = \big(q_6(x) - q_4(x)\big)\sqrt{q_1(x)q_2(x)}.$$

Squaring both sides, which may again introduce additional solutions, we get the polynomial

$$8\big(x - \tfrac{5}{8}\big)\big(-80 + 128x + 504x^2 - 768x^3 - 845x^4 + 1096x^5 + 256x^6\big) = 0$$

This polynomial has 7 real roots that can be approximated numerically. The smallest three roots of this polynomial are negative ($x \doteq -4.82037$, $x \doteq -0.657898$ and $x \doteq -0.523446$). The fourth smallest root, $x \doteq 0.546723$, is a solution to the original equation. The fifth and sixth roots are $x \doteq 0.577526$ and $x \doteq 0.596211$, which are not solutions to the original equation. The largest root of the polynomial is $x = \frac{5}{8}$, which is also a solution to the original equation. □

Combining the line of argumentation and lemmas above, we get the main result of this section.

Theorem 6.13. *For any set of sites S in general position (no three sites collinear), we can compute in polynomial time a plane tree of Euclidean length at least $\delta \cdot |T_{\mathrm{OPT}}|$, where $|T_{\mathrm{OPT}}|$ denotes the length of a longest plane tree on S and $\delta > 0.5467$ is the fourth smallest real root of the polynomial*

$$P(x) = -80 + 128x + 504x^2 - 768x^3 - 845x^4 + 1096x^5 + 256x^6.$$

6.4 Using Small Diameters for Approximation

The algorithms for approximating $|T_{\mathrm{OPT}}|$ often produce trees with small diameter. In this section, we consider upper bounds on the approximation factor achieved by the longest plane tree among those of diameter at most three. We show the upper bound, by focusing on convex point sets.

Given an integer $d \geq 2$ and a point set S, let $T^d_{\mathrm{OPT}}(S)$ be a longest plane tree spanning S among those whose diameter is at most d. One can then ask: *What is the approximation ratio $\frac{|T^d_{\mathrm{OPT}}(S)|}{|T_{\mathrm{OPT}}(S)|}$ achieved by such a tree?* As before, we drop the dependency on S in the notation and just use T^d_{OPT} and T_{OPT}.

When $d = 2$, this reduces to asking about the performance of stars. A result by Alon et al. [ARS95, Theorem 4.1] can be restated as $\frac{|T^2_{\mathrm{OPT}}|}{|T_{\mathrm{OPT}}|} = \frac{1}{2}$. Below we show a specific upper bound tailored to the case $d = 3$. This case is especially interesting, as in Theorem 7.5 we will show that such trees can be computed efficiently.

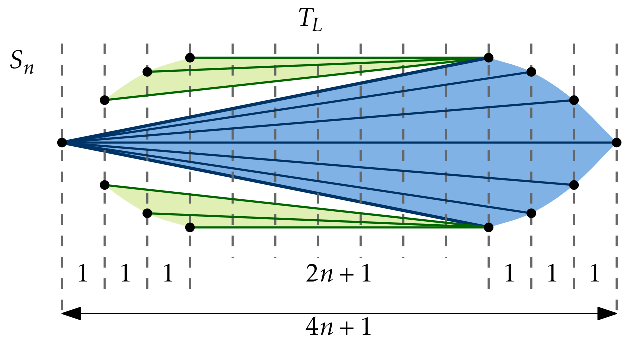


Figure 6.12: An illustration of the point set S_n when $n = 3$ with a tree T_L.

Theorem 6.14. *For every $\varepsilon > 0$ there exists a convex point set such that each longest plane tree of diameter three is at most $(5/6) + \varepsilon$ times as long as each longest plane tree of arbitrary diameter.*

Proof. Let S_n be a flat site set with $|S_n| = 4n + 2$, consisting of two flat convex point sets that are symmetric with respect to a horizontal line. As the point set is flat, it suffices to specify the x-coordinates of the sites in both convex sets. We set the x-coordinates in such a way that we have the following sequence of distances between two consecutive sites:

$$(\underbrace{1,\ldots,1}_{n\times},\ 2n+1,\ \underbrace{1,\ldots,1}_{n\times}).$$

In other words, S_n consists of two diametrically opposite sites, four unit-spaced arcs of n sites each, and a large gap of length $2n + 1$ in the middle, see Figure 6.12.

Let T_L be the tree depicted in Figure 6.12. On the one hand, straightforward counting, detailed below, will give $|T_{\mathrm{OPT}}| \geq |T_L| = 12n^2 + 6n + 1$. On the other hand, we claim that any tree T on S_{4n+2} of diameter at most 3 has length at most $10n^2 + 6n + 1$. Thus

$$\frac{|T^3_{\mathrm{OPT}}|}{|T_{\mathrm{OPT}}|} \leq \frac{10n^2 + 6n + 1}{12n^2 + 6n + 1},$$

which is implies $\lim_{n\to\infty} \frac{|T^3_{\mathrm{OPT}}|}{|T_{\mathrm{OPT}}|} \leq \frac{5}{6}$.

First, consider the tree T_L. We divide the edges into different parts, as shaded in Figure 6.12. We have

$$\begin{aligned}|T_L| &= 2\sum_{i=0}^{n-1}(3n+1+i) + 4n + 1 + 2\sum_{i=0}^{n-1}(2n+1+i)\\ &= 12n^2 + 6 + n,\end{aligned}$$

where the first sum and the $4n + 1$ term come from the edges that are shaded blue in Figure 6.12 and the last sum from the edges shaded green.

In the rest of this proof we will show that any longest tree T among those of diameter at most 3 on S_n has length at most $10n^2 + 6n + 1$. First, note that as T has diameter at

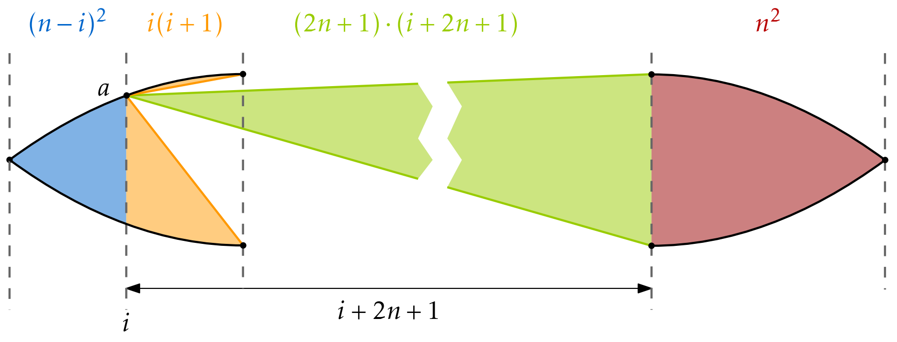


Figure 6.13: The longest star for S_n is short.

most 3 it is either a star or it has a cut edge $\{a, b\}$ whose removal decomposes T into a star rooted at a and a star rooted at b.

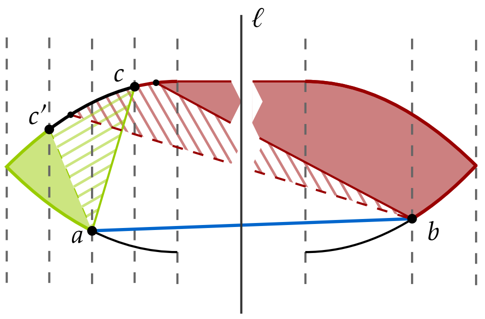
To sum up the lengths of the edges of the tree, it is often easier to split the edges into parts: one within the left of the gap, one within the right side of the large gap, and one crossing the gap. The edges in the parts on one side of the gap have maximum length n, and the edges going across the large gap, have a length between $2n+1$ and $2n+1+n = 3n+1$.

If T is a star, then without loss of generality we can assume that its root a is on the left of the large gap. Denote the distance from the large gap to a by $0 \leq i \leq n$. Straightforward algebra, visualized in Figure 6.13, then bounds of the lengths of union of these edges. All bounds within their respective sides again follow by using the nth triangular number, similar to the bound for $|T_L|$. Combining these bounds we get:

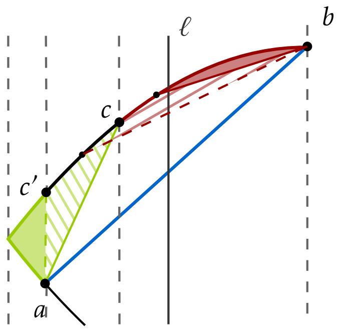
$$\begin{aligned}|T_a| &= (n-i)^2 + i(i+1) + (2n+1)\cdot(i+2n+1) + n^2\\ &\leq n(n+1) + (2n+1)(3n+1) + n^2\\ &= 8n^2 + 6n + 1.\end{aligned}$$

Now suppose T has diameter three and denote the line segments defined by its cut edge by $\{a, b\}$. If $\{a, b\}$ is vertical then $|T| = |T_a|$ and the above bound applies. In the remaining cases, we can assume without loss of generality that a is to the left of b. The line supporting $\overline{ab}$ then splits the remaining points of S_n into two arcs – one above the line and the other one below it. As a shorthand we will say that the points above or below this line are above or below the line segment $\overline{ab}$.

We claim that in the longest tree of diameter three, the points of one arc are either all connected to a or all to b. For the sake of contradiction, fix an arc and suppose c is the rightmost point connected to a. Then by the choice of c, everything right of it is connected to b and by convexity everything to the left of c is connected to a. Let ℓ be the vertical line through the midpoint of $\overline{ab}$, see Figure 6.14 for an illustration. If c lies to the left of ℓ, moving c to the left increases the length of the tree as the distance to b for all points left of ℓ is larger than the distance to a. If otherwise c lies to the right of ℓ, moving it to the right again increases the length of the tree. Thus we either have $c = a$ or $c = b$ as claimed.



(a) The line ℓ lies in the large gap.



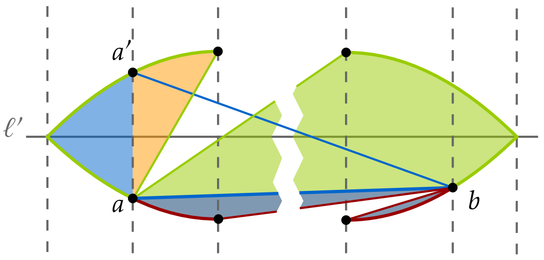
(b) The line ℓ lies on one side of the large gap.

Figure 6.14: The sites on one side of $\overline{ab}$ are either all connected to a or all to b.

