Master's Thesis

ROUTING SCHEMES FOR DISK GRAPHS AND POLYGONS

Max Willert

E-Mail address:	max.willert@fu-berlin.de
Target degree:	Master of Education
1st subject:	Computer Science
2nd subject:	Mathematics
1st advisor:	Prof. Dr. Wolfgang Mulzer
2nd advisor:	Prof. Dr. Günter Rote
Address:	Department of Computer Science Freie Universität Berlin Takustr. 9 14195 Berlin

Berlin, August 8, 2016

Statement of authorship

I declare that this document and the accompanying code has been composed by myself, and describes my own work, unless otherwise acknowledged in the text. It has not been accepted in any previous application for a degree. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Max Willert

(Berlin, August 8, 2016)

Acknowledgements

After six years studying computer science, mathematics and pedagogical (content) knowledge, this thesis shall finish this important period of my life. There are many people who helped me to conclude this thesis and I would like to thank them.

First of all, there is Prof. Dr. Wolfgang Mulzer, who was a brilliant supervisor, because he gave the right ideas and the best amount of freedom for solving the theoretical problems. Furthermore, I would like to thank him for the confidence he gave me during my time as teaching assistant.

Additionally, I would like to thank the group of the theoretical computer scientists, especially Yannik, Paul and Boris for their ideas and discussions about my topic. Moreover, I would like to give thanks to Dr. Frank Hoffmann and Dr. Klaus Kriegel for their great support of my preparations for the SoCG last year. Without their help and persuasion I would not have given that kind of talk about my bachelor thesis. Furthermore, I would like to thank for the financial support.

Of course, there are many people reviewing my thesis. Without these people my thesis would not have been as good as it is now.

Not forgetting my whole family, I would like to thank them for the financial and emotional support.

Last but not least, I would like to give some words to the students, especially the bioinformaticians, of the lectures I supervised over the last five years. I never understood why four symbols (A, C, G and T) are much more interesting than two symbols (0 and 1), but I am very thankful that you helped me to improve my teaching skills. I am really looking forward to (im-)prove these skills in other lectures.

Abstract

Routing in networks is a fundamental problem that occurred in the 1980's and was well studied since then. However, there are still open problems, which have not been solved until now. In this thesis we propose routing schemes for special graph classes.

Let $G = (V, E)$ be a weighted network or graph. For any two nodes $p, q \in V$ I would like to be able to route a data package from p to q. A routing scheme $\mathcal{R}$ assigns to each node $p \in V$ a *label* $l(p) \in \{0,1\}^*$ and a *routing table* $\rho(p) \in \{0,1\}^*$. The label identifies the node in the network and the routing table is its own local *read-only* memory. Now the scheme works in the following way: the scheme starts with a *current node* p, the label of a *target node* and some additional information in the data package, called *header*. As a next step, the scheme computes a new node in the network, where the data package is forwarded to. It can use the information in the header and the local memory. The resulting sequence of nodes is the *routing path*. The stretch of $\mathcal{R}$ is the maximum ratio of the Euclidean length of the routing path and the shortest path.

Usually, wireless networks are modelled using (unit) disk graphs. A *disk graph* $\mathrm{DG}(S)$ consists of a set S of n disks with certain radii. The centres of the disks correspond to transmitters of the wireless network. We say that two transmitters are connected, iff their disks intersect. From a practical point of view it is reasonable to say that the scope of the transmitters does not differ too much. Now for any given $\epsilon > 0$, we show how to find a routing scheme for disk graphs with n sites, constant bounded radius ratio and diameter D. This routing scheme has stretch $1 + \epsilon$ and uses $\mathcal{O}(\log n)$ bits for the labels. The header size is $\mathcal{O}(\log n \log D)$ and the routing tables need $\mathcal{O}(\epsilon^{-6} \log^2 n \log^2 D)$ bits. The preprocessing time is $\mathcal{O}(n^2 \log n + \epsilon^{-4} n^2 + \epsilon^{-6} n \log n \log^2 D)$.

Travelling in polygons is a well-known problem as well. The *visibility graph* $\mathrm{VG}(P)$ of a polygon P with n vertices and h holes is a graph in which two vertices in P are connected, iff they can see each other, meaning that the line segment between the two vertices is contained in P. We present the first routing scheme for polygons with holes. For any $\epsilon > 0$ the routing scheme provides the stretch $1 + \epsilon$ and uses no additional information in the header of a data package. The labels have $\mathcal{O}(\log n)$ bits and the corresponding routing tables are of size $\mathcal{O}(\epsilon^{-1} h \log n)$. The preprocessing time is $\mathcal{O}(n^3 + nh\epsilon^{-1})$ and can be improved for simple polygons to $\mathcal{O}(n^2 + n\epsilon^{-1})$.

Contents

List of Figures

List of Algorithms

CHAPTER 1
Introduction

Routing in graphs is a crucial problem in distributed graph algorithms, see for example [16] and [30]. Given a graph G we would like to route a data package from one location to another. There are two different properties, a routing scheme has to satisfy. First, the routing should be local. That means, it uses only information stored in the data package and the preprocessed memory of a current node of the graph. Second, the routing scheme should be efficient, meaning that the data package does not travel considerably too long. There is an obvious way to solve this problem: for each node p of G, we store the complete shortest path tree of p in its memory. Thus, we can route along the shortest path from one node to another. Unfortunately, this method is rather inefficient, because we have to store the entire topology of G at each node. Furthermore, both goals are conflicting properties. On the one hand, the routing scheme should yield paths that are as short as possible, but on the other hand, we would like to store a minimum routing information in the processors' local memory.

Thorup and Zwick introduced the notion of a *distance oracle* [39]. Given a graph G, the goal is to produce a compact data structure, named distance oracle that can quickly answer distance queries between any pair of nodes in the graph. Thus, routing schemes are distributed implementations of distance oracles [32]. The graph corresponds to the distributed network. A distributed algorithm runs on the processors – the nodes of the graph. Each node has a memory and a data package that is sent from a source to a target contains the name of the destination and perhaps some additional information for the routing scheme. If a node receives a message, it checks, whether it is the message's destination. Otherwise, it uses the additional information and its own routing table to choose the link on which it forwards the message. The stretch of a routing scheme is the worst possible ratio between the travelled distance and the shortest path distance.

For general graphs, there are many results available, because the problem was well-studied since the 1980's (see for instance [1, 4, 9, 11, 12, 31, 32]). Recently, Roditty and Tov developed a routing scheme for a graph G with n vertices and m edges. The scheme provides a poly-logarithmic header size and routes a message from p to q on a path with length $\mathcal{O}(k\Delta + m^{1/k})$, where Δ is the distance of the shortest path between p and q and $k > 2$ an integer. Their routing tables use $mn^{\mathcal{O}(1/\sqrt{\log n})}$ total space [32]. However, for general graphs, any efficient routing scheme needs to store $\Omega(n^c)$ bits per node, for some $c > 0$ (see [30]). Thus, it is useful to ask whether there are improved results for specialized graph classes. For instance, there are routing schemes for trees that follow the shortest path and require at most $\mathcal{O}(\log n)$ bits at each node [14, 33, 38]. Additionally, in planar graphs it is possible for any $\epsilon > 0$ to find a routing scheme with a poly-logarithmic number of bits in the routing tables and a stretch $1 + \epsilon$ [37].

1.1 Disk Graphs

The first graph class that is of particular interest for routing problems is given by *disk graphs*. The nodes are represented by points in the plane, called *sites*. Furthermore, each site corresponds to a transmitter in a wireless network and has a certain perimeter, in which a point can receive and send messages. Traditionally, most networks are modelled as *unit disk graphs* [16], in which two sites share an edge if and only if their distance is at most 1. Unit disk graphs maybe dense, but they share many properties with planar graphs. The first scheme for this graph class was given by Yan, Xiang and Dragan [41]. They extend the ideas of Gupta et al. [19] to obtain a routing scheme with routing table size $\mathcal{O}(\log^2 n)$ and constant stretch. Recently Kaplan et al. describe a routing scheme that provides for any $\epsilon > 0$ the stretch $1 + \epsilon$, using a header with poly-logarithmic size [21]. In this thesis, we extend this last routing scheme to disk graphs with constant bounded radius ratio, meaning that the radii of the various disks differ by not too much. Unlike the algorithm of Yan, Xiang and Dragan, we need a header of poly-logarithmic size. It would be interesting to evaluate, whether one can remove this header.

For the description of this routing scheme we need a geometric data structure – the *well-separated pair decomposition*. This decomposition was introduced by Callahan and Kosaraju [8] and extended for unit disk graphs by Gao and Zhang [15]. A pair of point sets (A, B) is c-well-separated, if the distance between A and B is at least c times the diameters of both A and B. Here we can use an arbitrary definition of distance – for example the Euclidean metric or the disk graph metric. A well-separated pair decomposition of a point set consists of a set of well-separated pairs that cover all pairs of distinct points. Gao and Zhang already mentioned that it is possible to extend their result for disk graphs with constant radius ratio. We will describe the details.

1.2 Polygons

The second graph class of interest is the visibility graph of a polygon with an arbitrary number of holes. We say that two vertices in a polygon P with h holes and n vertices are connected by an edge if they can see each other that is the line segment between the two vertices is contained in P. The problem of computing a shortest path between two vertices has been well studied in computational geometry, see for instance [3, 5, 18, 20, 22, 23, 25, 26, 29, 34, 35, 40]. Nevertheless, to the best of our knowledge there are no routing schemes for visibility graphs of polygons. Thus, we present the first routing scheme for polygons. For any $\epsilon > 0$ the routing scheme needs at most $\mathcal{O}(\epsilon^{-1} h \log n)$ bits in the routing table and produces a routing path with stretch $(1 + \epsilon)$.

1.3 Structure of this Thesis

First, we introduce relevant basic definitions and prove lemmas in the next chapter which will be needed for our main results. The following two chapters describe how to produce the routing scheme for disk graphs. In Chapter 3 we compute the details for the well-separated pair decomposition and apply them in Chapter 4 to find the routing scheme. After that, we describe the first routing scheme for polygons in Chapter 5. Last but not least, we discuss open problems and final issues in Chapter 6.

CHAPTER 2
Preliminaries

In this chapter we define the model for our routing problem. In the first section we look at graphs in general, whereas the following sections provide relevant basis definitions for routing schemes, disk graphs and polygons.

2.1 Basic Definitions

Let $G = (V, E)$ be an *undirected, connected graph* without multi-edges or loops. In our model this graph G is always embedded in the Euclidean plane $\mathbb{R}^2$: a *node* $p = (p_x, p_y) \in V$ corresponds to a point and an edge $\{p, q\} \in E$ represents the line segment $\overline{pq}$. Since we have the Euclidean plane, we denote by $|\overline{pq}|$ the Euclidean distance between the points p and q that means $|\overline{pq}| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$. Furthermore, we define $|\overline{pq}|$ to be the *weight* of the corresponding edge. The *degree* of $p \in V$, called $\deg_G(p)$, is the number of incident edges in G. We omit the index G, if the graph is known from the context.

Definition 2.1 (Shortest Path)**.** *Let $G = (V, E)$ be a graph as described above and $p, q \in V$. We say that $\pi = p_0 p_1 ... p_k$ is a path between p and q, if $p_0 = p$, $p_k = q$ and $\{p_{i-1}, p_i\} \in E$ (for all $1 \leq i \leq k$). The length of π is defined as $d(\pi) = \sum_{i=1}^{k} |\overline{p_{i-1}p_i}|$. Moreover, we define $d(p, q) = \min_\pi d(\pi)$ to be the length of a shortest path connecting p and q in G, where π goes over all paths with endpoints p and q.*

The definition of the shortest path can be extended to pairs of sets. Let V_1 and V_2 be two subsets of V. The distance between these two sets is defined as $d(V_1, V_2) = \min_{p_1 \in V_1, p_2 \in V_2} d(p_1, p_2)$. Additionally, $\operatorname{diam}(V_1) = \max_{p_1, p_2 \in V_1} d(p_1, p_2)$ denotes the *diameter* of V_1.