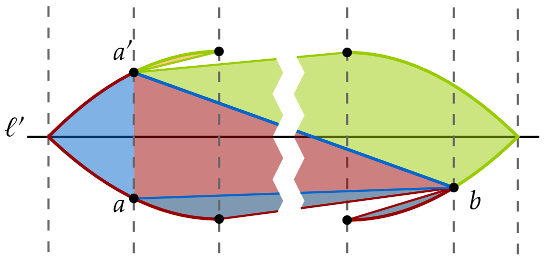
Since S_n is centrally symmetric and we have already dealt with the case where T is a star, we can assume without loss of generality that all points above $\overline{ab}$ are connected to a and all points below $\overline{ab}$ are connected to b. Now suppose that a is below the line ℓ' through the sites defining the diameter of S_n and consider the reflection $a' \in S_n$ of a about ℓ', see Figure 6.15(a). Then the tree T' with cut edge $\{a', b\}$ is longer than T, since in T' the points to the left of $\overline{aa'}$ are connected to b rather than to a and all the other edges have the same length, see Figure 6.15(b). Hence, we can assume that a lies above or on ℓ'. Similarly, b lies below ℓ' or on it.

It remains to distinguish two cases based on whether a and b lie on different sides of the large gap or not. Either way, denote the distance from the gap to a and b by i and j, respectively. In the first case, considering the subdivision of the edges as sketched in Figure 6.16(a), straightforward algebra gives:

$$\begin{aligned}|T| &= n^2 + (2n - j)(i + 2n + 1) + (i + 2n + 1 + j) + (2n - i)(2n + 1 + j) + n^2\\ &= 10n^2 + 6n + 1 - 2ij\\ &\leq 10n^2 + 6n + 1,\end{aligned}$$

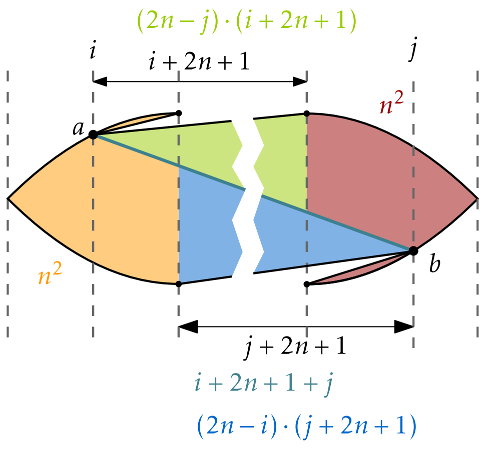


(a) a and b are both below ℓ'.

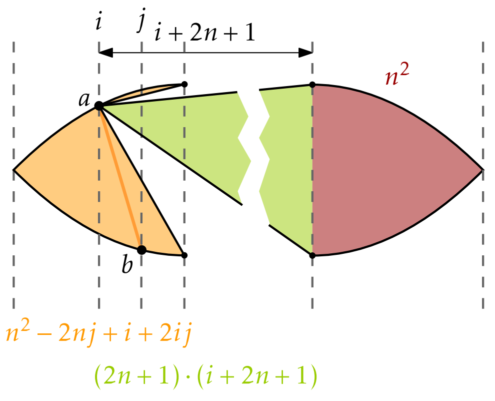


(b) Using a' instead of a.

Figure 6.15: Flipping a to a' along ℓ' increases the total length by the amount in the red area.



(a) Decomposition of the edges for case 1.



(b) Decomposition of the edges for case 2.

Figure 6.16: Decomposition of the edges for the upper bounds.

with equality if and only if either a or b (or both) lie on the boundary of the large gap. In the second case, since a is to the left of b, we have $i > j$ and similar algebra, see Figure 6.16(b), gives

$$|T| = \left(n^2 - 2nj + i + 2ij\right) \ + \ \left(2n+1\right)\left(2n+1+i\right) \ + \ n^2.$$

Since the right-hand side is increasing in i and $i \le k$, we get

$$|T| \le \left(n^2+n\right) + \left(2n+1\right)\left(3n+1\right) + n^2 = 8n^2 + 6n + 1,$$

which is less than the claimed upper bound $10n^2 + 6n + 1$ by a margin. □

CHAPTER 7

Polynomial Time Algorithms for Special Cases

In this chapter we study algorithms that compute exact longest plane spanning trees. First of all, one might hope that a simple greedy algorithm, that locally improves a given spanning tree by closing a cycle with an edge and then removing a shorter edge to increase the overall weight of the spanning tree, works. However, in Section 7.1, we explicitly construct a set of sites and a matching spanning tree, such that this algorithm fails.

Thus we aim to find efficient algorithms for special cases. For convex site sets, a standard dynamic programming approach, similar to the minimum weight triangulation [Gil79; Kli80], yields an $O(n^3)$ time algorithm. We describe this algorithm in Section 7.2

When considering general site sets, we restrict our attention to finding the longest plane spanning tree with bounded diameter. To be precise, in Section 7.3 we show that the longest plane spanning tree of diameter at most 3 can be computed in polynomial time. Such a tree may be relevant in providing an approximation algorithm with a better approximation factor, but as we will see, its efficient computation is not trivial.

In Section 7.4 we show how to compute in polynomial time a longest spanning plane tree of the following form: all sites are connected to one of three distinguished sites on the boundary of the convex hull. Again, such a tree may play an important role in designing future approximation algorithms. Intuitively, it seems to be better than the three stars considered in the approximation algorithm in Section 6.3, in the case when there is a point in the far region.

7.1 A Simple Greedy Algorithm Fails

The greedy algorithm `AlgGreedy(`S`)` works as follows. We start by computing any plane spanning tree T of S. Then, in each iteration we check if there are four points a, b, c, d with the following properties:

(a) a and b are currently not connected by an edge,

(b) $\{c, d\}$ is an edge in the cycle closed by adding the edge $\{a, b\}$; and

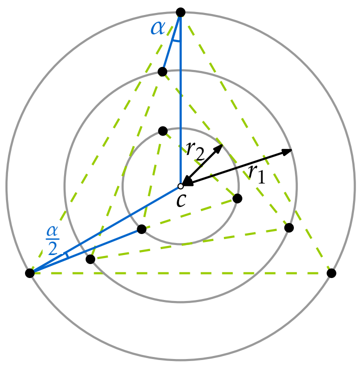


Figure 7.1: Construction of the site set.

(c) the tree T' defined by replacing $\{c,d\}$ with $\{a,b\}$ is plane and is longer than T.

If such sites a,b,c,d exist, set $T = T'$ and continue. After no more such sites are available, the algorithm outputs the current tree T.

In the remainder of this section we show that there are cases in which this algorithm does not yield the correct result.

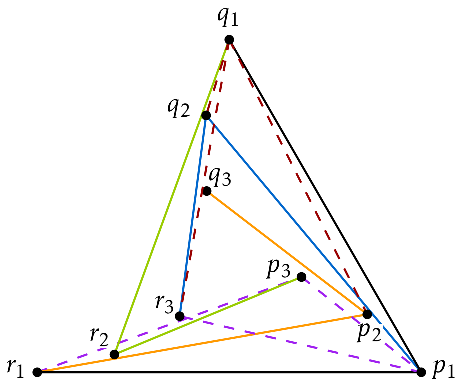
Lemma 7.1. *There is a site set S for which the algorithm* `AlgGreedy`*(S)) fails to compute the longest plane tree.*

Proof. We construct a set S consisting of nine sites to show the claim. The first three sites of S are the vertices of an equilateral triangle. We assume that two of these vertices are on a horizontal line and we denote by c the center of this triangle. Let r_0 be the circumradius of the triangle.

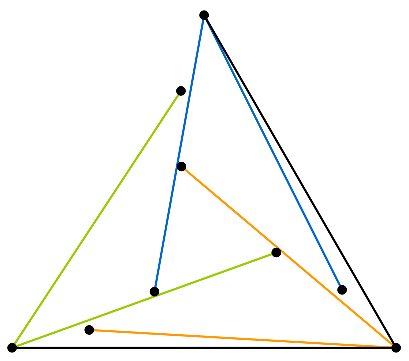
Inside of this triangle, we place the vertices of two smaller equilateral triangles, where again the smaller is contained in the larger one. Set $\alpha = 17°, r_1 = \frac{2}{3}r_0$ and $r_2 = \frac{1}{3}r_0$. We then place the first of the additional triangles on the circle $\partial D(c, r_1)$ in a way that the vertices have an angle of α to the nearest angular bisector of the outer triangle. We place the second additional triangle on the circle $\partial D(c, r_2)$ again with an angle to the nearest bisector of the outer triangle. However, this time the angle is $\alpha/2$. This construction is visualized in Figure 7.1.

Now consider the tree on this site set depicted by the solid edges in Figure 7.2(a). Note that the green, blue and orange edges are rotationally symmetric. The purple dashed edges in Figure 7.2(a) are representatives of the possible edges currently not in the tree, which connect the outer to the inner triangle. However as they are either the smallest edges in the cycle they close or they intersect some edge not in a cycle, the algorithm cannot choose any of these edges as the pair $\{a, b\}$.

Representatives of the possible edges connecting the outer to the middle triangle are depicted in red. The red edge $\{q_1, p_2\}$ is not in a cycle with the edge it crosses, and thus the edge $\{q_1, p_2\}$ cannot be added by the greedy algorithm. For the edge $\{q_1, q_2\}$ and the possible edges connecting the middle to the inner triangle, it can similarly be seen that they are shorter than any edge of the current tree. The possible edge $\{q_1, r_3\}$ is shorter than the blue edge $\{q_2, p_1\}$ it crosses, and thus these two edges cannot be swapped. Finally,



(a) A tree which cannot be locally improved.



(b) A tree where each pair of edges in the same color is at least as long as the matching pair in Figure 7.2(a).

Figure 7.2: The algorithm `AlgGreedy(S)` can output a non-optimal spanning tree.

for the edges along the triangles or the edges connecting the interior and the middle triangle, it can easily be verified that there will be no strictly smaller edge in the unique cycle closed by them.

On the other hand, in the tree depicted in Figure 7.2(b), each pair of same colored edges is longer than its counterpart in Figure 7.2(a). Therefore `AlgGreedy(S)` does not return a correct result. □

7.2 Finding the Longest Tree in a Convex Site Set

Let S be a site set in convex position. In this section, we give a polynomial time algorithm that finds the longest plane spanning tree on S in $O(n^3)$ time.