2.2 Routing Schemes

Let $G = (V, E)$ be a graph as described above. Now we define a *routing scheme* for G. There are several different definitions for routing schemes, see [21, 32, 41]. Most commonly, a routing scheme has a ”behaviour”, meaning that a physical node of the scheme forwards data packages to other physical nodes using certain protocols. This behaviour is called *routing function*. Moreover, each physical node has a name tag that identifies the physical node in the network and is available to the routing function. These name tags are called *labels*. They usually are bit strings. Last but not least, each node has to use certain information during its routing function. This information is stored in the memory of the physical node or additionally in the forwarding data package itself.

The memory of the node is called *routing table*, and the information stored in the data package which is necessary for the routing is called *header*. There are routing schemes that do not use any headers, see for instance [41] or Chapter 5.

Definition 2.2 (Routing scheme)**.** *A routing scheme $\mathcal{R} = (l, \rho, f)$ of a graph G consists of the following elements:*

- *a label $l(p) \in \{0,1\}^*$ for each node $p \in V$,*
- *a routing table $\rho(p) \in \{0,1\}^*$ for each node $p \in V$ and*
- *a routing function $f: V \times \{0,1\}^* \times \{0,1\}^* \to V \times \{0,1\}^* \times \{0,1\}^*$.*

The routing function takes as input a node $p \in V$, the label $l(q)$ of a target node q and a header $h \in \{0,1\}^*$. Now the routing scheme works as follows: starting at a point $p \in V$ we can use some information to route an arbitrary data package. This information is composed of three parts. The first part is the routing table of p. This routing table usually contains some information about the topology of the graph. The second part is the label of a target site, perhaps an intermediate target. This label is useful from a practical point of view, because labels can be stored in routing tables of any other node. Thus, it represents the name of a node but also additive information. Sometimes we need additional information from the past during the routing of data from one location to another. This information is stored in the header. The header enables us to use recursive routing schemes. Figure 2.1 shows the conceptual idea behind that mathematical definition.

Now using the three pieces of information the routing function will compute new information. The first part of the output is a node p' that has to be adjacent to p. The data now has to travel from p to p'. Furthermore, it is possible to compute a new target label (second part of the output) and a new header (third part of the output). This can be useful for the mentioned recursive routing schemes.

Definition 2.3 (Properties of a routing scheme)**.** *Let $\mathcal{R} = (l, \rho, f)$ be a routing scheme of a graph G. We say that $\mathcal{R}$ is correct if the following holds: for any two sites $p, q \in V$ consider the sequence of triples given by $(p_0, l_0, h_0) = (p, l(q), \epsilon)$ and $(p_i, l_i, h_i) = f(p_{i-1}, l_{i-1}, h_{i-1})$ for $i \geq 1$. Then there exists a $k = k(p,q) \geq 0$ such that $p_k = q$ and $p_i \neq q$ for $i < k$.*
The routing scheme reaches q after k steps. Furthermore, $\pi = p_0 p_1 ... p_k$ is the routing path between p and q. The routing distance is $d_\rho(p,q) = d(\pi)$. If $\mathcal{R}$ does not use any headers that means $h_i = \epsilon$ for all $i \geq 0$, then we write $f(p,l)$ and mean $f(p,l,\epsilon)$.

Now let $\mathcal{R}$ be a correct routing scheme. We would like to know how efficient a routing scheme is. To measure this efficiency we have to define some useful items. First of all, each site in V gets a label. The *label size* L of $\mathcal{R}$ is the size of a largest label among all nodes. This size should be as small as possible. The label size is defined as

$$L(n) = \max_{|V|=n} \max_{p \in V} |l(p)|.$$

Moreover, we would like to have small routing tables. If the routing table is as small as possible, the routing process of a data package should be faster. The item for measuring the *size of the*

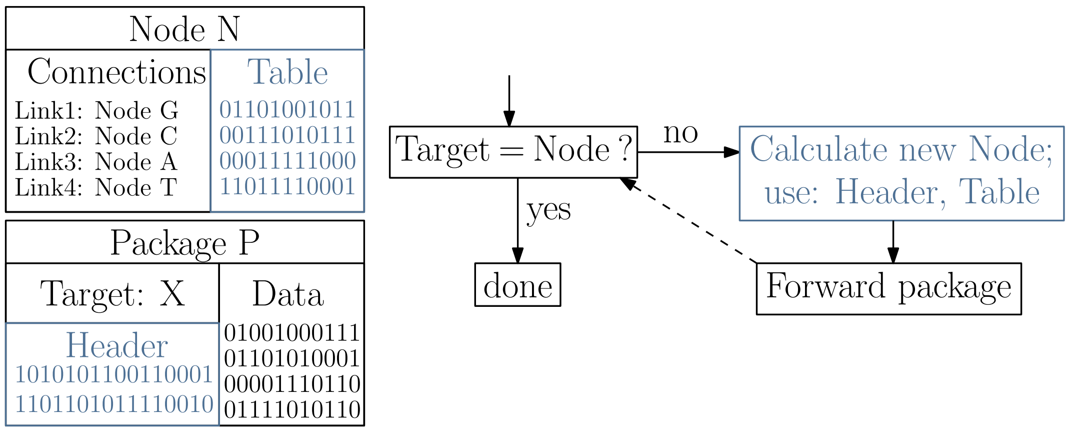

Figure 2.1: *The conceptual idea of a routing scheme. A node N receives a data package P. Unless N is the target of P, N can use its routing table and the package's header to compute the link to which P is forwarded to.*

routing tables is T and is defined as

$$T(n) = \max_{|V|=n} \max_{p \in V} |\rho(p)|.$$

If the routing scheme uses a header, the size of this header should be as small as possible, because it belongs to the data travelling through the network. The less bits this header contains the faster the transportation is realized. The corresponding item for measuring the *size of the header* is H and is defined as

$$H(n) = \max_{|V|=n} \max_{p \neq q \in V} \max_{i=1\ldots k(p,q)} |h_i|.$$

Last but not least, we would like to know more about the distance of the routed data package in relation to an optimal path. It is possible that the routing scheme computes a path that is not a shortest path. The *stretch* ζ tells us, how large the travelled path is in relation to the shortest path:

$$\zeta(n) = \max_{|V|=n} \max_{p \neq q \in V} \frac{d_\rho(p,q)}{d(p,q)}.$$

We have defined correct routing schemes and reasonable measures for general graphs. In this thesis, we discuss two special graph classes: disk graphs and visibility graphs of polygons with holes. In the following two sections, we define them and provide first simple properties of these graph classes.

2.3 Disk Graphs

Given a point $p \in \mathbb{R}^2$ and a real number $r > 0$, we denote by $D_r(p) = \{q \in \mathbb{R}^2 \mid |\overline{pq}| \leq r\}$ the *closed disk* with *center* p and *radius* r. Now let $\mathcal{S}$ be a set of n disks in the plane. The set $\mathcal{S}$ can be represented as a set S of n *sites* and a function $r\colon S \to \mathbb{R}^+$, which assigns to each center $p \in S$

the radius $r(p)$, such that $D_{r(p)}(p)$ is in $\mathcal{S}$. For technical reasons we introduce some important conventions on the notations. The conventions hold for disk graphs in the entire thesis. First of all, if we refer to a set S of n sites, we always assume without mentioning it that there is an additional function r, which assigns to each point p the radius r of the corresponding disk. That means if we talk about a set S of n points then there is a set of n disks whose centres are the points in S. Furthermore, we set $r(p) = r_p$ or $r(p_i) = r_i$ and $D_{r(p)}(p) = D(p)$. For convenience, we write $D^c_r(p)$ instead of $D_{c\cdot r}(p)$ for each $c > 0$ and $D^c(p)$ instead of $D_{c\cdot r_p}(p)$, respectively.

Definition 2.4 (Disk graph)**.** *Let $S \subset \mathbb{R}^2$ be a set of n sites.* $\mathrm{DG}(S) = (S, E)$ *is called the disk graph of S, in which two distinct sites $p, q \in S$ share an edge $e \in E$, iff $|\overline{pq}| \leq r_p + r_q$.*

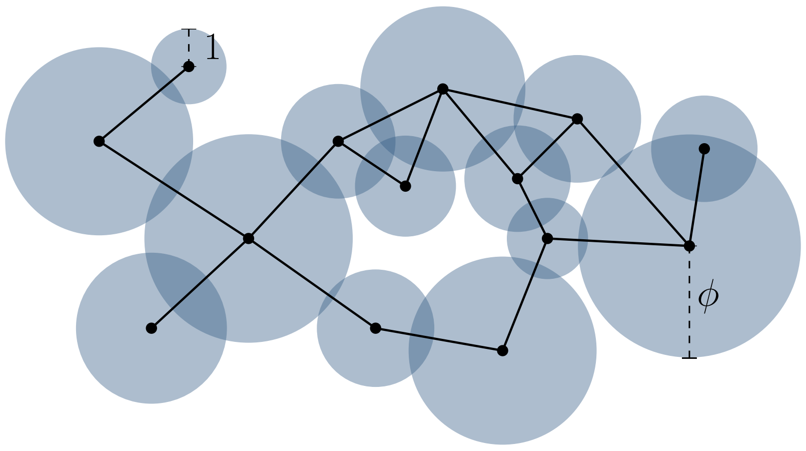

Figure 2.2: *A disk graph with smallest radius* 1 *and largest radius* ϕ*.*

In other words, we can say that there is an edge between $p, q \in S$ in the disk graph if and only if their corresponding disks intersect (see Figure 2.2). Additionally, we define the *radius ratio* of S to be $\phi = \max_{p,q\in S}(\frac{r_p}{r_q})$. In our model we will assume that this ratio does not depend on the size of S, i.e. $\phi \in \mathcal{O}(1)$. This assumption seems to be reasonable from a practical point of view, because the scope of a transmitter does usually not depend on the number of transmitters. Furthermore, we assume without loss of generality that the smallest radius equals 1, because otherwise we would scale the whole input by a suitable factor, such that the smallest radius is 1. Thus we can assume that all the radii of our disk graph are at most the radius ratio ϕ. We say that S has *density* δ if every unit disk (disk with radius 1) contains at most δ sites. The first observations will be useful for further proofs.

Observation 2.1. *Let* $\mathrm{DG}(S)$ *be a disk graph with n sites and maximum radius ϕ. Then we have* $\mathrm{diam}(S) \leq 2\phi(n-1)$.

Proof. Consider two sites $p, q \in S$ with $d(p, q) > 2\phi(n-1)$. On a shortest path from p to q there have to be at least two adjacent sites $s, t \in S$ with Euclidean distance more than 2ϕ. This contradicts the assumption that ϕ is the maximum radius and s and t are adjacent in $\mathrm{DG}(S)$. □

Observation 2.2. *Let $S_1, S_2 \subseteq S$ be two subsets of sites such that* $\mathrm{DG}(S_1)$ *and* $\mathrm{DG}(S_2)$ *are connected, $p, s \in S_1$ and $q, t \in S_2$. Then we have* $\mathrm{diam}(S_1) + d(s, t) + \mathrm{diam}(S_2) \geq d(p, q)$.

Proof. This claim holds by the triangle inequality and the fact that $\operatorname{diam}(S_1) \geq d(p,s)$ and $\operatorname{diam}(S_2) \geq d(t,q)$. □

The following lemma will immediately imply that the largest degree in the minimum spanning tree is in $\mathcal{O}(\phi^2)$. It is easy to show this asymptotic upper bound. But since we need the maximum degree in a later analysis, we have to know something about the constant. Therefore, the proof is a bit more involved.

Lemma 2.3. *Let* $\mathrm{DG}(S)$ *be a disk graph and* $T = (S,E)$ *a minimum spanning tree of* $\mathrm{DG}(S)$. *Then we have for all* $p \in S$: $\deg_T(p) \leq 26\phi^2$.

Proof. Let p be a site in the minimum spanning tree and $N(p)$ be the set of all sites in T that are adjacent to p. Without loss of generality p is the origin.