Lemma 7.2. *The longest plane spanning tree of a convex site set of size n can be found in $O(n^3)$ time.*

Proof. We solve the problem using dynamic programming. For this, let the sites be ordered $s_1,\dots,s_n$ along their counterclockwise order on the convex hull. In the following, we interpret all indices modulo n. We consider suitable subproblems: let $Z(i,j)$ be the length of the longest plane spanning tree for the sites $s_{i+1},s_{i+2},\dots,s_{j-1}$ where at least one site is connected to either s_i or s_j. We claim that the following recurrence correctly describes the solutions for $Z(i,j)$:

$$Z(i,j) = \begin{cases} 0 & \text{if } |i-j| \le 1 \\ \max_{k\in\{i+1,\dots,j-1\}}\big(Z(i,k)+Z(k,j)+\max(\|s_is_k\|,\|s_js_k\|)\big) & \text{otherwise} \end{cases} \tag{7.1}$$

To see that the recurrence is correct, let s_k be a site and assume that it is either the last site in the sequence $s_{i+1},\dots,s_{j-1}$ connected to s_i or the first site in the same sequence connected to s_j. In both cases, there can be no edge connecting a site in $s_i,\dots,s_{k-1}$ to

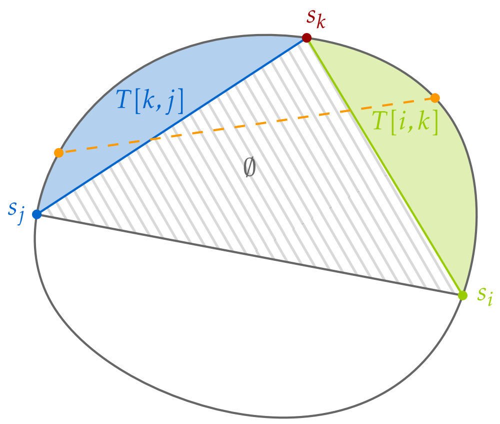


Figure 7.3: To compute $Z(i,j)$, the site s_k is either connected to s_i or to s_j. The orange edge cannot exist.

a site in $s_{k+1},\dots,s_j$ as it would intersect both edges $\{s_i,s_k\}$ and $\{s_k,s_j\}$, see the dashed orange edge in Figure 7.3. In order to connect the spanning tree, at least one of the sites in $s_{i+1},\dots,s_{k-1}$ has to be connected to s_i or s_k, analogously at least one site in $s_{k+1},\dots,s_{j-i}$ has to be connected to s_k or to s_j. Thus, the length for these optimal trees is exactly $Z(i,k)$ or $Z(k,j)$, respectively.

We also have to add the length of the edge $\{s_i,s_k\}$ or $\{s_j,s_k\}$ to the weights of the two subproblems. Thus, if s_k is the last site connected to s_i, we get a total weight of $Z(i,k)+Z(k,j)+\|s_is_k\|$, and in the other case we get $Z(i,k)+Z(k,j)+\|s_js_k\|$. As a site s_k with one of the properties is not known in advance, we have to determine the maximum for both cases and all possible values of k. However, one such site has to exist in a valid solution for $Z(i,j)$ by the definition of the subproblem. Observe, that for a fixed value of k, the same subproblems are considered recursively and thus we can merge both cases into the statement in (7.1). The length of the overall longest spanning tree can then be determined by iterating over all pairs $\{s_i,s_j\}$ of sites and picking the pair that maximizes $Z(i,j)+Z(j,i)+\|s_is_j\|$.

Next, we consider the running time of this algorithm. When filling out an $n\times n$ table for the values of $Z(i,j)$ in increasing sizes of the interval, the entry of each cell can be found in $O(n)$ time, resulting in an overall $O(n^3)$ time. While filling out the table, appropriate pointers can be maintained within the same time bound, to backtrack the actual edges that were picked to obtain the value. To find the longest tree, we have to find the maximum of a total of $O(n^2)$ values that can be found in constant time each, once the table for Z is available. The computation of the table dominates the running time and the claim follows. □

7.3 Finding the Longest Tree of Diameter Three

In this section we show that we can find the longest plane spanning tree of diameter at most three in polynomial time. For any two sites a,b of S, a *bistar rooted* at a and b is

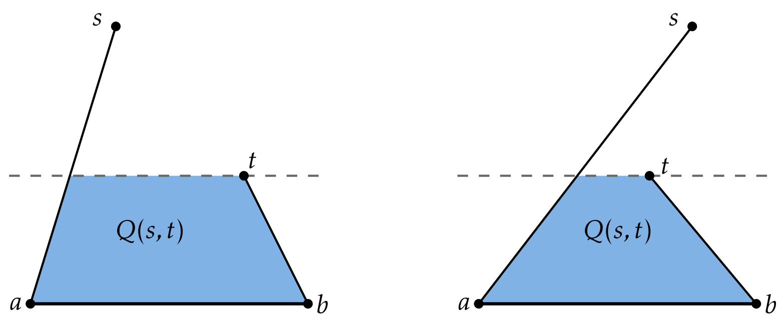


Figure 7.4: Two examples of valid pairs (s,t) with their quadrilaterals $Q(s,t)$ shaded.

a tree that contains the edge $\{a,b\}$, where each site in $S \setminus \{a,b\}$ is connected to either a or b. Note that stars are also bistars and have diameter at most three. Conversely, each tree of diameter at most three has one edge $\{u,v\}$ where each site $s \in S$ has (unweighted) distance at most 1 to either u or v. It follows that all trees of diameter at most three are bistars and it suffices to compute the longest bistar.

We start by describing a dynamic programming algorithm that finds the longest plane bistar rooted at two fixed sites a and b. Without loss of generality, we can assume that the sites a and b lie on a horizontal line with a to the left of b. Furthermore, as we can compute the best plane bistar above and below the line through a and b independently, we can assume that all other sites lie above this horizontal line. In order to solve this problem by dynamic programming, we consider suitable subproblems.

The subproblems considered in the dynamic program are indexed by an ordered pair (s,t) of different sites of S such that the edges $\{a,s\}$ and $\{b,t\}$ do not cross. A pair (s,t) satisfying these condition is a ***valid pair***. For each valid pair (s,t), note that the a,s,t,b,a in this order form a simple, possibly non-convex, polygon. Let $Q(s,t)$ be the convex portion of this polygon below the horizontal line $y = \min\{s_y, t_y\}$, as shown in Figure 7.4. We define the value $Z(s,t)$ to be the length of the longest plane bistar rooted at a and b on the sites in the interior of $Q(s,t)$, without counting $\|ab\|$. If there are no sites of S within the quadrilateral $Q(s,t)$, we set $Z(s,t) = 0$.

Consider the case when the quadrilateral $Q(s,t)$ contains some sites, and let $k_{s,t}$ be the site with largest y-coordinate of S inside of $Q(s,t)$. Then we might connect $k_{s,t}$ either to a or to b. By connecting it to a, we force all sites in the triangle $L_{s,t}$ defined by the edges as and $ak_{s,t}$ and the line $y = (k_{s,t})_y$, to be connected to a. Similarly, when connecting $k_{s,t}$ to b, we force the sites in the triangle $R_{s,t}$ defined by $\overline{bt}$, $\overline{bk_{s,t}}$ and the line $y = (k_{s,t})_y$ to be connected to b. See Figure 7.5 for an illustration. In the former case we are left with the subproblem defined by the valid pair $(k_{s,t},t)$, while in the latter case we are left with the subproblem defined by the valid pair $(s,k_{s,t})$. Formally, for each valid pair (s,t) we have the following recurrence:

$$Z(s,t) = \begin{cases} 0 & \text{if no site of } S \text{ is in } Q(s,t), \\ \max\begin{cases} Z(k_{s,t},t) + \|ak_{s,t}\| + \sum_{l \in L_{s,t}} \|al\| \\ Z(s,k_{s,t}) + \|bk_{s,t}\| + \sum_{r \in R_{s,t}} \|br\| \end{cases} & \text{otherwise} \end{cases}$$

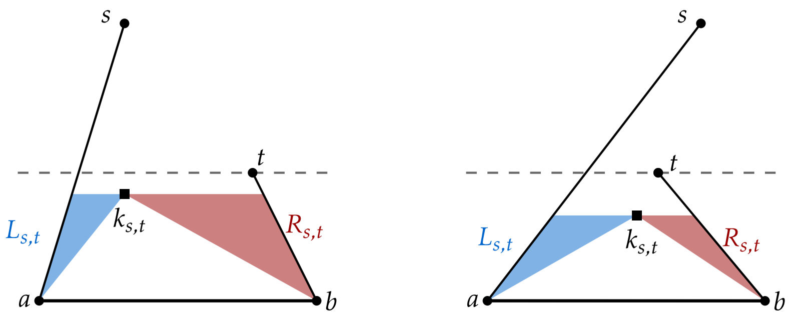


Figure 7.5: Fixing $k_{s,k}$ gives two possible triangular regions where edges are fixed.

Lemma 7.3. *Using $Z(s,t)$ for all valid pairs (s,t) we can find a best plane bistar rooted at a and b.*

Proof. Consider a fixed best plane bistar and assume, without loss of generality, that the highest site is connected to a; the other case is symmetric. Let t^* be the highest site that is connected to b; if t^* does not exist then the bistar degenerates to a star rooted at a. This means that all sites above t^* are attached to a. Let s^* be the site above t^* that, circularly around a, is closest to the edge $\{a,b\}$, see Figure 7.6(a). Note that (s^*,t^*) is a valid pair and all the sites above t^* are to the left or on $\overline{as^*}$. For $s \in S$, denote by L_s the set of sites in S that, circularly around a, are to the left of as. Similarly, denote by R_t the set of sites in S below t and to the right of bt, when sorted circularly around b, see Figure 7.6(b). The length of this optimal plane bistar rooted at a and b is then

$$\left(\sum_{l\in L_{s^*}} \|al\|\right) + \left(\sum_{r\in R_{t^*}} \|br\|\right) + \|as^*\| + \|bt^*\| + \|ab\| + Z(s^*,t^*).$$

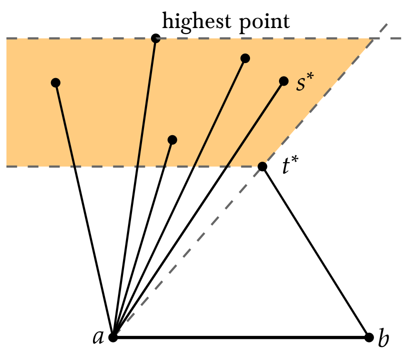
On the other hand, each of the values of the form

$$\left(\sum_{l\in L_{s}} \|al\|\right) + \left(\sum_{r\in R_{t}} \|br\|\right) + \|as\| + \|bt\| + \|ab\| + Z(s,t), \tag{7.2}$$

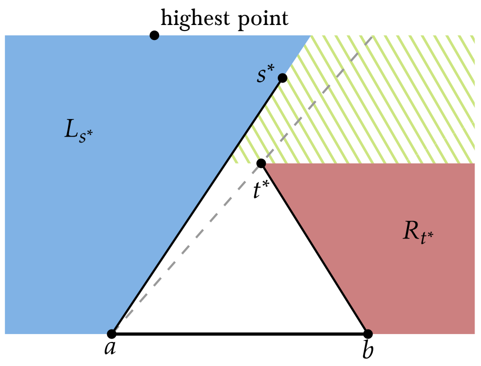
where (s,t) is a valid pair of sites such that $s_y > t_y$, is the length of a plane, not necessarily spanning, bistar rooted at a and b. It is not spanning, if there is some site above t and to the right of as.