Consider the disk $D_2(p)$. Then we have $|D_2(p) \cap N(p)| \leq 6$. To prove this, we assume the opposite. Then there has to be a cone C with apex p and apex angle $60°$ such that two points $r, q \in N(p)$ are located in the interior of C. Without loss of generality we have $|\overline{pq}| \leq |\overline{pr}|$. Thus, we can derive

$$|\overline{rq}| = \frac{\sin\alpha}{\sin\beta}|\overline{pr}| < \frac{\sin 60°}{\sin 60°}|\overline{pr}| = |\overline{pr}|,$$

where $\alpha < 60°$ is the angle at p and $\beta > 60°$ is the angle at q in the triangle $\Delta(p,q,r)$. This contradicts the assumption that T is a minimum spanning tree, because we can delete the edge $\{p,r\}$ and and since the minimum radius is 1 we can add the edge $\{r,q\}$ to obtain a new spanning tree with less weight.

As a next step, we look at the grid defined by the horizontal lines $h_i\colon y_i = i \cdot \sqrt{2}$ and the vertical lines $v_i\colon x_i = i \cdot \sqrt{2}$ (for each $i \in \mathbb{Z}$). Then each square in the grid has diameter 2. We observe that $N(p)$ is contained in a square Q with centre p (intersection of the diagonals) that contains at most $4\lceil\sqrt{2}\phi\rceil^2$ small grid squares. Moreover, we define $D_{i,j}$ to be the perimeter of the square that is defined by the lines h_i, h_{i+1}, v_j and v_{j+1}. Thus, we need at most $4\lceil\sqrt{2}\phi\rceil^2$ disks $D_{i,j}$ to cover $N(p)$. We will apply the same argument as above and show that $D_{i,j} \cap N(p)$ can have at most one site (with some exceptions). Again, we assume the opposite and derive a contradiction. Let m be the midpoint of $D_{i,j}$ and $r \neq q \in D_{i,j} \cap N(p)$. We have to discuss three cases.

Case 1: In the first case we have $|\overline{pm}| \geq 3$. Then we have $|\overline{pq}|, |\overline{pr}| \geq 2$ and since $q \neq r$ we have either $|\overline{pq}| > 2$ or $|\overline{pr}| > 2$. But since $|\overline{qr}| \leq 2$, we can apply the same argument as above to obtain a new spanning tree with less weight.

Case 2: In the second case we have $2 < |\overline{pm}| < 3$. If either q satisfies $|\overline{pq}| > 2$ or r satisfies $|\overline{pr}| > 2$, we have the same situation as in Case 1. Otherwise q and r are contained in $D_2(p)$. Since $|\overline{pm}| > 2$, q and r lie within a cone C with apex p and apex angle $< 60°$ and we can derive the contradiction again.

Case 3: In the third case, we only have to look at the four disks $D_{0,0}$, $D_{0,-1}$, $D_{1,1}$ and $D_{-1,0}$. It is possible that these disks contain more than one site in the neighbourhood of p. However, we notice that these four disks are contained in $D_2(p)$.

Finally, we have the following situation. $D_2(p)$ and $4\lceil\sqrt{2}\phi\rceil^2 - 4$ disks with radius 1 cover the entire neighbourhood of p. We have shown that $D_2(p)$ contains at most 6 sites and all the other disks contain at most 1 site of $N(p)$. Hence, we have

$$\deg_T(p) \leq 6 + 4\lceil\sqrt{2}\phi\rceil^2 - 4 \leq 4(\sqrt{2}\phi + 1)^2 + 2 \leq 26\phi^2.$$

□

Lemma 2.4. *Let $T = (S, E)$ be a tree with n vertices and maximum degree α. Then there exists an edge e, such that $T \setminus e$ consists of two trees with at least $n/(\alpha+1)$ vertices each.*

Proof. We show that there is an edge e such that $T \setminus e$ consists of two trees with at least $\lceil (n-1)/\alpha \rceil$ vertices each. The claim then follows from the fact that $n \geq \alpha + 1$.

Let v be an arbitrary vertex of T. By the pigeonhole principle the largest subtree T_u of v rooted at u has at least $\lceil (n-1)/\alpha \rceil$ vertices. Now let e be the edge between u and v. There are two cases. In the first case $T \setminus e$ satisfies our lemma and we are done. Otherwise, the subtree of u rooted at v has less than $\lceil (n-1)/\alpha \rceil$ vertices. But then - again by the pigeonhole principle - the largest subtree T_w of u rooted at $w \neq v$ has at least $\lceil (n-1)/\alpha \rceil$ vertices. Again, there are two cases and in fact the process starts from the beginning. The process has to stop after at most $\lceil (n-1)/\alpha \rceil$ steps, because in each step we gain at least one vertex and there are no cycles in the tree. □

Lemma 2.5. *Let $S \subset \mathbb{R}^2$ be a point set with density δ and $p \in \mathbb{R}^2$. Then the disk $D_r(p)$ with radius $r \geq 1$ contains at most $\mathcal{O}(r^2\delta)$ points of S.*

Proof. This proof works by a simple volume argument. We can use $\mathcal{O}(r^2)$ different unit disks to cover the entire disk $D_r(p)$. Since S has density δ each unit disk contains at most δ points of S. In fact, we have $\mathcal{O}(r^2\delta)$ points in $D_r(p) \cap S$. □

2.4 Polygons

For our second application we need definitions for polygons. First of all, a *polygonal chain* $\pi = \overline{p_0p_1}, \overline{p_1p_2}, \dots, \overline{p_{n-1}p_n}$ is a sequence of line segments. For convenience, we identify the polygonal chain with its vertices p_i, i.e. $\pi = p_0p_1 \dots p_n$. We say that the polygonal chain is *closed*, if $p_0 = p_n$. Furthermore, if any two non-adjacent line segments do not intersect and adjacent line segments intersect only in one of their endpoints, the polygonal chain is called *simple*. It is a well-known fact that each simple, closed polygonal chain π separates the surface into two distinct sets: the bounded interior of π – noted by $\operatorname{int} \pi$ – and the unbounded exterior.

As a next step, we define a polygon that has a certain number of holes. A *hole* is a connected region in the interior of a polygon, but it does not belong to the polygon itself. Thus, holes have to satisfy three conditions: a hole is contained in the interior of the polygon (i) and two distinct holes do not intersect each other (ii and iii). The following definition states these conditions more precisely.

Definition 2.5 (Polygon). *Let $h \in \mathbb{N}$ and $\pi_i = p_{i,0} \dots p_{i,n_i}$ be $h+1$ different simple and closed polygonal chains for $0 \leq i \leq h$, such that*

(i) $\pi_i \subset \operatorname{int} \pi_0$ *for all* $1 \leq i \leq h$,

(ii) $\pi_i \cap \pi_j = \varnothing$ *for* $1 \leq i \neq j \leq h$ *and*

(iii) $\pi_i \notin \operatorname{int} \pi_j$ *for* $1 \leq i \neq j \leq h$.

Then P is called a polygon with h holes and is defined as $P := (\pi_0 \cup \operatorname{int} \pi_0) \setminus \left(\bigcup_{i=1}^{h} \operatorname{int} \pi_i\right)$.

Thus, the *boundary* of P has $h+1$ components: one outer boundary and h hole boundaries. The boundary of P is the union of the $h+1$ polygonal chains, meaning $\partial P = \bigcup_{i=0}^{h} \pi_i$. Moreover, we set $\operatorname{int} P := P \setminus \partial P$ and $|P| := \sum_{i=0}^{h}(n_i + 1)$. For practical reasons, we introduce further convention. We define for each $i = 0 \ldots h$ and each $j \in \mathbb{Z}$ the point $p_{i,j}$ to be $p_{i,k}$ with $k = j \mod (n_i + 1)$. Moreover, if the context is clear, we omit the first index of the $p_{i,j}$.

To define graph classes for polygons, we need the concept of visibility.

Definition 2.6 (Visibility)**.** *Let P be a polygon with h holes and p and q two points in P. The two points can see each other iff the line segment between the two points is contained in P, i.e. $p \leftrightsquigarrow q$, iff $\overline{pq} \subset P$. Additionally, the visibility polygon $V(p)$ of a point p is the set of all points that can see p. That means $V(p) = \{q \in P \mid p \leftrightsquigarrow q\}$.*

Finally, we can define the visibility graph for a polygon P, see Figure 2.3.

Definition 2.7 (Visibility graph)**.** *Let P be a polygon with h holes. The visibility graph* $\operatorname{VG}(P)$ *consists of the vertices of P that is $V = \{p_{i,j} \mid 0 \leq i \leq h, 1 \leq j \leq n_i\}$. Furthermore, there is an edge between two points, if and only if they can see each other, meaning $E = \{\{p,q\} \mid p \leftrightsquigarrow q\}$.*

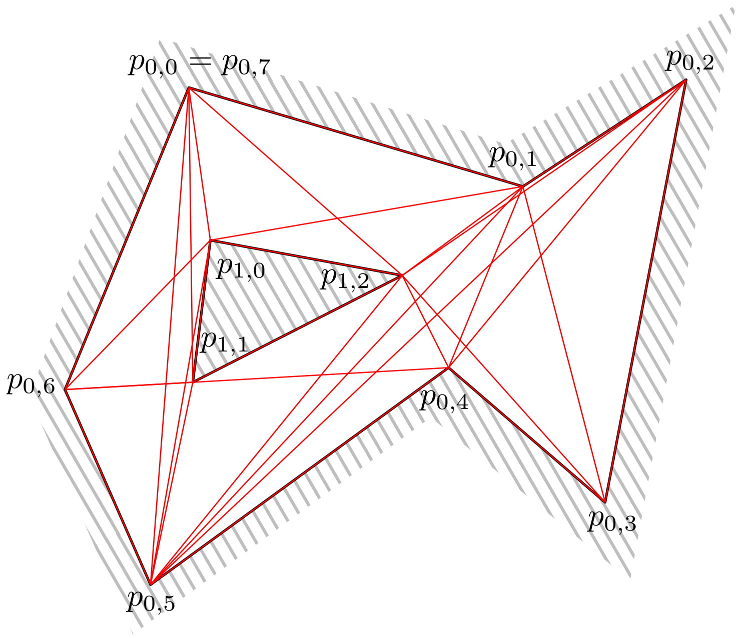

Figure 2.3: *The visibility graph (red) of a polygon with ten points and one hole.*

CHAPTER 3

Well-separated Pair Decomposition

Callahan and Kosaraju introduced the concept of well-separated pair decomposition of a point set [8]. This decomposition consists of well-separated pairs that cover all the pairs of distinct points. Two sets of points in the Euclidean plane are *well-separated*, if the distance between the two point sets is large enough in relation to their diameters (see for instance Figure 3.1). The decomposition depends on the chosen distance metric, since the diameter is defined using this metric. Callahan and Kosaraju have shown how to compute a well-separated pair decomposition of a point set using the Euclidean distance [8]. Although there are $\binom{n}{2}$ pairs of distinct points, we can always find a well-separated pair decomposition with $\mathcal{O}(n)$ pairs of point sets. The decomposition can be computed in $\mathcal{O}(n \log n)$ time, which is optimal. In the last two decades well-separated pair decomposition have found various applications in solving proximity problems for points in the Euclidean space [2, 6–8, 13, 17, 24, 27, 28]. One well-studied problem is the computation of a spanner. A *t-spanner* H of a graph G is a subgraph, such that for any two points p and q in G, there is a path in H between them, whose length is at most $t \cdot d_G(p, q)$. Narasimhan and Smid gave a nice survey on this topic, especially on using well-separated pair decompositions for computing sparse spanners [28].

However, well-separated pair decompositions are defined for arbitrary distance metrics. In this chapter we will concentrate on the disk graph metric – the shortest path metric of a disk graph $\mathrm{DG}(S)$. The following definition is illustrated by Figure 3.1.

Definition 3.1 (Well-separated pair decomposition)**.** *Let* $\mathrm{DG}(S)$ *be a disk graph with* n *sites and* $c \geq 1$. *Two non-empty subsets* $A, B \subseteq S$ *are called* c*-well-separated if* $d(A, B) \geq c \cdot \max(\mathrm{diam}(A), \mathrm{diam}(B))$.
A set of pairs $\Xi = \{P_1, P_2, ..., P_m\}$, *where* $P_i = (A_i, B_i)$, *is called a pair decomposition of* S *if*

(i) $A_i, B_i \subseteq S$ *for* $1 \leq i \leq m$,

(ii) A_i *and* B_i *are disjoint and*

(iii) for each pair $(a, b) \in S \times S$ *of distinct sites, there is a unique* i *such that* $(a, b) \in A_i \times B_i$.