Taking the maximum of $|T_a|$ for each $a \in S$ and (7.2) over the valid pairs s,t such that $s_y > t_y$ gives the longest plane bistar for which the highest site is connected to a. A symmetric formula gives the best plane bistar if the highest site is connected to b. Taking the maximum of both cases yields the optimal value. The actual edges of the solution can be backtracked by standard methods. □

Next next step is to show that the dynamic program can be implemented efficiently.



(a) The site s^* is the site of S in the orange region that is angularly closest to t^* around a.



(b) The region containing the sets L_{s^*} and R_{t^*} are marked blue and red. The region with bars is empty.

Figure 7.6: The best bistar is found by using the dynamic program.

Lemma 7.4. *The algorithm described in the proof of Lemma 7.3 can be implemented in* $O(n^2)$ *time.*

Proof. A main complication in implementing the dynamic program and evaluating (7.2) efficiently is finding sums of the form $\sum_{l\in\Delta\cap S}\|al\|$ or $\sum_{r\in\Delta\cap S}\|br\|$, and the highest site in $\Delta\cap S$, where Δ is a query triangle (with one vertex at a or b). These type of sums can be seen as range searching queries which can be handled using standard data structures [CY84; Mat93]: after preprocessing S in time $O(n^2 \operatorname{polylog} n)$, any such query can be answered in $O(\operatorname{polylog} n)$ time. Noting that there are $O(n^2)$ such queries, a running time of $O(n^2 \operatorname{polylog} n)$ is immediate. However exploiting our specific structure and using careful bookkeepingm we can reduce the running time down to $O(n^2)$.

As a preprocessing step, we first compute two sorted lists $\mathcal{L}_a$ and $\mathcal{L}_b$ of $S\setminus\{a,b\}$. The list $\mathcal{L}_a$ is sorted by the angle a and, similarly $\mathcal{L}_b$ is sorted by the angle at b.

The values $\sum_{l\in L_p}\|al\|$ and $\sum_{r\in R_q}\|br\|$, depicted in Figure 7.6, for all $s,t\in S$ can be trivially computed in $O(n^2)$ overall time: there are $O(n)$ such values and for each of them we can scan the whole set S to explicitly get L_s or R_t in $O(n)$ time. There are faster ways of doing this, but for our result this suffices.

Assuming that additionally the values $Z(s,t)$ are already available for all valid pairs (s,t), we can evaluate (7.2) or the symmetric formula in constant time. For any two sites s,t we check whether they form a valid pair and whether $s_y > t_y$ in constant time. We can then either evaluate the formula, again in constant time. Thus, in $O(n^2)$ time we obtain the optimal solution.

It remains to compute the values $Z(s,t)$ for all valid pairs (s,t). First, we explain how to compute all the triples of the form $(s,t,k_{s,t})$ for all valid pairs s,t in $O(n^2)$ time. Then we group these triples in a clever way to implement the dynamic program efficiently.

We focus on the triples with $s_y < t_y$ and show how to find the triples of the form $(s,\cdot,\cdot)$ for a fixed s. The case with $t_y < s_y$ and a fixed t is symmetric. Let W be the sites of S to the right of the ray $\overrightarrow{bs}$ above the horizontal line $y = s_y$. Furthermore, let K be the sites of

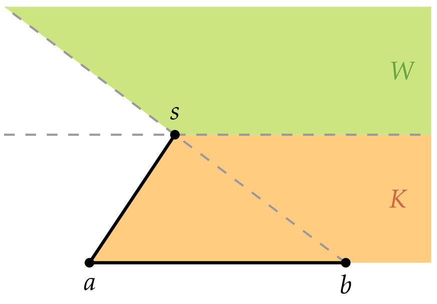


Figure 7.7: The regions defining W and K for fixed a, b and s.

S to the right of $\overline{as}$ and with y-coordinate between y_a and s_y. An illustration of W and K can be found in Figure 7.7. Any site t with $t_y > s_y$ forms a valid pair s, t if and only if t lies in W. The site $k_{s,t}$ must lie in K by its definition.

We use $\mathcal{L}_b$ to find the triples $(s, \cdot, \cdot)$. We iterate through the list in clockwise order starting at the ray $\overrightarrow{ba}$ and keep track of the highest $k^* \in K$ encountered so far. If the current site lies in $S \setminus (K \cup W)$ we simply skip it. If the current site lies in K we update k^* if necessary. Finally, if the current site lies in W, we report the triple $(s, t, k^* = k_{s,t})$.

For a fixed s we only iterate $\mathcal{L}_b$ once. Thus, for this fixed s the running time for finding all triples $(s, t, k_{s,t})$ with (s, t) forming a valid pair and $s_y < t_y$ is $O(n)$. By applying the procedure we just described and its symmetric counterpart to all $s \in S \setminus \{a, b\}$ we find all triples $(s, t, k_{s,t})$ where s, t is a valid pair in overall $O(n^2)$ time.

To compute $Z(s, t)$ for all valid pairs using dynamic programing, we also need to compute the corresponding values $\sum_{l \in L_{s,t}} \|al\|$ and $\sum_{r \in R_{s,t}} \|br\|$. Refer to Figure 7.5 to recall the definition of $L_{s,t}$ and $R_{s,t}$. For the following procedure we shift the focus to the site k. For each site $k \in S \setminus \{a, b\}$ we collect all triples $(s, t, k = k_{s,t})$, and compute the sums $\sum_{l \in L_{s,t}} \|al\|$ and $\sum_{r \in R_{s,t}} \|br\|$ in linear time for each fixed k, as follows.

We concentrate on the first type of sum, $\sum_{l \in L_{s,t}} \|al\|$, where t plays no role, as the sum is defined by s and k. For the following description we assume that $\mathcal{L}_a$ is sorted in counterclockwise order. We create a sorted subsequence $\mathcal{L}_a^k$ of $\mathcal{L}_a$ containing only the sites with y-coordinate below k_y and an angle at a larger than the angle between $\overline{ab}$ and $\overline{ak}$, see Figure 7.8. We iterate over the elements of $\mathcal{L}_a$, starting at the successor of k. While iterating, we maintain the last element from $\mathcal{L}_a^k$ we have seen and the prefix sum $\sum_l \|al\|$ of all points l in $\mathcal{L}_a^k$ we encountered so far. When advancing to the next site s in $\mathcal{L}_a$ there are two possibilities. If s also lies in $\mathcal{L}_a^k$, we advance in $\mathcal{L}_a^k$ and update the prefix sum. Otherwise, we can report $\sum_{l \in L_{s,t'}} \|al\|$ to be the current prefix sum for all t' in a triple (s, t', k).

For any fixed k this process iterates through the list $\mathcal{L}_a$ at most twice and can thus be executed in $O(n)$ time. A symmetric procedure can be carried out for $\sum_{r \in R_{s,t}} \|br\|$ using $\mathcal{L}_b$. As in the case for finding the triples, this results in $O(n^2)$ overall time to compute all relevant sums $\sum_{l \in L_{s,t}} \|al\|$ and $\sum_{r \in R_{s,t}} \|br\|$.

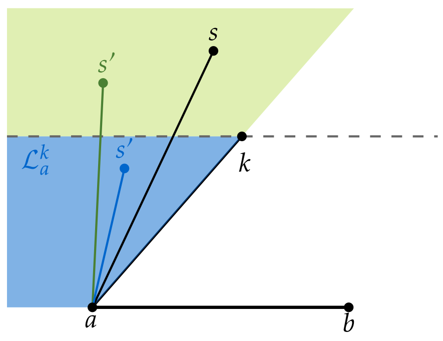


Figure 7.8: We only consider the sites in the union of the green and blue region. If the next point after s is contained in $\mathcal{L}_a^k$, we update the prefix sum. In the other case we can return the stored sum.

Now we can implement the dynamic program in the straightforward way. Using the precomputed information we spend $O(1)$ time for each value $Z(s,t)$, for a total running time of $O(n^2)$. □

Theorem 7.5. *For any set S of n sites in general position, a longest plane tree of diameter at most three on S can be computed in $O(n^4)$ time.*

Proof. In Lemma 7.4 we show that for any two fixed sites a, b, the longest plane bistar rooted at a and b can be computed in time $O(n^2)$. By iterating over all possible pairs of roots and taking the longest such plane bistar, we find the longest plane spanning tree of diameter at most three in time $O(n^4)$. □

7.4 Extending the Approach to Special Trees of Diameter Four

Now we show how to extend the ideas presented in Section 7.3 to get a polynomial time algorithm for special trees of diameter four. Given three sites a,b,c of S, a *tristar rooted* at a,b and c is a tree such that each edge has at least one incident site at a, b or c. We will focus on the case when these three points lie on the convex hull of the site set S and the sites are fixed in advance. In case, we want to get the longest tristar among all of this form, we can iterate over all possible triples of specified sites like in the proof of Theorem 7.5.

In order to be able to use a dynamic programming approach, we first have to make some assumptions on the site set. Let a,b,c be the specified sites on the boundary of the convex hull of S. We assume, without loss of generality, that the edges ac and bc are present in the tristar; the other cases are symmetric. We further assume that a and b lie on a horizontal line, with a to the left of b.

The regions to the left of $\{a,c\}$ and to the right of $\{b,c\}$, depicted blue in Figure 7.9, can be solved independently from the rest of the instance: the edges $\{a,c\}$ and $\{b,c\}$ block any edge connecting a site in one of those regions to a site outside the region. Each

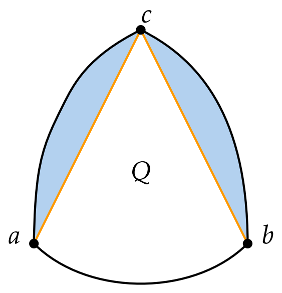


Figure 7.9: The regions cut off by ac and bc can be solved independently as bistars.

one of these regions can be solved as plane bistars, one rooted at a, c and one rooted at b, c, using Lemma 7.4. It remains to solve the, again independent, problem for the sites enclosed by ac, cb and the portion of the boundary of the convex hull from a to b in counter-clockwise order. We call this region Q and in the following assume without loss of generality that all sites of S are contained in Q.