We say that (A_i, B_i) *represents* (a, b). *If in addition, every pair in* Ξ *is* c*-well-separated,* Ξ *is a* c*-well-separated pair decomposition.*

Gao and Zhang [15] have shown that it is possible to build a well-separated pair decomposition for the unit disk graph metric that has $\mathcal{O}(n \log n)$ pairs. We will use this approach to build a well-separated pair decomposition for the disk graph with constant radius ratio ϕ and density δ.

The remaining part of this chapter will be very similar to the article of Gao and Zhang. We are going to extend their description by using pseudo-code. Nevertheless, the lemmas and theorems will be slightly different, because we have to deal with the additional parameter ϕ.

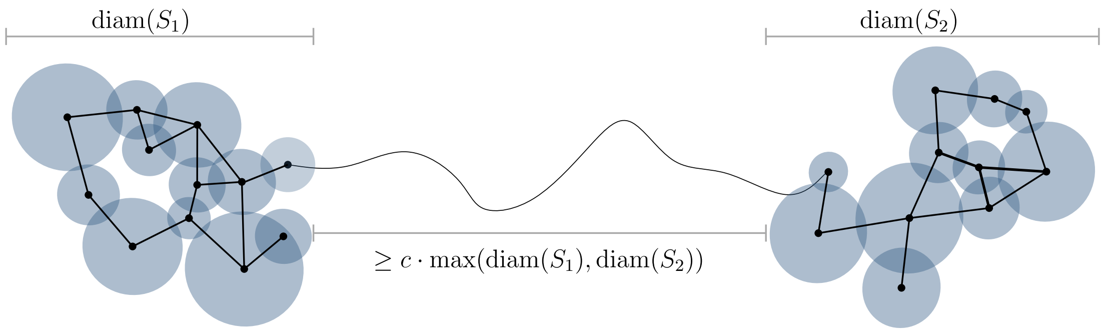
$\operatorname{diam}(S_1)$
$\operatorname{diam}(S_2)$
$\geq c \cdot \max(\operatorname{diam}(S_1), \operatorname{diam}(S_2))$

Figure 3.1: *S_1 and S_2 are c-well-separated.*

3.1 Computing the Well-separated Pair Decomposition

First of all, let S be a set of n sites with radius ratio $\phi \in \mathcal{O}(1)$ and density δ. In the first step we compute the minimum spanning tree T of $\mathrm{DG}(S)$. Since S has constant radius ratio, we know by Lemma 2.3 that the maximum degree α of T is in $\mathcal{O}(1)$. Applying Lemma 2.4, we find an edge e such that deleting this edge in T, we get two disjoint subtrees, with at least $\frac{n}{\alpha+1}$ sites each. Applying this argument recursively, we obtain a so called *hierarchical decomposition* of T (see Figure 3.2). We can represent this decomposition as a binary decomposition tree H, in which an inner node corresponds to the selected edge and a leaf represents a site of T (see Algorithm 3.1). Since α is a constant, this decomposition tree is balanced and has height $\mathcal{O}(\log n)$.

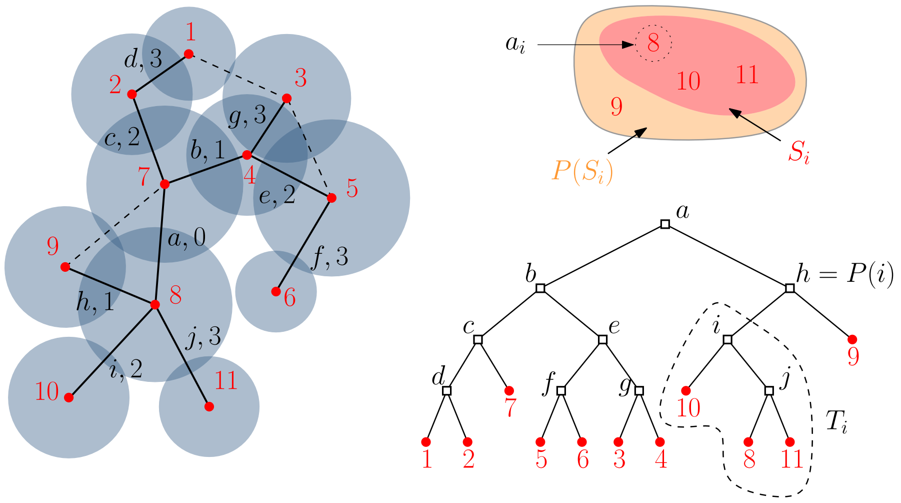

Figure 3.2: *The minimum spanning tree of a disk graph $\mathrm{DG}(S)$ (left) and its hierarchical decomposition (right bottom). The terms S_v, T_v, a_v and $P(S_v)$ are illustrated for the edge i.*

Algorithm 3.1 Computes the hierarchical decomposition for an MST with largest degree α.

```
Input: spanning tree $T_r$ rooted at $r$
Output: hierarchical decomposition $H$ of $T_r$
 1: compute for each site $p$ the number $n_p$ of leafs in the tree $T_p$
 2: if $n_r = 1$ then
 3:     $H.element := r$
 4:     return $H$
 5: end if
 6: $size := 0$; $p := r$
 7: while $size < \lceil (n_r - 1)/\alpha \rceil$ do
 8:     $q := p$
 9:     $p :=$ argmax$\{n_s \mid s$ is a child of $q\}$ /* root of largest subtree of $q$ */
10:     $size := n_r - n_p$ /* size of $T_r \setminus T_p$ */
11: end while
12: $H.element := \{p, q\}$
13: $H.left :=$ hierarchical decomposition of $T_p$
14: $H.right :=$ hierarchical decomposition of $T_q = T_r \setminus T_p$
15: return $H$
```

For the next step of the algorithm and the later analysis we will need some notations. Let v be a node of H. Then v corresponds to a subtree T_v of T with vertex set $S_v \subseteq S$ and a_v represents an arbitrary point in S_v. Furthermore, let v be an inner node and u and w its children, then v has an associated edge $e_v \in T_v$. Removing this edge from T_v yields the two subtrees T_u and T_w. Again by Lemma 2.4 we have $|S_u|, |S_w| \geq |S_v|/(\alpha + 1)$. Moreover, let $P(v)$ be the parent node of a node v ($\neq$ root) and we set $P(S_v) = S_{P(v)}$. In Figure 3.2 we can see a minimum spanning tree, the corresponding hierarchical decomposition and the different notations.

As we can see in Algorithm 3.1, the computation of the hierarchical decomposition does not depend on the separating parameter c. Using this parameter and the decomposition tree H, we can compute the c-well-separated pair decomposition for $\mathrm{DG}(S)$ with an iterative process (see Algorithm 3.2). We discuss the details of this algorithm in the following section.

3.2 Analysis

In this section we bound the number of computed pairs and give the first theorem of this thesis. First of all, we prove that Algorithm 3.2 computes a c-well-separated pair decomposition for $\mathrm{DG}(S)$. Then we bound the number of computed pairs and analyse the running time for our approach.

Let Ξ be the output of Algorithm 3.2 for the hierarchical decomposition H of a disk graph $\mathrm{DG}(S)$.

Lemma 3.1. *Ξ is a c-well-separated pair decomposition of $\mathrm{DG}(S)$ and each pair of different sites (p, q) is covered by exactly one pair in Ξ.*

Algorithm 3.2 Computes the c-well-separated pair decomposition for $\mathrm{DG}(S)$

```
Input: $\mathrm{DG}(S)$, $c \geq 1$
Output: $c$-well-separated pair decomposition $\Xi$
 1: $H$ := hierarchical decomposition of minimum spanning tree of $\mathrm{DG}(S)$
 2: $Q := Queue()$
 3: $\Xi := \varnothing$
 4: $Q.add((S_r, S_r))$ /* $r$ is the root of $H$*/
 5: while $Q \neq \varnothing$ do
 6:    $(S_v, S_w) := Q.remove()$
 7:    if $|\overline{a_v a_w}| > 2\phi(c+2)\max(|S_v|-1, |S_w|-1)$ then
 8:       $\Xi.add((S_v, S_w))$
 9:    else if $|S_v| = |S_w| = 1$ then
10:       $NOP$
11:    else if $|S_v| \geq |S_w|$ then
12:       $Q.add((S_{v_1}, S_w))$; $Q.add((S_{v_2}, S_w))$ /* $v_1$ and $v_2$ are the children of $v$ */
13:    else
14:       $Q.add((S_v, S_{w_1}))$; $Q.add((S_v, S_{w_2}))$ /* $w_1$ and $w_2$ are the children of $w$ */
15:    end if
16: end while
17: return $\Xi$
```

Proof. First, we show that Ξ is a pair decomposition. The process of Algorithm 3.2 obviously ends and we have to argue that all the pairs of distinct sites are covered by a pair in Ξ. Line 9 ensures that pairs of the form $(\{p\}, \{p\})$ are discarded. Furthermore, we know that the pair (S_r, S_r) is the first pair in the queue. This pair covers all the pairs $(a, b) \in S \times S$ with $a \neq b$. Now each time we split a pair in the queue, the union of the covered pairs remains the same. Thus, each ordered pair of distinct sites is covered by a pair in Ξ. Furthermore, we know that the splitting of S_v in S_u and S_w is a splitting in two disjoint sets (see Line 11 to 14). That guarantees that each ordered pair is covered exactly once.

Second, we have to prove that each pair in Ξ is c-well-separated. Let $(S_v, S_w) \in \Xi$. By construction we have $|\overline{a_v a_w}| > 2\phi(c+2)\max(|S_v|-1, |S_w|-1)$. Since S_v is connected by construction, we have $\mathrm{diam}(S_v) \leq 2\phi(|S_v|-1)$ by Observation 2.1. Furthermore, we have $d(p,q) \geq |\overline{pq}|$ and $\mathrm{diam}(S_v) + d(S_v, S_w) + \mathrm{diam}(S_w) \geq d(a_v, a_w)$. Finally, we can estimate by Observation 2.2

$$\begin{aligned}
d(S_v, S_w) &\geq d(a_v, a_w) - (\mathrm{diam}(S_v) + \mathrm{diam}(S_w)) \\
&\geq d(a_v, a_w) - 4\phi \cdot \max(|S_v|-1, |S_w|-1) \\
&\geq |\overline{a_v a_w}| - 4\phi \cdot \max(|S_v|-1, |S_w|-1) \\
&> (2\phi(c+2) - 4\phi) \cdot \max(|S_v|-1, |S_w|-1) \\
&\geq c \cdot 2\phi \cdot \max(|S_v|-1, |S_w|-1) \\
&\geq c \cdot \max(\mathrm{diam}(S_v), \mathrm{diam}(S_w))
\end{aligned}$$

and hence the pair is c-well-separated. □

Lemma 3.2. *Let α be the maximum degree in the minimum spanning tree of $\mathrm{DG}(S)$. Then each pair (S_v, S_w) that ever appears in the queue Q in Algorithm 3.2 satisfies the inequality*

$$\frac{1}{\alpha+1} \le \frac{|S_v|}{|S_w|} \le \alpha + 1.$$

Proof. The proof works by induction. The first pair in Q is (S_r, S_r). Obviously, this pair satisfies the inequality. In the induction step we consider some pair (S_v, S_w) that occurs in the queue. We assume without loss of generality that $P(S_v)$ has been split to generate (S_v, S_w). This means $(P(S_v), S_w)$ appeared in the queue and we can apply the induction hypothesis to get

$$|S_w| \ge \frac{|P(S_v)|}{\alpha+1} \ge \frac{|S_v|}{\alpha+1}.$$

In addition, since $P(S_v)$ has been split, we have $|P(S_v)| \ge |S_w|$ (see Algorithm 3.2 Line 11). Furthermore, the splitting is balanced because of Lemma 2.4. In the end, we have the second inequality

$$|S_v| \ge \frac{|P(S_v)|}{\alpha+1} \ge \frac{|S_w|}{\alpha+1}.$$

If we combine both statements, we can finish the proof. □

Lemma 3.3. *Let $\Xi(S_v) = \{S_w \mid (S_v, S_w) \in \Xi\}$ and $m(S_v) = |\Xi(S_v)|$ be the multiplicity of S_v in Ξ. Then we have $m(S_v) \in \mathcal{O}(\delta c^2 |S_v|)$.*

Proof. This proof needs a volume argument to show that sets of one pair are not far away from each other. As will be seen, the proof of this lemma makes use of the fact that the radius ratio ϕ is bounded by a constant. Otherwise, the size of $\Xi(S_v)$ would grow strongly.

Let $S_w \in \Xi(S_v)$, then we have $(P(S_v), P(S_w)) \notin \Xi$. Now we can assume without loss of generality that $(P(S_v), S_w)$ occurred in Q. Thus, we get by Algorithm 3.2 (Line 7) and the fact that $a_w, a_{P(w)} \in P(S_w)$

$$\begin{aligned} |\overline{a_{P(v)}a_{P(w)}}| &\le |\overline{a_{P(v)}a_w}| + 2\phi(|P(S_w)| - 1) \\ &\le |\overline{a_{P(v)}a_w}| + 2\phi \max(|P(S_v)| - 1, |P(S_w)| - 1) \\ &\le 2\phi(c+3)\max(|P(S_v)| - 1, |P(S_w)| - 1). \end{aligned}$$

Next, we define $R := (\alpha+1)|P(S_v)|$ to show that $\max(|P(S_v)| - 1, |P(S_w)| - 1) \le R$ and $\mathrm{diam}(P(S_w)) \le R$. If we split $P(S_w)$ in Algorithm 3.2 to get (S_v, S_w), then $(S_v, P(S_w))$ has to appear in the queue Q and by Lemma 3.2, we estimate

$$|P(S_w)| - 1 \le |P(S_w)| \le (\alpha+1)|S_v| \le (\alpha+1)|P(S_v)| = R.$$

Alternatively, if we split $P(S_v)$ to get (S_v, S_w), we first apply Lemma 2.4 again and then use the splitting rule (Algorithm 3.2 Line 9) to obtain

$$|P(S_w)| - 1 \leq |P(S_w)| \leq (\alpha + 1)|S_w| \leq (\alpha + 1)|P(S_v)| = R.$$

We immediately obtain $\max(|P(S_v)|-1, |P(S_w)|-1) \leq R$ and finally we use the same inequality as in the proof of Lemma 3.1 to get $\operatorname{diam}(P(S_w)) \leq 2\phi R$ and $|\overline{a_{P(v)}a_{P(w)}}| < 2\phi(c+3)R$. Then all the points of S_w have to be inside the disk $D_r(a_{P(v)})$ with radius $r = 2\phi(c+4)R$. Since R does not depend on S_w, all $S_w \in \Xi(S_v)$ have to lie within $D_r(a_{P(v)})$. Because of Lemma 2.5, we know that $D_r(a_{P(v)})$ contains at most $\mathcal{O}(\delta\phi^2(c+4)^2R^2)$ sites. By combining both properties we obtain

$$\left|\bigcup_{S_w \in \Xi(S_v)} S_w\right| = \mathcal{O}(\delta\phi^2c^2R^2).$$

Using Lemma 3.2, we get $|S_w| \geq |S_v|/(\alpha+1) \geq |P(S_v)|/(\alpha+1)^2$ and therefore $|S_w| \geq R/(\alpha+1)^3$ by our choice of R. We apply Lemma 2.3 and bound $m(S_v)$ with

$$m(S_v) \in \mathcal{O}\left(\frac{\phi^2\delta c^2R^2}{R/(\alpha+1)^3}\right) = \mathcal{O}(\phi^2\alpha^3\delta c^2R) = \mathcal{O}(\phi^2\alpha^5\delta c^2|S_v|) = \mathcal{O}(\phi^{12}\delta c^2|S_v|).$$

We used the fact that $R = (\alpha+1)|P(S_v)| \leq (\alpha+1)^2|S_v|$ (again by Lemma 2.4). As predicted, the size $m(S_v)$ grows strongly with the radius ratio. In our model we postulated a constant radius ratio. Finally, we have $m(S_v) \in \mathcal{O}(\delta c^2|S_v|)$. □

Lemma 3.4. *We have* $|\Xi| \in \mathcal{O}(\delta c^2 n \log n)$.

Proof. For $0 \leq i \leq m := \lceil \log n \rceil$ we define $V_i = \{v \in H \mid 2^i \leq |S_v| < 2^{i+1}\}$. Since H is balanced, V_i contains at most $\mathcal{O}(n/2^i)$ nodes. In addition, we define $\Lambda_i = \{(S_v, S_w) \in \Xi \mid v \in V_i\}$. By Lemma 3.3, we have

$$|\Lambda_i| = \sum_{v \in V_i} m(S_v) \in \mathcal{O}\left(\sum_{v \in V_i} \delta c^2|S_v|\right) = \mathcal{O}(\delta c^2 2^{i+1} \cdot n/2^i) = \mathcal{O}(\delta c^2 n).$$

Finally, $|\Xi| = \sum_{i=0}^{m} |\Lambda_i| \in \mathcal{O}(\delta c^2 n \log n)$. □

We obtain our first theorem by combining the four lemmas. Gao and Zhang used the relative neighbourhood graph (see [36]) to compute a minimum spanning tree in time $\mathcal{O}(n \log n)$. It does not seem to be obvious to use this approach for general disk graphs. Instead we use the fact that S has density δ.

Theorem 3.5. *Let* $\mathrm{DG}(S)$ *be a disk graph with* n *sites, constant bounded radius ratio and density* δ*. Moreover, let* $c \geq 1$*. There is an algorithm that computes a c-well-separated pair decomposition of* $\mathrm{DG}(S)$ *with* $\mathcal{O}(\delta c^2 n \log n)$ *pairs. The running time is* $\mathcal{O}(\delta c^2 n \log n)$.

Proof. Proving the running time proves the theorem. By assumption, we know that the density of S is δ. By Lemma 2.5, we get $\deg(p) \in \mathcal{O}(\delta)$ for all $p \in S$ because $\phi \in \mathcal{O}(1)$. Thus, $\mathrm{DG}(S)$ has $\mathcal{O}(\delta n)$ edges and we can compute its minimum spanning tree in time $\mathcal{O}(\delta n \log n)$ [10]. Algorithm 3.1 needs $\mathcal{O}(n \log n)$ time, because one invocation needs linear time and since the splitting of the tree is balanced we get a balanced computation tree with height $\mathcal{O}(\log n)$. To bound the running time of Algorithm 3.2, we count the number of pairs which occur in the queue Q. Obviously, the running time is proportional to the number of pairs that appear in Q. The construction of Ξ can now be represented by a tree. A node is related to a pair that occurs in Q and if a pair has been split, the resulting pairs are the children of the node. The leaves of the tree are either the pairs of Ξ or the discarded pairs $(\{p\}, \{p\})$. The second case occurs at most $\mathcal{O}(n)$ times whereas the appearance of the first case is bounded by $|\Xi|$. Thus, the computation tree has $\mathcal{O}(\delta c^2 n \log n)$ nodes and since each split costs $\mathcal{O}(1)$, we can bound the total running time by $\mathcal{O}(\delta c^2 n \log n)$. □

3.3 Further Properties and a first step towards arbitrary Density

The following two technical lemmas on well-separated pair decompositions will be useful later on. They are quite similar to the lemmas we can find in [21]. Kaplan et al. use these lemmas for the routing scheme. We need them in Chapter 4.

Lemma 3.6. *Let Ξ be a c-well-separated pair decomposition for $\mathrm{DG}(S)$ and let p, q be two sites such that the pair $(S_u, S_v) \in \Xi$ represents (p, q). Then $c \cdot \mathrm{diam}(S_u) \leq 2\phi c(|S_u| - 1) \leq d(p, q)$.*

Proof. The first inequality follows from Observation 2.1. For the second inequality we use the assumption and the fact that $p \in S_u$ and $q \in S_v$. That means $|\overline{a_u a_v}| > 2\phi(c+2)\max(|S_u| - 1, |S_v| - 1)$. Now we use Observation 2.1 and 2.2 to get

$$\begin{aligned}
d(p,q) &\geq d(a_u, a_v) - 2\max(\mathrm{diam}(S_u), \mathrm{diam}(S_v)) \\
&\geq |\overline{a_u a_v}| - 2\max(\mathrm{diam}(S_u), \mathrm{diam}(S_v)) \\
&> 2\phi(c+2)\max(|S_u| - 1, |S_v| - 1) - 4\phi\max(|S_u| - 1, |S_v| - 1) \\
&\geq 2\phi c \max(|S_u| - 1, |S_v| - 1) \\
&\geq 2\phi c(|S_u| - 1).
\end{aligned}$$

□

Lemma 3.7. *Let Ξ be a c-well-separated pair decomposition for $\mathrm{DG}(S)$ and let p, q be two sites with $d(p, q) < c$. If $(S_u, S_v) \in \Xi$ represents (p, q), then $S_u = \{p\}$ and $S_v = \{q\}$.*

Proof. By Lemma 3.6 and assumption, we get $2\phi c(|S_u| - 1) \leq d(p, q) < c$. Thus we have $|S_u| < 1 + \frac{1}{2\phi}$. Since $\phi \geq 1$ we obtain $|S_u| < 2$ and the claim follows immediately. A similar argument works for S_v. □

In the remaining part of this chapter, we attempt to diminish the density δ. For this step we follow the strategy of Gao and Zhang [15], see also Kaplan et al. [21].

Let $\epsilon_0 > 0$ be an appropriate parameter. Using an iterative strategy we compute an ϵ_0-*net* of S in the following way: we sort the elements of S by decreasing radius, repeatedly pick a site $p \in S$ and delete all the other sites in $D^{\epsilon_0}(p)$ (see Algorithm 3.3). This algorithm ensures that

Algorithm 3.3 Computes an ϵ_0-net of S.

```
Input: $S$, $\epsilon_0 > 0$
Output: $C \subseteq S$ with $|\overline{pq}| \geq \epsilon_0 \max(r_p, r_q)$ (for all $p, q \in C$)
1: sort $S$ by decreasing radius
2: $C := \emptyset$
3: while $S \neq \emptyset$ do
4:     $p := S.next()$
5:     $C := C \cup \{p\}$
6:     $S := S \setminus D^{\epsilon_0}(p)$
7: end while
8: return $C$
```

each $r_p\epsilon_0$-disk of a site p contains only one site and we would like to represent the sites in S with the *clusterheads* in C. There is still one problem: the connectivity of $\mathrm{DG}(C)$ might differ from $\mathrm{DG}(S)$. To resolve this problem, we add further sites of S to C. Two sites $p, q \in C$ are called *neighbours*, iff $|\overline{pq}| > r_p + r_q$, but there are $s, t \in S$ (possibly $p = s$ or $q = t$) such that $pstq$ is a path in $\mathrm{DG}(S)$ and such that $|\overline{ps}| \leq \epsilon_0 r_p$ and $|\overline{tq}| \leq \epsilon_0 r_q$. We call $\{s, t\}$ the *bridge* of p and q. We define B to be the set that contains one arbitrary bridge for each neighbouring pair and set $Z = C \cup B$ (see Algorithm 3.4 and Figure 3.3). This setting has useful properties, which are shown in the next lemmas.