To solve the problem for Q we will use a variation of the dynamic programming approach for bistars. For any two sites s, t of S, let us write $s \preceq_c t$, when in the counterclockwise order around c we have the horizontal ray from c towards $-\infty$ to the left, then $\overrightarrow{cs}$ and then $\overrightarrow{ct}$, or the segments cs and ct are collinear. Since we assume general position, the latter case only occurs when $s = t$.

The subproblems for the dynamic program are defined by a 5-tuple (s, s', m, t', t) of sites such that

(a) s' and t' are distinct

(b) $s \preceq_c s' \preceq_c m \preceq_c t' \preceq_c t$; and

(c) s' and t' are contained in the closed triangle cst.

These 5-tuples extend the definition of a valid pair from Section 7.3 and are called ***valid tuples***, see Figure 7.10. Note that the first and the second condition imply that $\overline{as}$ and $\overline{bt}$ do not cross. Some of the sites may be equal in the tuple; to be precise we may have $s = s'$ or $t = t'$. Intuitively, the role of these five sites can be interpreted as follows. The sites s and t play the same role as in Section 7.3, in the sense that we assume that s is connected to a and t is connected to b. Furthermore, m is the lowest site currently connected to c. The sites s' and t' are the sites connected to a and b, that induce the most stringent constraints on the part of Q below s' and t' visible from c. Equivalently, s' and t' are the left and right neighbors of the ray $\overrightarrow{cm}$ in the angular order around c among the sites connected to a and b respectively.

For each valid tuple (s, s', m, t', t), let $Q(s, m, t)$ be the sites of S contained in Q below the polygonal path a, s, t, b, and below the horizontal line through the lowest of the sites s, m, r. Note that the point m is used only to define the horizontal line.

Given a valid tuple (s, s', m, t', t), we define $Z(s, s', m, t', t)$ as the length of the optimal plane tristar rooted at a, b, c for the sites in the interior of $Q(s, m, t)$, without counting $\|ac\| + \|bc\|$, and with the additional restriction that a site k can be connected to c only if

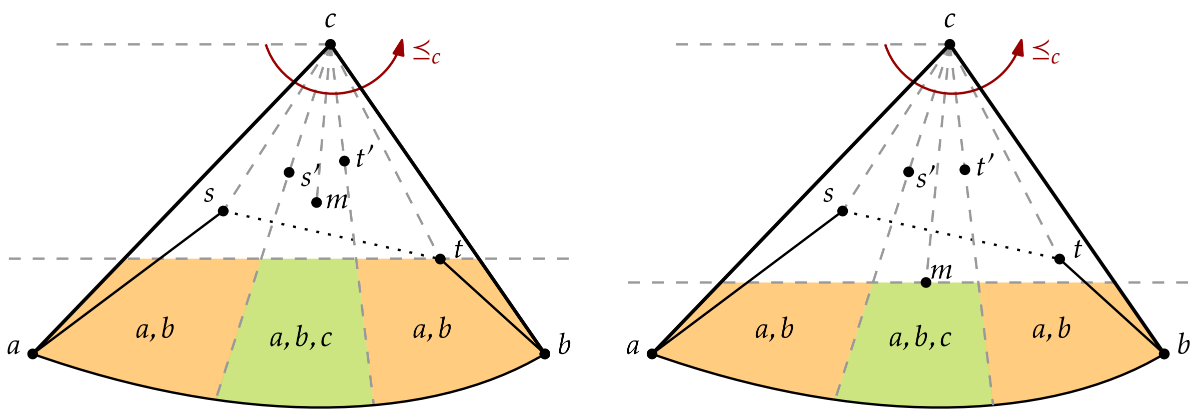


Figure 7.10: Two examples of valid tuples (s, s', m, t', t) with the region $Q(s, m, t)$ shaded. Each $Q(s, m, t)$ is split into 3 regions telling, for each of them, which roots can be used for that region.

$s' \preceq_c k \preceq_c t'$. This last condition is equivalent to saying that no edge incident to c in the tristar can cross $\overline{as'}$ or $\overline{bt'}$. Thus, we are looking at the length of a longest plane forest in which each site in the interior of $Q(s, m, t)$ must be connected to either a, b or c, and no edge crosses $\overline{as'}$ or $\overline{bt'}$.

If $Q(s, m, t)$ contains no sites, then $Z(s, s', m, t', t) = 0$. In the other case, we find the highest site $k = k_{s,m,t}$ in $Q(s, m, t)$ and check to which roots it can be attached, and which edges are enforced by each possible root. We can always connect k to a or b, as $Q(s, m, t)$ is free of obstacles. If $s' \preceq_c k \preceq_c t'$, then we can also connect k to c. If k is connected to c it only lowers the boundary of the region $Q(\cdot, \cdot, \cdot)$ that has to be considered. However, if k is connected to a or b, it splits off independent regions, some of them can be attached to only one of the roots, some of them are essentially a bistar problem rooted at c and one of a, b. To state the recursive formulas precisely, for a region R and roots p and q, let $\mathrm{BS}_{p,q}(R)$ be the length of the optimal plane bistar rooted at p and q for the sites in R. Such a value can be computed using Lemma 7.4.

Formally, the recurrence for $Z(s, s', m, t', t)$ looks as follows. If $Q(s, m, t)$ is empty, then $Z(s, s', m, t', t) = 0$. If $Q(s, m, t)$ is not empty, let k be the highest site in $Q(s, m, t)$ and we distinguish three subcases:

Case 1 $k \preceq_c s'$: See Figure 7.11 for an illustration of the notions defined for this case. Let L be the sites of $Q(s, m, t)$ above $\overline{ak}$. Let B be the sites of $Q(s, m, t)$ below $\overline{bk}$ and let G be the set of sites $p \in Q(s, m, t) \setminus B$ such that $k \preceq_c p \preceq_c s'$. Let g be the site with largest angle at b among the sites in G. Then we define R' to be the set of all sites p in $Q(s, m, t) \setminus B$ with $s' \preceq_c p \preceq_c t'$ which are above $\overline{gb}$ and let R be the remaining sites in $Q(s, m, t)$ above $\overline{bk}$. We get the following recurrence:

$$Z(s, s', m, t', t) = \max \begin{cases} Z(k, s', m, t', t) + \|ak\| + \sum_{l \in L} \|al\| \\ \mathrm{BS}_{a,b}(B) + \|bk\| + \mathrm{BS}_{b,c}(R') + \sum_{r \in R} \|br\| \end{cases} \tag{7.3}$$

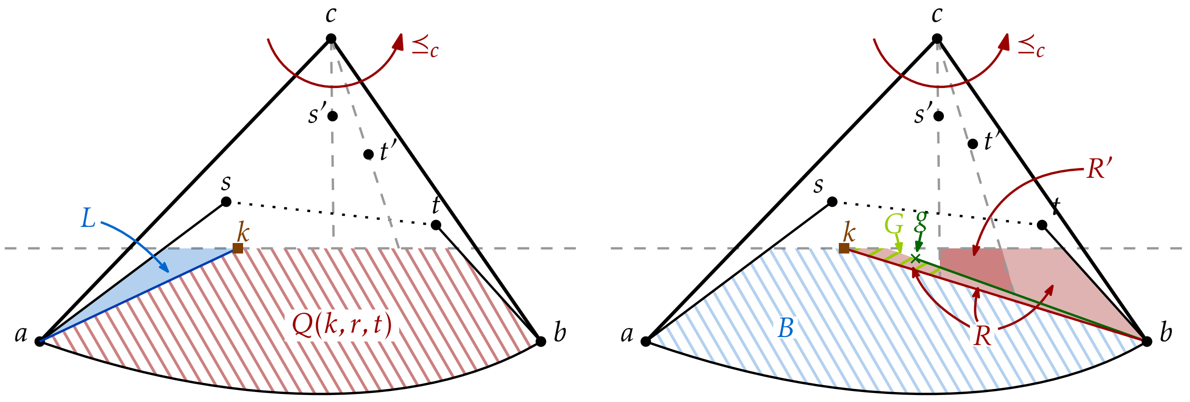


Figure 7.11: The two cases in the recurrence when $k \preceq_c s'$.

Case 2 $s' \preceq_c k \preceq_c t'$: Let L be the sites p of $Q(s,m,t)$ above $\overline{ak}$ such that $p \preceq_c s'$ and let L' be the sites p of $Q(s,m,t)$ above $\overline{ak}$ such that $s' \preceq_c p \preceq_c k$. Furthermore, let R' be the sites p of $Q(s,m,t)$ above bk such that $k \preceq_c p \preceq_c t'$, and let R be the sites p of $Q(s,m,t)$ above bk such that $t' \preceq_c p$, see Figure 7.12. We get the following recurrence in this case:

$$Z(s,s',m,t',t) = \max \begin{cases} Z(s,s',k,t',t) + \|ck\| \\ Z(k,k,m,t',t) + \|ak\| + \mathrm{BS}_{a,c}(L') + \sum_{l \in L} \|al\| \\ Z(s,s',m,k,k) + \|bk\| + \mathrm{BS}_{b,c}(R') + \sum_{r \in R} \|br\| \end{cases} \tag{7.4}$$

Case 3: $t' \preceq_c k$. The case is symmetric to the case $k \preceq_c s'$.

Theorem 7.6. *For any set S of sites in general position and any three specified sites on the boundary of the convex hull of S, the algorithm described above computes the longest plane tree such that each edge is incident to at least one of the three specified sites in $O(n^6)$ time.*

Proof. The values $Z(s,s',m,t',t)$ for all valid tuples (s,s',m,t',t) can be computed using dynamic programming and the formulas described above. The dynamic program is

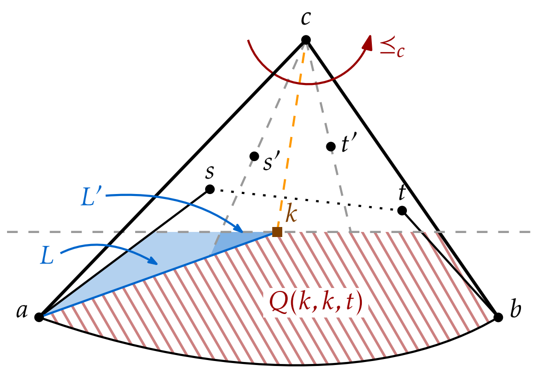


Figure 7.12: One of the cases in the recurrence when $s' \preceq_c k \preceq_c t'$.

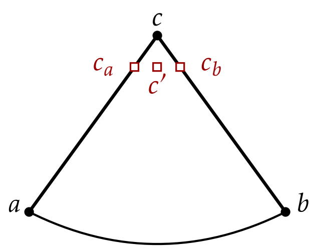


Figure 7.13: Starting Case for the dynamic program.

correct by a simple but tedious inductive argument. We will give the argument for the most involved subcase where $k \preceq_c s'$. The other cases then follow by similar arguments.

Assume $k \preceq_c s'$, then by definition the line segments $\overline{ck}$ and $\overline{as'}$ intersect. Thus k can only be connected to either a or b. First assume that in the optimum solution k is connected to a. Then the edge $\{a,k\}$ prevents any site in L from being connected to b. Thus all sites in L are connected to a, which is taken care of by the $\sum_{l\in L} \|al\|$ term in the recurrence. The sites in $Q(k,m,t)$ can still be connected to both a and b and some sites possibly also to c. This problem is solved optimally by induction. As $L \dot\cup Q(k,m,t)$ contains all sites in $Q(s,m,t)$, this solves the subproblem for $Q(r,m,t)$.

In the other case, k is connected to b in the optimum solution. In this case we again define regions that can be solved independently, however the regions are more involved than in the first case. As $k \preceq_c s'$ any edge connecting c to a site in B would intersect $\{k,b\}$. Thus all sites in B are connected to either a or b, forming a bistar problem, which can be solved using the algorithm from Lemma 7.4. On the other hand, no site in $Q(s,m,t)$ above $\{k,b\}$ can be connected to a, as any such edge would again intersect $\{k,b\}$. However, as the invariant prevents some of the sites above $\{k,b\}$ to be connected to c, we cannot simply use these sites as an input for a bistar problem. More precisely, only the sites which are between s' and t' in the angular ordering around c can be connected to c, the remaining sites can only be connected to b. The region G contains part of the sites that can only be connected to b. The edge $\{b,g\}$ is the edge that shaves off the most restricting part of the sites between s' and t'. Connecting any site in this shaved off region with c would lead to an intersection with $\{g,b\}$, thus all sites in this bottom part also have to be connected to b. The region R contains all sites that are in one way or the other enforced to be connected to b and their length is taken care of in the $\sum_{r\in R} \|rb\|$ part of the formula. The remaining sites, are those that lie in R'. Connecting them to either c or b cannot cross any of the edges incident to a site in R and thus the optimal solution for this part can be found by an independent bistar problem.

Now we can focus on the running time. There are a few solutions for bistars that have to be computed. These have the form $\mathrm{BS}_{p,q}(R)$ for $p,q \in \{a,b,c\}$ and some region R of constant size description. More precisely, each such region is defined by four sites of S in the case of R' and B in (7.3), which are defined by k, g, t', t, or two sites of S in case of L' and R' in (7.4). Thus, we have $O(n^4)$ different bistar problems, and each of them can be solved in $O(n^2)$ time using Lemma 7.4, for a total of $O(n^6)$ time for all bistar problems.

To compute the optimal plane tristar, we add three dummy sites to S before we start the dynamic programming. We add a site c_a on the edge ac arbitrarily close to c, a site c_b on the edge bc arbitrarily close to c, and a site c' between c_a and c_b, see Figure 7.13. We perturb the sites c_a, c' and c_b slightly to get them into general position. Note that (c_a, c_a, c', c_b, c_b) is a valid tuple. We can now compute $Z(c_a, c_a, c', c_b, c_b)$ by implementing the recurrence with standard dynamic programming techniques. There are $O(n^5)$ valid tuples. In addition to the bistar problems, we have to find $O(1)$ sums of the form $\sum_{l \in L} \|al\|$ and $\sum_{r \in R} \|br\|$ for each valid pair. After sorting the sites angular around a and b in $O(n \log n)$ time as a preprocessing step, each sum can be found in $O(n)$ time. Thus, after having the solutions for all possible bistars available, we can find the values $Z(s, s', m, t', t)$ in $O(n^6)$ time. By adding $\|ac\|$, $\|bc\|$ and the lengths of the independent bistars from Figure 7.9 to $Z(c_a, c_a, c', c_b, c_b)$, we get the length of the longest plane tristar rooted at a, b and c in the promised time. $\square$

CHAPTER 8
Conclusion

In this thesis we considered several algorithmic problems related to geometric graphs. We considered the related graph classes of disk graphs and transmission graphs, as well as spanning trees of site sets.

Triangles and Girth We were able to extend the results by Kaplan et al. [Kap+19] on triangles in disk graphs to the weighted girth. By using a suitable grid and an extension of Dijkstra's algorithm, we were able to match the $O(n \log n)$ expected time bound of the unweighted girth. The bound is optimal by the reduction from ε-Closeness of Polishchuk [Pol17].

We then moved our attention to triangles in transmission graphs, and after showing that a similar approach as for disk graphs is not feasible, found a new approach for this type of graphs. By using two types of batched range queries that can both be solved in expected time $O(n \log n)$, we developed a algorithms that find a (shortest) triangle in a transmission graph in $O(n \log n)$ expected time. The $O(n \log n)$ bounds for the two triangle finding problems are again optimal by the reduction from ε-Closeness. For cycles in transmission graphs, we focused on the decision problem: *is there a cycle with at most k edges in $\mathcal{T}(S)$?* We gave an algorithm that has a polynomial dependency on n and an exponential dependency on k for this problem.

There are still some open problems in this area. Firstly, while our algorithms match the $\Omega(n \log n)$ lower bounds for triangles and the girth, we do so with randomized algorithms. To settle the complexity of these problems, we would have to find *deterministic* $O(n \log n)$ time algorithms for triangles and the girth in disk graphs, as well as for triangles in transmission graphs. Secondly, our bound for cycles, even unweighted ones, in transmission graphs is significantly worse than the situation for disk graphs. As transmission graphs that do not contain triangles can be dense, finding an algorithm that is significantly faster than applying Theorem 3.10 to each vertex in both the weighted and unweighted case and whose running time does not depend on k seem to be hard. Possible venues of research in this area would be to improve the running time, or to find general or conditional lower bounds for this case.

A topic that is closely related to finding the girth is that of finding shortest paths in disk graphs and transmission graphs. There are many results for the single source shortest path problem in unit disk graphs, that is disk graphs, where all sites have the same radius.

Cabello and Jejčič [CJ15] give an optimal $O(n \log n)$ algorithm for the unweighted case. For the weighted case, Wang and Xue [WX20] give the currently best known algorithm with a running time of $O(n \log^2 n)$. In their core, both approaches use similar ideas as Dijkstra's algorithm, and heavily rely on the underlying geometry. In particular, the assumption that the graph is a unit disk graph is central to many of their arguments. There have been some approaches to extend results on unit disk graphs to more general disk graphs. A first step towards efficient shortest path algorithms in general disk graphs or transmission graphs, could be to examine the applicability of these approaches to the shortest path problem.

Dynamic Connectivity We considered the problem of decremental dynamic connectivity in disk graphs. By tailoring our approach to the deletion-only setting, we were able to obtain much faster algorithms than in the general case. To be precise, in the setting where all radii are in the interval $[1, \Psi]$, we managed to reduce the dependency on Ψ from polynomial to logarithmic. For general disk graphs, we developed the first decremental data structure with a polylogarithmic update time.

There are still some open questions in the field of dynamic connectivity for disk graphs. First of all, the running times for the updates come from a quite general data structure that maintains certain types of lower envelopes [Kap+21a]. Considering specialized data structures more tailored to disk graphs might improve the update times. For the case of bounded radius ratio, a very similar approach to the one presented in Section 5.1, yields an incremental data structure that has a logarithmic dependency on Ψ in the update time [Kap+21a]. Adapting the proxy graph for general disk graphs given in Section 5.2 for use in an incremental data structure with similar time bounds than our decremental data structure poses an interesting question. A next step would then be to develop fully dynamic data structures. For unit disk graphs, there are already efficient data structures [Kap+21a]. For disk graphs with bounded or unbounded radius ratio, the main issue is that the deletion and subsequent insertion of a single disk can affect many edges. Developing an efficient data structure that maintains the connectivity information under insertions and deletions in an efficient manner seems to be a non-trivial task that is central to developing fully dynamic connectivity data structures for general disk graphs.

Spanning Trees We considered the problem of finding a long plane spanning tree on a site set. First, we gave an algorithm that improves the approximation factor for finding the longest plane spanning tree to 0.546. Furthermore, we gave a dynamic programming algorithm that finds the longest tree of diameter three in $O(n^4)$ time. The dynamic programming approach was then extended to work on special trees of diameter four.

These results are far from settling all open problems in this area. First of all, while the analysis given in Chapter 6 is tight in every step, it is hard to believe that the whole analysis is tight. We conjecture that the actual approximation factor of this algorithm is better and finding an analysis showing this improved approximation factor would be of interest. Furthermore, the algorithm in Section 7.3 gives the longest plane tree of diameter three, while on the other hand we showed that no such tree can give a better approximation factor than $\frac{5}{6}$. This upper bound however is much larger than the bound given by our approximation algorithm. So a natural question would be to ask, what approximation factor can be achieved by using the optimal tree with diameter at most three.

We were able to extend the exact polynomial time algorithm from diameter three to some trees of diameter four. This poses the question, if there is a polynomial time algorithm that outputs the best plane spanning tree for any fixed diameter d. This is of particular interest, as a result of Cabello et al. [Cab+21] implies that a hypothetical PTAS would have to consider trees of unbounded diameter. Their result gives a constant upper bound on the approximation factor achieved by any tree of fixed diameter d. As it is compatible with our current knowledge that a PTAS can be obtained by computing an optimal tree of diameter say, $O\left(\frac{1}{\varepsilon}\right)$, being able to find an algorithm for any fixed d would be a step towards finding a PTAS.

Finally, there is the big open problem of settling the complexity of finding the longest plane tree, especially if a polynomial time algorithm for the problem would be feasible or if conjectured NP hardness can be shown. This question can also be asked for several other long plane objects, such as paths, cycles or matchings.