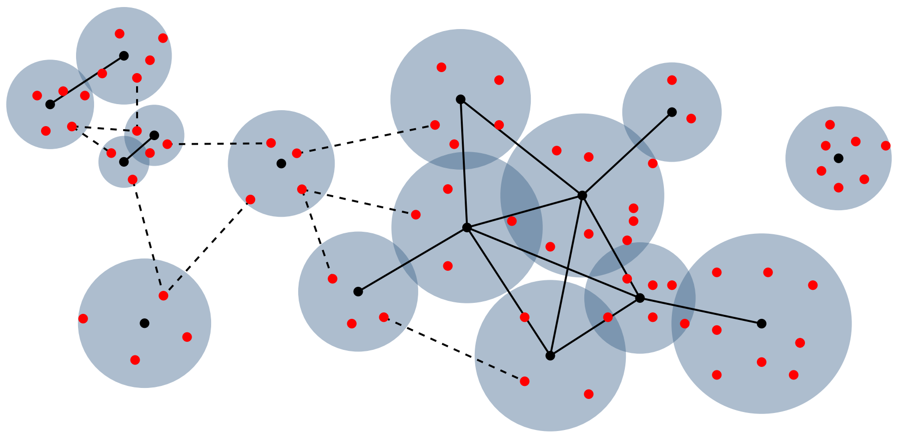

Figure 3.3: *The set C (black points) and the corresponding $\epsilon_0 r$-disks. The normal lines are the connections between two sites of C and the dashed lines are the bridges for some sites. Thus, B consists of all sites that are part of a bridge.*

Lemma 3.8. *The density of C is in $\mathcal{O}(\epsilon_0^{-2})$ and the density of Z is in $\mathcal{O}(\epsilon_0^{-4})$.*

Proof. Let U be a unit disk. Any two sites $p, q \in C$ are of distance at least ϵ_0 away from each other. Thus, we can cover the entire unit disk with $\mathcal{O}(\epsilon_0^{-2})$ disks with radius ϵ_0. Hence, we have $\mathcal{O}(\epsilon_0^{-2})$ sites in U and C has density $\mathcal{O}(\epsilon_0^{-2})$.

Algorithm 3.4 Computes the set Z.

```
Input: $S$, $\epsilon_0 > 0$
Output: $Z \subseteq S$
 1: $C := \epsilon_0$-cluster of $S$ /* Algorithm 3.3 */
 2: $B := \varnothing$
 3: $T := RangeTree(C)$
 4: for $p \in C$ do
 5:   $R_p :=$ square with center $p$ and side length $2(1+\epsilon_0)(r_p + \phi)$
 6:   $Q_p := T.query(R_p)$
 7:   for $q \in Q_p$ do
 8:     $(s,t) :=$ bichromatic closest pair of $(D^{\epsilon_0}(p), D^{\epsilon_0}(q))$
 9:     if $|\overline{st}| \le r_s + r_q$ then
10:       $B := B \cup \{s,t\}$ /* $\{s,t\}$ is a bridge for $p$ and $q$ */
11:     end if
12:   end for
13: end for
14: $Z := C \cup B$
15: return $Z$
```

Let $p \in C$ be a clusterhead. We want to bound the number of bridges for the site p. Therefore, we bound the number of neighbours. A neighbour of p is not contained in the disk $D_{r_p+1}(p)$. But since each radius of a corresponding bridge site is at most ϕ, the neighbour has to be in $D^{1+\epsilon_0}_{r_p+\phi}(p)$. That means, a neighbour lies in an annulus with center p and width $\phi - 1 + \epsilon_0(r_p + \phi)$. This annulus contains at most $\mathcal{O}(\epsilon_0^{-2})$ different sites of C. Since each clusterhead p is neighbour of at most $\mathcal{O}(\epsilon_0^{-2})$ other clusterheads, the disk $D^{\epsilon_0}(p)$ contains $\mathcal{O}(\epsilon_0^{-2})$ bridge sites. Finally, we have $\mathcal{O}(\epsilon_0^{-2})$ clusterheads in U. This finishes the proof. □

However, we are not able to bound the density of Z to $\mathcal{O}(\epsilon_0^{-3})$ or $\mathcal{O}(\epsilon_0^{-2})$. The following lemma shows on the one hand that there is hope to improve the last lemma, but on the other hand it gives some hints about the upcoming problems.

Lemma 3.9. *Let* $\mathrm{DG}(S)$ *be a disk graph with constant bounded radius ratio and a constant number of different radii. Then the density of* Z *is in* $\mathcal{O}(\epsilon_0^{-3})$. *Furthermore, for each* ϵ_0 *there is a set of points with density* $\Omega(\epsilon_0^{-3})$.

Proof. Let U be a unit disk and p be a clusterhead. Again, we bound the number of neighbours for p. Therefore, consider a site q with radius $r_q = r$ that is a neighbour of p. Thus, it is not contained in the disk $D_{r_p+r_q}(p)$, but it has to be in $D^{1+\epsilon_0}_{r_p+r_q}(p)$. That means, q lies in an annulus with center p and the small width $\epsilon_0(r_p + r_q) \le 2\phi\epsilon_0$. This annulus contains at most $\mathcal{O}(\epsilon_0^{-1})$ different sites of C – each site has the corresponding radius r. Since we have a constant number of different radii, all the neighbours of p have to lie in a constant number of annuli with $\mathcal{O}(\epsilon_0^{-1})$ neighbours each. Thus, each clusterhead p is neighbour of at most $\mathcal{O}(\epsilon_0^{-1})$ other clusterheads and the disk $D^{\epsilon_0}(p)$ contains $\mathcal{O}(\epsilon_0^{-1})$ bridge sites. By Lemma 3.8, we have $\mathcal{O}(\epsilon_0^{-2})$ clusterheads in U. This finishes the proof of the upper bound.

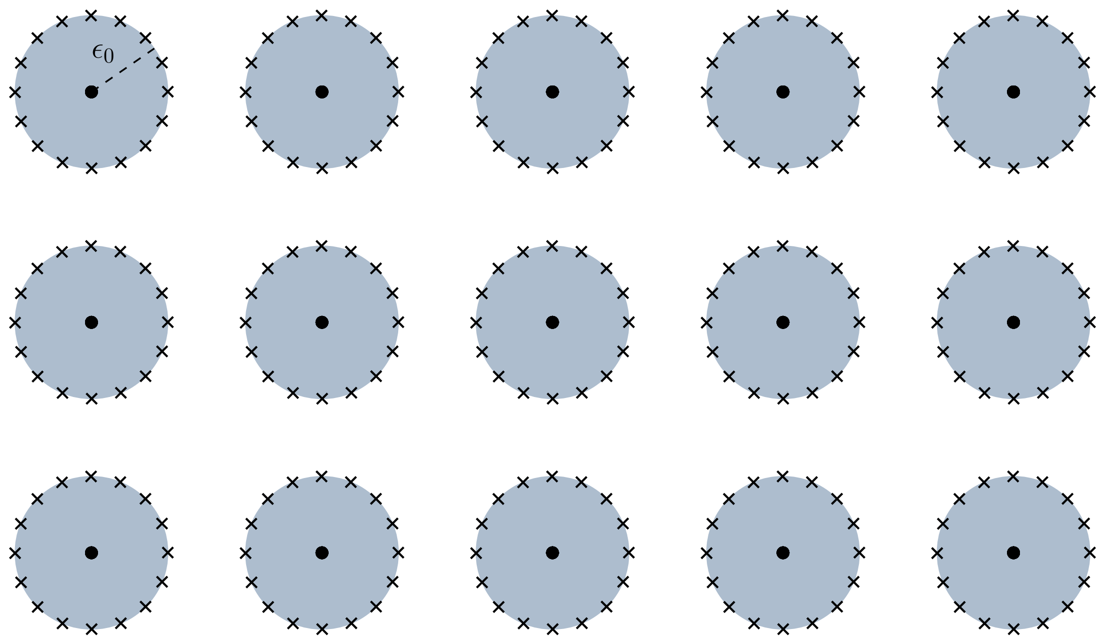

Figure 3.4: *The clusterheads lie on the grid and the crosses are possible candidates for bridge points.*

For the lower bound consider the grid defined by the horizontal lines $h_i: y = i \cdot 3\epsilon_0$ and the vertical lines $v_i: x = i \cdot 3\epsilon_0$ (for $i \in \mathbb{Z}$). As a next step, we describe a set S of sites, such that the corresponding set Z has density $\Omega(\epsilon_0^{-3})$. The radius ratio of S is 1 that means each disk is a unit disk and has radius 1. Furthermore, the clusterheads are the intersections of the vertical and horizontal lines and for each clusterhead $p \in C$ there is a sufficiently large number of sites arranged in a regular polygon on the boundary of $D^{\epsilon_0}(p)$ (candidates for bridge points, see Figure 3.4). Again, let U be a unit disk. It is obvious that U has $\Omega(\epsilon_0^{-2})$ clusterheads. Furthermore, for each clusterhead p there are $\Omega(\epsilon_0^{-1})$ distinct clusterheads in $D_2^{1+\epsilon_0} \setminus D_2(p)$. By construction, we can assume that all the clusterheads are neighbours of p. Thus, we have $\Omega(\epsilon_0^{-2})$ different clusterheads in U and each corresponding ϵ_0-disk contains $\Omega(\epsilon_0^{-1})$ bridge points. Finally, Z has density $\Omega(\epsilon_0^{-3})$. □

Lemma 3.10. *The computation of Z needs $\mathcal{O}(n^2 + \epsilon_0^{-2} n \log n)$ time.*

Proof. Using Algorithm 3.3 we can compute the set C of clusterheads naively in time $\mathcal{O}(n^2)$. For each $p \in C$ we remember the set $D^{\epsilon}(p)$. To compute the neighbouring pairs we use Algorithm 3.4. First, we build a range tree for C in time $\mathcal{O}(n \log n)$ and make a query for each $p \in C$. The query consists of the square with side length $2(1+\epsilon_0)(r_p + \phi)$ centered at p. Since C has density $\mathcal{O}(\epsilon_0^{-2})$ (Lemma 3.8), each query has running time $\mathcal{O}(\log^2 n + \epsilon_0^{-2})$ and the total running time of this part is in $\mathcal{O}(n \log^2 n + n\epsilon_0^{-2})$. The output of the query is called Q_p. By construction, all neighbours of p have to be in Q_p. Now for each pair (p, q) with $q \in Q_p$, we compute the bichromatic closest pair between the two sets $D^{\epsilon_0}(p)$ and $D^{\epsilon_0}(q)$. In the plane, this can be done in $\mathcal{O}(|D^{\epsilon_0}(p) \cup D^{\epsilon_0}(q)| \log n)$ time. Now let s, t be the output of this computation. If s and t share an edge in S, we have found a neighbouring pair. By Lemma 3.8, we know that the density of C is $\mathcal{O}(\epsilon_0^{-2})$. Thus, Q_p contains $\mathcal{O}(\epsilon_0^{-2})$ distinct sites. Furthermore, we observe that each site $s \in S \setminus C$ is related to a constant number of different clusterheads. Finally, we can bound the time for the computation of the bridges by

$$\sum_{\substack{p \in C \\ q \in Q_p}} \mathcal{O}(|D^{\epsilon_0}(p) \cup D^{\epsilon_0}(q)| \log n) = \mathcal{O}(\epsilon_0^{-2} n \log n).$$

This finishes our proof. □

Lemma 3.11. *Let $d_Z(\cdot,\cdot)$ be the metric of $\mathrm{DG}(Z)$. For any two sites $p, q \in C$ we have*

$$d(p,q) \leq d_Z(p,q) \leq (1 + 6\phi\epsilon_0)d(p,q) + 6\phi\epsilon_0.$$

Proof. The first inequality is obvious since we have $Z \subseteq S$. Now let $\pi = p_0 p_1 \ldots p_k$ be a shortest path in $\mathrm{DG}(S)$ for $p_0 = p$ and $p_k = q$ and let s_i be the clusterhead with radius r_i that covers p_i (for $0 \leq i \leq k$). Notice that $s_0 = p$ and $s_k = q$.

First of all, consider two consecutive sites p_i and p_{i+1} and their corresponding clusterheads s_i and s_{i+1}. There are three different cases, how s_i is related to s_{i+1}, see Figure 3.5.

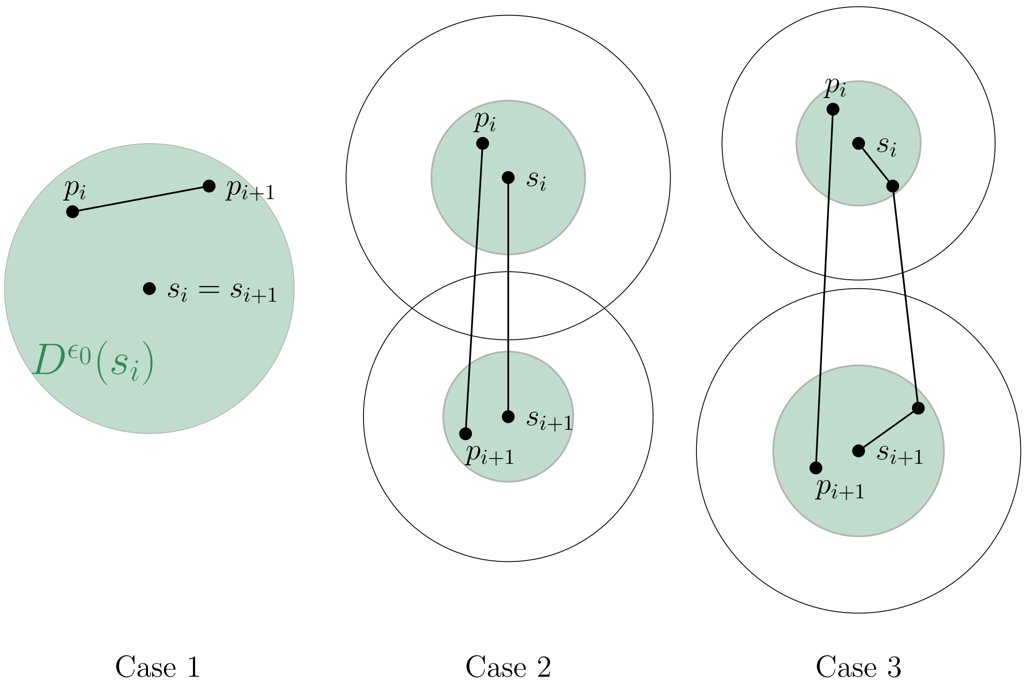

Figure 3.5: *The three cases of Lemma 3.11.*

Case 1: In the first case s_i is the same as s_{i+1}. This means that $d_Z(s_i, s_{i+1}) \leq |\overline{p_i p_{i+1}}| \leq 2r_i\epsilon_0$ because p_i and p_{i+1} have to be inside the disk $D^{\epsilon_0}(s_i)$.
Case 2: In the second case we have $s_i \neq s_{i+1}$ and $|\overline{s_i s_{i+1}}| \leq r_i + r_{i+1}$. Then the triangle inequality, $p_i \in D^{\epsilon_0}(s_i)$ and $p_{i+1} \in D^{\epsilon_0}(s_{i+1})$ show that $d_Z(s_i, s_{i+1}) = |\overline{s_i s_{i+1}}| \leq |\overline{p_i p_{i+1}}| + (r_i + r_{i+1})\epsilon_0$.

Case 3: In the third case we have $s_i \neq s_{i+1}$ again, but $|\overline{s_i s_{i+1}}| > r_i + r_{i+1}$. By definition s_i and s_{i+1} has to be a neighbouring pair. Again, the triangle inequality yields $d_Z(s_i, s_{i+1}) \leq |\overline{p_i p_{i+1}}| + 3(r_i + r_{i+1})\epsilon_0$.
Combining the previous steps, we see

$$\begin{aligned} d_Z(p,q) &\leq \sum_{i=0}^{k-1} d_Z(s_i, s_{i+1}) \\ &\leq 6\phi k \epsilon_0 + \sum_{i=0}^{k-1} |\overline{p_i p_{i+1}}| \\ &\leq 6\phi k \epsilon_0 + d(p,q) \end{aligned}$$

Finally, we have to eliminate the factor k. Knowing that π is a shortest path between p and q, we can estimate $d(p_i, p_{i+2}) \geq 2$ for any $0 \leq i \leq k-2$ because otherwise we could find a short cut using the triangle inequality. This would contradict the assumption that π is a shortest path. Thus, we have

$$\begin{aligned} d(p,q) &= \sum_{i=0}^{k-1} d(p_i, p_{i+1}) \\ &\geq \sum_{j=0}^{\lfloor k/2 \rfloor - 1} d(p_{2j}, p_{2j+1}) + d(p_{2j+1}, p_{2j+2}) \\ &\geq \sum_{j=0}^{\lfloor k/2 \rfloor - 1} d(p_{2j}, p_{2j+2}) \\ &\geq \sum_{j=0}^{\lfloor k/2 \rfloor - 1} 2 = 2 \cdot \left\lfloor \frac{k}{2} \right\rfloor \geq k - 1 \end{aligned}$$

and we can substitute k into the formula to get $d_Z(p,q) \leq (1 + 6\phi\epsilon_0) d(p,q) + 6\phi\epsilon_0$. □

CHAPTER 4

Routing in Disk Graphs

Let S be a set of n sites with density δ and $\mathrm{DG}(S)$ the corresponding disk graph. Again, we assume that the radius ratio ϕ is constant. We present a routing strategy for $\mathrm{DG}(S)$ whose parameters depend on the density. However, in the last section we describe how to eliminate this parameter δ and extend our strategy to site sets with arbitrary density. The idea is based on the work of Kaplan et al. [21] where they use the well-separated pair decomposition (see Chapter 3) to develop a routing scheme for unit disk graphs. We extend this idea to design a routing scheme for disk graphs with constant bounded radius ratio.

The idea is presented in Figure 4.1 and is as follows: let Ξ be the c-well-separated pair decomposition for $\mathrm{DG}(S)$ and T the minimum spanning tree used to compute it. We use the information of the pairs in Ξ to store them among the sites in S such that each site stores $\mathcal{O}(\delta c^2 \log n)$ pairs in its routing table. To walk from p to q, we start from p and walk along T until we find a site r that stores a pair (S_u, S_v) that represents (p, q). We can guarantee that $p, r \in S_u$ and therefore it is sufficient to walk along T_u. We call this process the *local routing*.

For the *global routing* we use a recursive approach. With (S_u, S_v) we store the *middle site* m that lies on the path from r to a_v. This middle site is "halfway" between r and a_v. We recursively route from r to m and then from m to q. Since m is an intermediate target we have to save the primary target by storing it in the header we can use in our routing scheme.

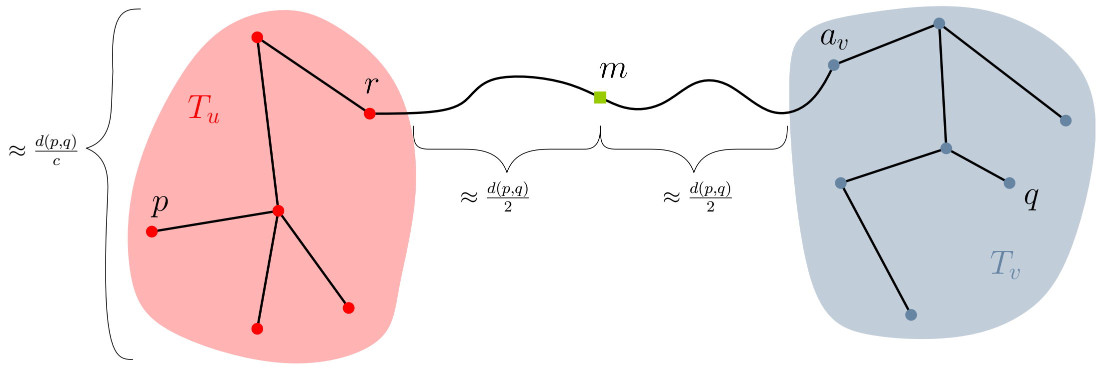
$\approx \frac{d(p,q)}{c}$
T_u
r
p
m
$\approx \frac{d(p,q)}{2}$
$\approx \frac{d(p,q)}{2}$
a_v
q
T_v

Figure 4.1: *To route a package from p to q, we walk along T_u until we find r. Then we route from r to m and from m to q in a recursive manner.*

4.1 Preprocessing the Disk Graph

Let $\epsilon > 0$ such that $1 + \epsilon$ is the desired stretch factor of our routing scheme. First of all, we set $c = (t\phi^3/\epsilon)\log D$ to be the separation parameter of a well-separated pair decomposition, where D is the diameter of $\mathrm{DG}(S)$ and t a sufficiently large constant which will be needed at a later stage. The computed c-well-separated pair decomposition consists of a spanning tree T, the hierarchical decomposition H of T and a sequence of pairs $\Xi = (S_{u_1}, S_{u_2}), ..., (S_{u_m}, S_{u_m})$ of $m \in \mathcal{O}(\delta c^2 n \log n) = \mathcal{O}(\delta\epsilon^{-2} n \log n \log^2 D)$ well-separated pairs that forms a partition of $S \times S$.

In the first step we use Algorithm 4.1 to compute the label $l(p) \in \{0,1\}^*$ for each $p \in S$. We perform a postorder traversal of the hierarchical decomposition of H. For this let l be a counter, initialized to 1. Whenever we visit a leaf p in H, we set $l(p)$ to l and increment l. Otherwise, we encounter an inner node u in H for the last time we annotate it with the interval I_u of the labels in T_u. This will be useful for a later part of the preprocessing, because we can say that a site $p \in S$ lies in the subtree of T_u if and only if $l(p) \in I_u$. Obviously, each label has at most $O(\log n)$ bits.

Algorithm 4.1 Computes the Labelling for the Routing Scheme

```
Input: hierarchical decomposition $H$
    Global integer $l$ (initially 1)
Output: computes labels for all $p \in S$
 1: if $H$ is leaf then
 2:     $p := H.element$
 3:     $l(p) := l$
 4:     $l := l + 1$
 5:     return [$l(p)$]
 6: end if
 7: $I_1 := computeLabelling(H.left)$ /* postorder traversal */
 8: $I_2 := computeLabelling(H.right)$
 9: $H.interval$ :=[$\min I_1, \max I_2$]
10: return $H.interval$
```

Next, we describe the local and the global routing tables. The local routing table $\rho_L(p)$ stores all neighbours of p in the minimum spanning tree T, in counterclockwise order. Additionally, for each corresponding edge we store the level of the corresponding node in H. Since each site in S has at most $\alpha \in \mathcal{O}(1)$ neighbours and the height of H is $\mathcal{O}(\log n)$, the local routing table needs only $\mathcal{O}(\log n)$ bits.

For the global routing table $\rho_G(p)$ we have to work a bit more. We go through all nodes u of H that contain p in their subtree T_u. There are at most $\mathcal{O}(\log n)$ such subtrees. By Lemma 3.3 we know that our well-separated pair decomposition has $\mathcal{O}(\delta c^2 |S_u|)$ well-separated pairs, in which S_u is part of one of the sets. $\mathcal{O}(\delta c^2) = \mathcal{O}(\delta\epsilon^{-2}\log^2 D)$ of these pairs are assigned to the global routing table of p and we ensure that each pair is assigned to exactly one site in S_u. Moreover, for each pair (S_u, S_v) that is assigned to p, we store the interval I_v of the node in the hierarchical decomposition that corresponds to S_v. To store (S_u, S_v) we cannot use the whole set S_u and S_v, but we can store a binary representation of their respective nodes in H. If in addition a_v is not a neighbour of p we save the label $l(m)$ of the middle site m of a shortest path

π from p to a_v. The middle site m is a site lying on π that minimizes the maximum distance, $\max(d(p,m), d(m,a_v))$, between the endpoints of π. Since H has height $\mathcal{O}(\log n)$, p can lie in $\mathcal{O}(\log n)$ different sets S_u. For each such set we store $\mathcal{O}(\delta\epsilon^{-2}\log^2 D)$ different lines in $\rho_G(p)$. Each line consists of the following parts: the binary presentation of the node in H that corresponds to S_u, the binary presentation of the node in H that corresponds to S_v, the interval boundary of I_v corresponding to S_v and in certain cases the label $l(m)$ of a middle site. Each representation needs at most $\mathcal{O}(\log n)$ bits. Thus, ρ_G has $\mathcal{O}(\delta\epsilon^{-2}\log^2 n\log^2 D)$ bits. Again, we used the fact that $\phi \in \mathcal{O}(1)$.

In the next lemma, we bound the preprocessing time. However, we should mention that there are two possibilities for our input. The input is either just the set of sites S or the complete corresponding disk graph. Since we want a routing scheme for $\mathrm{DG}(S)$, we have to compute the graph in the first case. Albeit, this can be done in $\mathcal{O}(n^2)$ time with a simple brute force variant. Fortunately, the preprocessing time will be slightly higher than this. Thus, we can assume that our input is a set of sites.