Parting Thoughts We started this thesis by observing that the connection between graphs and geometry is as old as the field of graph theory. Even though the fields are not as closely related anymore, many connections can still be made. We considered several problems on geometrically defined graphs. In the case of the long plane spanning trees, no interesting questions would arise without the underlying geometry. Disk graphs and transmission graphs can be represented as abstract graphs, but only the combination of combinatorial and geometric properties of these graphs allowed us to find algorithms and data structures with significantly better guarantees than the best currently known bounds for general graphs.

Bibliography

[AYZ97] N. Alon, R. Yuster, and U. Zwick. “Finding and Counting given Length Cycles”. In: *Algorithmica* 17.3, pp. 209–223. 1997.

[ARS95] Noga Alon, Sridhar Rajagopalan, and Subhash Suri. “Long Non-Crossing Configurations in the Plane”. In: *Fundam. Inform.* 22.4, pp. 385–394. 1995.

[Aro98] Sanjeev Arora. “Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems”. In: *Journal of the ACM* 45.5, pp. 753–782. 1998.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge: Cambridge University Press. 2009.

[AC04] Sanjeev Arora and Kevin L. Chang. “Approximation Schemes for Degree-Restricted MST and Red-Blue Separation Problems”. In: *Algorithmica* 40.3, pp. 189–210. 2004.

[Bar+03] Alexander I. Barvinok, Sándor P. Fekete, David S. Johnson, Arie Tamir, Gerhard J. Woeginger, and Russell Woodroofe. “The Geometric Maximum Traveling Salesman Problem”. In: *Journal of the ACM* 50.5, pp. 641–664. 2003.

[Ben83] Michael Ben-Or. “Lower Bounds for Algebraic Computation Trees”. In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing.* STOC ’83. New York, NY, USA: Association for Computing Machinery, pp. 80–86. 1983.

[Bin20a] Ahmad Biniaz. “Euclidean Bottleneck Bounded-Degree Spanning Tree Ratios”. In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020.* SIAM, pp. 826–836. 2020.

[Bin20b] Ahmad Biniaz. *Improved Approximation Ratios for Two Euclidean Maximum Spanning Tree Problems.* arXiv: `2010.03870 [cs]`. URL: `http://arxiv.org/abs/2010.03870`. 2020.

[Bin+19] Ahmad Biniaz, Prosenjit Bose, Kimberly Crosbie, Jean-Lou De Carufel, David Eppstein, Anil Maheshwari, and Michiel Smid. “Maximum Plane Trees in Multipartite Geometric Graphs”. In: *Algorithmica* 81.4, pp. 1512–1534. 2019.

[Buc+11] Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. “Preprocessing Imprecise Points for Delaunay Triangulation: Simplified and Extended”. In: *Algorithmica* 61.3, pp. 674–693. 2011.

[BM11] Kevin Buchin and Wolfgang Mulzer. "Delaunay Triangulations in $O(\mathbf{sort}(n))$ Time and More". In: *Journal of the ACM* 58.2, pp. 1–27. 2011.

[Cab+20] Sergio Cabello, Aruni Choudhary, Michael Hoffmann, Katharina Klost, Meghana M Reddy, Wolfgang Mulzer, Felix Schröder, and Josef Tkadlec. "A Better Approximation for Longest Noncrossing Spanning Trees". In: *36th European Workshop on Computational Geometry (EuroCG).* 2020.

[Cab+21] Sergio Cabello, Michael Hoffmann, Katharina Klost, Wolfgang Mulzer, and Josef Tkadlec. *Long Plane Trees.* arXiv: `2101.00445 [cs]`. URL: `http://arxiv.org/abs/2101.00445`. 2021.

[CJ15] Sergio Cabello and Miha Jejčič. "Shortest Paths in Intersection Graphs of Unit Disks". In: *Computational Geometry* 48.4, pp. 360–367. 2015.

[Cha16] Timothy M. Chan. "A Simpler Linear-Time Algorithm for Intersecting Two Convex Polyhedra in Three Dimensions". In: *Discrete & Computational Geometry* 56.4, pp. 860–865. 2016.

[Cha04] Timothy M. Chan. "Euclidean Bounded-Degree Spanning Tree Ratios". In: *Discrete & Computational Geometry* 32.2, pp. 177–194. 2004.

[Cha99] Timothy M. Chan. "Geometric Applications of a Randomized Optimization Technique". In: *Discrete & Computational Geometry* 22.4, pp. 547–567. 1999.

[CPR11] Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. "Dynamic Connectivity: Connecting to Networks and Geometry". In: *SIAM Journal on Computing* 40.2, pp. 333–349. 2011.

[CL13] Hsien-Chih Chang and Hsueh-I Lu. "Computing the Girth of a Planar Graph in Linear Time". In: *SIAM Journal on Computing* 42.3, pp. 1077–1094. 2013.

[Cha92] Bernard Chazelle. "An Optimal Algorithm for Intersecting Three-Dimensional Convex Polyhedra". In: *SIAM Journal on Computing* 21.4, pp. 671–696. 1992.

[CM11] Bernard Chazelle and Wolfgang Mulzer. "Computing Hereditary Convex Structures". In: *Discrete & Computational Geometry* 45.4, pp. 796–823. 2011.

[CY84] Richard Cole and Chee-Keng Yap. "Geometric Retrieval Problems". In: *Information and Control* 63.1/2, pp. 39–57. 1984.

[Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* 3rd. The MIT Press. 2009.

[dBer+08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* 3rd. Springer-Verlag, Berlin. 2008.

[DK82] David P. Dobkin and David G. Kirkpatrick. "Fast Detection of Polyhedral Intersections". In: *Automata, Languages and Programming.* International Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 154–165. 1982.

[DT10] Adrian Dumitrescu and Csaba D. Tóth. "Long Non-Crossing Configurations in the Plane". In: *Discrete & Computational Geometry* 44.4, pp. 727–752. 2010.

[EIK01] Alon Efrat, Alon Itai, and Matthew J. Katz. "Geometry Helps in Bottleneck Matching and Related Problems". In: *Algorithmica* 31.1, pp. 1–28. 2001.

[Epp00] David Eppstein. "Spanning Trees and Spanners". In: *Handbook of Computational Geometry.* Ed. by Jörg-Rüdiger Sack and Jorge Urrutia. North Holland / Elsevier, pp. 425–461. 2000.

[Epp+92] David Eppstein, Giuseppe F Italiano, Roberto Tamassia, Robert E Tarjan, Jeffery Westbrook, and Moti Yung. "Maintenance of a Minimum Spanning Forest in a Dynamic Plane Graph". In: *Journal of Algorithms* 13.1, pp. 33–54. 1992.

[Eul41] Leonhard Euler. "Solutio Problematis Ad Geometriam Situs Pertinentis". In: *Commentarii academiae scientiarum Petropolitanae*, pp. 128–140. 1741.

[Eva+16] William Evans, Mereke van Garderen, Maarten Löffler, and Valentin Polishchuk. "Recognizing a DOG Is Hard, But Not When It Is Thin and Unit". In: *8th International Conference on Fun with Algorithms (FUN 2016).* Ed. by Erik D. Demaine and Fabrizio Grandoni. Vol. 49. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 16:1–16:12. 2016.

[Fel04] Stefan Felsner. *Geometric Graphs and Arrangements.* Advanced Lectures in Mathematics. Wiesbaden: Vieweg+Teubner Verlag. 2004.

[For87] Steven Fortune. "A Sweepline Algorithm for Voronoi Diagrams". In: *Algorithmica* 2.1, p. 153. 1987.

[FH09] Andrea Francke and Michael Hoffmann. "The Euclidean Degree-4 Minimum Spanning Tree Problem Is NP-Hard". In: *Proceedings of the 25th ACM Symposium on Computational Geometry.* ACM, pp. 179–188. 2009.

[Gil79] P. D. Gilbert. *New Results in Planar Triangulations.* Technical Report R–850. Univ. Illinois Coordinated Science Lab. 1979.

[GSW98] A. Gräf, M. Stumpf, and G. Weißenfels. "On Coloring Unit Disk Graphs". In: *Algorithmica* 20.3, pp. 277–293. 1998.

[Har11] Sariel Har-Peled. *Geometric Approximation Algorithms.* Vol. 173. Mathematical Surveys and Monographs. Providence, Rhode Island: American Mathematical Society. 2011.

[HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. "Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity". In: *Journal of the ACM* 48.4, pp. 723–760. 2001.

[HS95] M.L. Huson and A. Sen. "Broadcast Scheduling Algorithms for Radio Networks". In: *Proceedings of MILCOM '95.* MILCOM '95. Vol. 2. San Diego, CA, USA: IEEE, pp. 647–651. 1995.

[IR78] Alon Itai and Michael Rodeh. “Finding a Minimum Circuit in a Graph”. In: *SIAM Journal on Computing* 7.4, pp. 413–423. 1978.

[Kap+21a] Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. *Dynamic Connectivity in Disk Graphs.* arXiv: `2106.14935 [cs]`. URL: `http://arxiv.org/abs/2106.14935`. 2021.

[KKM21] Haim Kaplan, Alexander Kauer, and Wolfgang Mulzer. “Sampling Hyperplanes and Revealing Disks”. In: *Proc. 37th European Workshop on Computational Geometry (EWCG).* Proc. 37th European Workshop on Computational Geometry (EWCG), p. 7. 2021.

[Kap+21b] Haim Kaplan, Katharina Klost, Kristin Knorr, and Liam Roditty. “Deletion Only Dynamic Connectivity for Disk Graphs”. In: *38th European Workshop on Computational Geometry (EuroCG)EuroCG*, p. 8. 2021.

[Kap+19] Haim Kaplan, Katharina Klost, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. “Triangles and Girth in Disk Graphs and Transmission Graphs”. In: *27th Annual European Symposium on Algorithms (ESA 2019).* Ed. by Michael A. Bender, Ola Svensson, and Grzegorz Herman. Vol. 144. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 64:1–64:14. 2019.

[Kap+18] Haim Kaplan, Katharina Klost, Wolfgang Mulzer, and Roditty, Liam. “Finding the Girth in Disk Graphs and a Directed Triangle in Transmission Graphs”. In: *Proc. 34nd European Workshop Comput. Geom.(EWCG)*, p. 6. 2018.

[Kap+16] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. “Dynamic Connectivity for Unit Disk Graphs”. In: *Proc. 32nd European Workshop on Computational Geometry (EWCG).* 2016.

[Kap+17] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. “Finding Triangles and Computing the Girth in Disk Graphs”. In: *Proc. 33nd European Workshop Comput. Geom.(EWCG).* 2017.

[Kli80] Gheza Tom Klincsek. “Minimal Triangulations of Polygonal Domains”. In: *Annals of Discrete Math.* 9, pp. 121–123. 1980.

[ŁS11] Jakub Łącki and Piotr Sankowski. “Min-Cuts and Shortest Cycles in Planar Graphs in $O(n \log\log n)$ Time”. In: *European Symposium on Algorithms.* Springer, pp. 155–166. 2011.

[Le 14] François Le Gall. “Powers of Tensors and Fast Matrix Multiplication”. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation.* ISSAC ’14. New York, NY, USA: Association for Computing Machinery, pp. 296–303. 2014.

[Mat93] Jirí Matoušek. “Range Searching with Efficient Hiearchical Cutting”. In: *Discrete & Computational Geometry* 10, pp. 157–182. 1993.

[Mit99] Joseph S. B. Mitchell. “Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems”. In: *SIAM Journal on Computing* 28.4, pp. 1298–1309. 1999.

[Mit17] Joseph S. B. Mitchell. “Shortest Paths and Networks”. In: *Handbook of Discrete and Computational Geometry.* Ed. by Jacob E. Goodman and Joseph O’Rourke. 3rd. Chapman and Hall/CRC, pp. 607–641. 2017.

[MM17] Joseph S. B. Mitchell and Wolfgang Mulzer. “Proximity Algorithms”. In: *Handbook of Discrete and Computational Geometry.* Ed. by Jacob E. Goodman, Joseph O’Rourke, and Csaba D. Tóth. 3rd. Boca Raton: CRC Press, pp. 849–874. 2017.

[Mul04] Wolfgang Mulzer. “Minimum Dilation Triangulations for the Regular N-Gon”. Diplomarbeit. Freie Universität Berlin, Germany. 2004.

[MO20] Wolfgang Mulzer and Johannes Obenaus. “The Tree Stabbing Number Is Not Monotone”. In: *Proceedings of the 36th European Workshop on Computational Geometry (EWCG)*, 78:1–78:8. 2020.

[MR08] Wolfgang Mulzer and Günter Rote. “Minimum-Weight Triangulation Is NP-Hard”. In: *Journal of the ACM* 55.2, 11:1–11:29. 2008.

[NS07] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks.* Cambridge University Press, Cambridge. 2007.

[Pac13] János Pach. “The Beginnings of Geometric Graph Theory”. In: *Erdős Centennial.* Ed. by László Lovász, Imre Z. Ruzsa, and Vera T. Sós. Vol. 25. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 465–484. 2013.

[Pac04] János Pach. *Towards a Theory of Geometric Graphs.* American Mathematical Soc. 2004. 300 pp. Google Books: `zosbCAAAQBAJ`.

[Pap77] Christos H. Papadimitriou. “The Euclidean Traveling Salesman Problem Is NP-Complete”. In: *Theoretical Computer Science* 4.3, pp. 237–244. 1977.

[PV84] Christos H. Papadimitriou and Umesh V. Vazirani. “On Two Geometric Problems Related to the Traveling Salesman Problem”. In: *Journal of Algorithms* 5.2, pp. 231–246. 1984.

[PY81] Christos H. Papadimitriou and Mihalis Yannakakis. “The Clique Problem for Planar Graphs”. In: *Information Processing Letters* 13.4, pp. 131–133. 1981.

[Pol17] Valentin Polishchuk. *Personal Communication.* 2017.

[PS85] Franco P. Preparata and Michael Shamos. *Computational Geometry: An Introduction.* Monographs in Computer Science. New York: Springer-Verlag. 1985.

[QW06] Jianbo Qian and Cao An Wang. “Progress on Maximum Weight Triangulation”. In: *Computational Geometry* 33.3, pp. 99–105. 2006.

[RS09] Jan Remy and Angelika Steger. “A Quasi-Polynomial Time Approximation Scheme for Minimum Weight Triangulation”. In: *Journal of the ACM* 56.3, 15:1–15:47. 2009.

[RW11] Liam Roditty and Virginia Vassilevska Williams. “Minimum Weight Cycles and Triangles: Equivalences and Algorithms”. In: *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. FOCS ’11. USA: IEEE Computer Society, pp. 180–189. 2011.

[Sch79] Arnold Schönhage. “On the Power of Random Access Machines”. In: *Automata, Languages and Programming*. Ed. by Hermann A. Maurer. Red. by G. Goos, J. Hartmanis, P. Brinch Hansen, D. Gries, C. Moler, G. Seegmüller, J. Stoer, and N. Wirth. Vol. 71. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 520–529. 1979.

[SS05] Raimund Seidel and Micha Sharir. “Top-Down Analysis of Path Compression”. In: *SIAM Journal on Computing* 34.3, pp. 515–525. 2005.

[Sei16] Paul Seiferth. “Disk Intersection Graphs: Models, Data Structures, and Algorithms”. PhD Thesis. Freie Universität Berlin. 2016.

[Sha85] Micha Sharir. “Intersection and Closest-Pair Problems for a Set of Planar Discs”. In: *SIAM Journal on Computing* 14.2, pp. 448–468. 1985.

[ST83] Daniel D. Sleator and Robert Endre Tarjan. “A Data Structure for Dynamic Trees”. In: *Journal of Computer and System Sciences* 26.3, pp. 362–391. 1983.

[Syl78] J. J. Sylvester. “Chemistry and Algebra”. In: *Nature* 17.432 (432), pp. 284–284. 1878.

[vEmd91] Peter van Emde Boas. “Machine Models and Simulations”. In: *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*. Cambridge, MA, USA: MIT Press, pp. 1–66. 1991.

[WX20] Haitao Wang and Jie Xue. “Near-Optimal Algorithms for Shortest Paths in Weighted Unit-Disk Graphs”. In: *Discrete & Computational Geometry* 64.4, pp. 1141–1166. 2020.

[War23] H. C. von Warnsdorf. *Des Rösselsprunges Einfachste Und Allgemeinste Lösung /*. 1823.

[Wel92] Emo Welzl. “On Spanning Trees with Low Crossing Numbers”. In: *Data Structures and Efficient Algorithms*. Vol. 594. Lecture Notes in Comput. Sci. Springer, Berlin, pp. 233–249. 1992.

[WL85] Dan E. Willard and George S. Lueker. “Adding Range Restriction Capability to Dynamic Data Structures”. In: *Journal of the ACM* 32.3, pp. 597–617. 1985.

[WW18] Virginia Vassilevska Williams and R. Ryan Williams. “Subcubic Equivalences Between Path, Matrix, and Triangle Problems”. In: *Journal of the ACM* 65.5, 27:1–27:38. 2018.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, Cambridge. 2011.

[Yu15] Huacheng Yu. “An Improved Combinatorial Algorithm for Boolean Matrix Multiplication”. In: *Automata, Languages, and Programming.* Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 1094–1105. 2015.

[Yus11] Raphael Yuster. “A Shortest Cycle for Each Vertex of a Graph”. In: *Information Processing Letters* 111.21-22, pp. 1057–1061. 2011.

Zusammenfassung

Ein *geometrischer Graph* auf einer Menge von Punkten $S \subseteq \mathbb{R}^2$ ist ein Graph mit S als Knotenmenge, dessen Kanten durch Strecken zwischen den Punkten dargestellt werden. Die Punkte können hierbei durch Angabe eines Radius auf Kreisscheiben erweitert werden. Die Kantenmenge wird dabei durch geometrische Eigenschaften von S definiert. In dieser Arbeit werden zwei Klassen von geometrischen Graphen betrachtet: *Spannbäume* und Graphen die auf Kreisscheiben definiert sind. Die zweite Klasse unterteilen wir in Disk Graphen und Transmissionsgraphen. Zwei Knoten in einem *Disk Graphen* sind mit einer Kante verbunden, wenn sich die zugehörigen Kreisschreiben schneiden. In einem *Transmissionsgraphen* sind zwei Knoten s und t verbunden, wenn t in der von s definierten Kreisscheibe liegt. Wir betrachten drei Arten von Problemen auf geometrischen Graphen:

Dreiecke und Taillenweite in Disk Graphen und Transmissionsgraphen Für Transmissionsgraphen beschreiben wir je einen Algorithmus, der ein Dreieck beziehungsweise ein kürzestes Dreieck in $O(n \log n)$ erwarteter Zeit finden kann. Die Länge eines kürzesten gewichteten Kreises eines Disk Graphen, kann mit gleichem Zeitaufwand gefunden werden. Für Kreise mit maximal k Kanten in Transmissionsgraphen zeigen wir, dass diese in $O(n \log n) + n \cdot 2^{O(k)}$ erwarteter Zeit gefunden werden können. Für alle Ergebnisse in Transmissionsgraphen entwickeln wir Datenstrukturen für gesammelte Bereichsabfragen, die von unabhängigem Interesse sind.

Dynamische Zusammenhangsabfragen in Disk Graphen Wir beschreiben verschiedene Datenstrukturen für Disk Graphen, die es erlauben einzelne Knoten zu löschen und Anfragen, ob zwei Knoten in der gleichen Zusammenhangskomponente liegen zu beantworten. Für die Situation in der alle Punkte einen Radius im Intervall $[1, \Psi]$ haben, beschreiben wir eine Datenstruktur mit $O\left(\frac{\log n}{\log \log n}\right)$ amorisierter Anfragezeit, welche m Löschvorgänge in insgesamt $O\left(\left(n \log^5 n + m \log^9 n\right) \lambda_6(\log n) + n \log \Psi \log^4 n\right)$ erwarteter Zeit bearbeiten kann. Wenn der maximal erlaubte Radius nicht beschränkt ist, kann die Datenstrukur so erweitert werden, dass m Löschvorgänge in $O\left(\left(n \log^6 n + m \log^{10} n\right) \lambda_6(\log n)\right)$ erwarteter Laufzeit durchgeführt werden können, ohne das sich die Anfragezeit verändert.

Lange planare Spannbäume Wir betrachten zudem noch das Problem einen planaren Spannbaum mit maximaler Gesamtlänge zu finden. In diesem Zusammenhang beschreiben wir einen Approximationsalgorithmus, der einen Approximationsfaktor von 0.5467 erreicht. Zudem geben wir eine obere Schranke des Approximationsfaktors von $\frac{5}{6}$ an, der von einem Baum mit Graphdurchmesser drei erreicht werden kann. Wir betrachten auch genaue Algorithmen für Sonderfälle. Mithilfe des Paradigmas der dynamischen Programmierung beschreiben wir einen Algorithmus, der in polynomieller Zeit den längsten Spannbaum mit Graphdurchmesser maximal drei findet. Dieser wird auf spezielle Bäume mit Graphdurchmesser maximal vier erweitert.