Lemma 4.1. *The preprocessing time for the routing scheme described above is $\mathcal{O}(n^2\log n + \delta n^2 + \delta\epsilon^{-2} n\log n\log^2 D)$.*

Proof. By Theorem 3.5 and $\phi \in \mathcal{O}(1)$, we can compute the c-well-separated pair decomposition in $\mathcal{O}(\delta\epsilon^{-2} n\log n\log^2 D)$ time. Algorithm 4.1 computes the labels of S in linear time. The same time is needed to distribute the pairs of Ξ and their corresponding intervals among the sites in S. Finally, we have to find a clever way to compute all the middle sites. S has density δ. Thus, $\mathrm{DG}(S)$ has $\mathcal{O}(\delta n)$ edges. First, we compute $\mathrm{DG}(S)$ naively in time $\mathcal{O}(n^2)$ and then compute for each site $p \in S$ the shortest path tree T_p with root p. We use n invocations of Dijkstra's algorithm to compute all the shortest path trees in $\mathcal{O}(n^2\log n + \delta n^2)$ time.

Next, we perform a post-order traversal of each T_p to find the middle sites for the p-q-paths in T_p. We create a mergeable max-heap for each leaf q that contains the key $d(p,q)$ and the value q. Now let m be an inner node of T_p that is processed during the traversal. Each child of m in T_p has a max-heap. We merge all the max-heaps and insert $d(p,m)$ with value m into this heap. The resulting heap is named H_m. During the traversal we maintain the invariant that H_m contains only sites in the subtree of T_p rooted at m for which we have not yet found a middle site. The sites for which m could be a middle site are a prefix of the decreasingly sorted distances $d(p,q)$ with $q \in H_m$, because the distances $d(p,q)$ are increasing monotonically along a root-leaf-path. If we want to find the sites in H_m for which m is a middle site we repeatedly perform an extract-max operation to get the next candidate q and check whether the parent node m' of m in T_p is a better middle site. That means we compare the values $\max(d(p,m), d(m,q))$ and $\max(d(p,m'), d(m',q))$. If the first value is smaller than the second one, m has to be the middle site between p and q. Otherwise, m cannot be the middle site for any other site in H_m. In this case, we process the next site in the post-order traversal. If the max-heap is a Fibonacci Heap, we can merge two heaps in $\mathcal{O}(1)$ time and perform the extract-max operation in $\mathcal{O}(\log n)$ amortized time [10]. Additionally, each element of T_p is inserted and extracted at most once. Thus, we need $\mathcal{O}(n\log n)$ time to find the middle sites for p. Finally, we can find all middle sites in time $\mathcal{O}(n^2\log n)$ and the preprocessing time is $\mathcal{O}(n^2\log n + \delta n^2 + \delta\epsilon^{-2} n\log n\log^2 D)$. □

4.2 The Routing Scheme

In this section, we will describe how the routing function $f: S \times \{0,1\}^* \times \{0,1\}^* \to S \times \{0,1\}^* \times \{0,1\}^*$ works. Consider a start site p and a target site q of S. Our strategy works as follows: we explore the graph $\mathrm{DG}(S)$ in a suitable manner until we discover a site r storing the pair $(S_u, S_v) \in \Xi$ which represents (p,q). This pair has to lie in the global routing table $\rho_G(r)$ of r. Now we consider the subtrees of T that contain p by increasing size. We do an Euler tour in the subtrees until we find the site r. The middle site m of a shortest path from r to a_v is stored in r's global routing table. We push the target site q onto the header stack and perform recursive routing from r to m. After reaching m we pop the original target q from the header stack and route from m to q recursively. Algorithm 4.2 describes the routing process in pseudo-code.

Local Routing

We start at the site p and we would like to find the site r that stores the pair (S_u, S_v) representing (p,q). We know by construction that p and q are contained in S_u. Thus, we only have to perform an Euler tour on T_u to discover the site r. But there is a problem, which we have to solve. Before starting the tour, we do not know u in advance. To fix this, we begin with the leaf in the hierarchical decomposition H that contains p and explore all nodes on the path from the leaf to the root of H until we find u (see Figure 4.2).

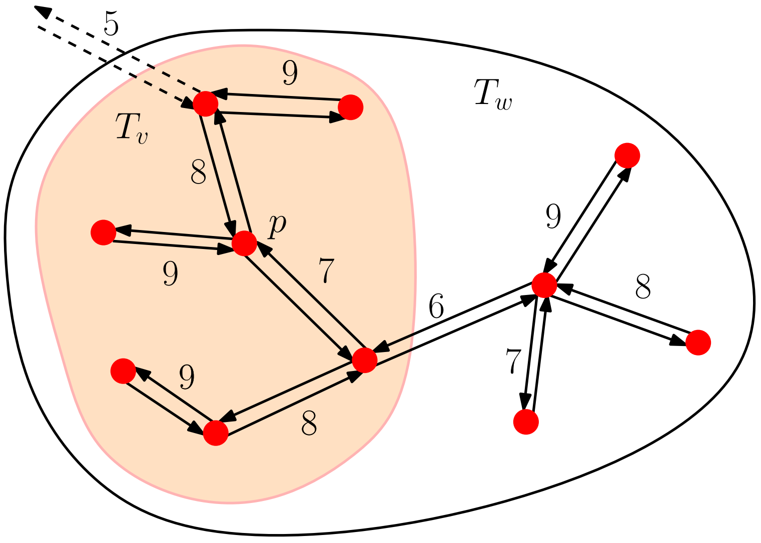

Figure 4.2: *To find the site r we perform an Euler Tour on the subtree T_v that contains s and whose edges have level at least 7. Since we cannot find r, we perform an Euler Tour in the next larger subtree T_w that is $w = P(v)$ in H.*

First, we store p as the start site in the header. Let $v \in H$ be the node to be explored. Furthermore, let l_v be the depth of v in H. We can store this $l = l_v$ in our header, because the depth of v in H is stored in the local routing table of a point incident to the edge represented by v. Recall that T_v is a subtree of the minimum spanning tree T where all edges are of level at least l.

We perform an Euler tour on T_v by using the local routing table. Starting at p we follow an edge in $\rho_L(p)$ that has level at least l. Every time we visit a site r, we check for all pairs (S_u, S_w) in the global routing table $\rho_G(r)$ whether $l(q) \in I_w$ that means, whether $p \in S_w$. If we found such a pair, we clear the local routing information in the header – the Euler tour stops – and start the global routing. If there is none such pair, we scan the local routing table $\rho_L(r)$ for the next edge in counterclockwise order that has level at least l and follow this edge. However, we have to find out at which time the Euler tour is finished. To do this, we have to remember in our header which neighbour of p we visited first. After we have reached p for the last time that means the next possible edge would be the starting edge, we proceed with the parent of v in H by decrementing the counter l. The Euler tour starts from the beginning.

Algorithm 4.2 The routing function for disk graphs.

```
Input: site $p \in S$, targetLabel $l(q) \in \{0,1\}^*$, header $h \in \{0,1\}^*$
Output: nextSite, nextLabel, header
 1: if $l(p) = l(q)$ then
 2:    if $h.stack.isEmpty()$ then
 3:       return $(p, \epsilon, \epsilon)$ /* $\epsilon$ represents the empty word */
 4:    else
 5:       return $(p, h.stack.pop(), h)$
 6:    end if
 7: else if $\rho_G(p)$ stores $(S_u, S_w)$ with $l(q) \in I_w$ then
 8:    $h.startSite := null$
 9:    if $p$ is neighbour of $q$ then
10:       return $(q, l(q), h)$
11:    else
12:       $nextLabel$ := label of middle site for $(S_u, S_w)$
13:       $h.stack.push(l(q))$
14:       return $(p, nextLabel, h)$
15:    end if
16: else
17:    if $h.startSite = null$ then
18:       $h.startSite := p$
19:       $h.level := l_p - 1$
20:       $h.startNext := r :=$ arbitrary neighbour of $p$ with level of edge $\{p, r\} \geq h.level$
21:    else
22:       $r :=$ next counterclockwise neighbour of $p$ with level of edge $\{p, r\} \geq h.level$
23:       if $h.startSite = p$ and $h.startNext = r$ then
24:          $h.level := h.level - 1$ /* Euler tour starts for the next level */
25:       end if
26:    end if
27:    return $(r, l(q), h)$
28: end if
```

Global Routing

Consider a start site p and a target site q such that $\rho_G(p)$ contains the pair (S_u, S_v) with $q \in S_v$. There are two cases:
Case 1: The site q is a neighbour of p. Then we go directly from p to q, since q is stored in the local routing table of p.
Case 2: The site q is not a neighbour of p. Then we also stored the middle site m in the global routing table of p. We push the label of q onto the header stack and use $l(m)$ as an intermediate target. Next we perform a local routing starting from p to find the site storing the pair $(S_{u'}, S_{v'})$ with $m \in S_{v'}$.
After we have reached the target q, we have to look at the header stack. If the stack is empty, q has to be our final destination. Otherwise, we pop the next element from the header stack and set it as the new target label (see Figure 4.3).

4.3 Analysis

Next, we prove that the routing scheme is correct and produces a routing path for all $p, q \in S$ of length at most $(1 + \epsilon)d(p, q)$.

Correctness

To prove the correctness of our routing scheme, we first consider only small distances and show that the routing scheme produces an actual optimal path.

Lemma 4.2. *Let $p, q \in S$ with $d(p, q) < c$ and $c > 2\phi$. Then our routing scheme $\mathcal{R}$ produces a routing path $p_0 p_1 ... p_k$ with the following properties:*

(i) $p_0 = p$ *and* $p_k = q$,

(ii) $d_\rho(p, q) = d(p, q)$ *and*

(iii) the header stack is in the same state at the beginning and at the end of the routing path.

Proof. We prove this lemma by induction. First, we have to characterize the induction parameter. We count for each pair p, q of distinct sites the number of edges on the shortest path between them and sort the pairs increasingly. We delete all pairs with $d(p, q) \geq c$. We conduct the induction on the rank of the sorted list.

First, consider that $d(p, q) = |\overline{pq}| \leq r_p + r_q \leq 2\phi < c$. Then p and q share an edge in $\mathrm{DG}(S)$ and by Lemma 3.7 there exists a pair $(S_u, S_v) \in \Xi$ with $S_u = \{p\}$ and $S_v = \{q\}$. Thus, Algorithm 4.2 correctly routes from p to q, because the pair (S_u, S_v) has to be stored in $\rho_L(p)$. Furthermore, the header remains the same and hence all properties are fulfilled.

Next, consider an arbitrary pair p, q with $r_p + r_q < d(p, q) < c$. Again, by Lemma 3.7 there is a pair $(S_u, S_v) \in \Xi$ with $S_u = \{p\}$ and $S_v = \{q\}$, which has to be stored in the global routing table of p. Thus, Algorithm 4.2 directly proceeds to the global routing phase. Furthermore, since p and q do not share an edge, the global routing table has to contain the middle site m between p and $a_v = q$. The routing algorithm pushes $l(q)$ onto the stack and sets m as an intermediate target. By induction, the routing scheme routes the data package along the shortest path from p to m and then pops the head of the stack (Line 5) which has to be $l(t)$. The properties (i)-(iii) are fulfilled. Afterwards, Algorithm 4.2 routes the packet from m to q and again by induction the data package follows a shortest path from m to q. Afterwards, the header stack is in the same state as before pushing $l(q)$. This satisfies (iii) and the claim follows. □

We proved that the routing scheme works well for short distances. Next, we use Lemma 4.2, to show that the routing scheme produces correct routing paths for other distances as well.

Lemma 4.3. *Let $p, q \in S$ and $c > 2\phi$. Then our routing scheme $\mathcal{R}$ produces a routing path $p_0p_1...p_k$ with the following properties:*

(i) $p_0 = p$ and $p_k = q$ and

(ii) the header stack is in the same state at the beginning and at the end of the routing path.

Proof. Again, we use induction on the rank of the sorted list as described in Lemma 4.2. If $d(p,q) < c$, the claim follows immediately by Lemma 4.2.

Next, consider two distinct sites $p, q \in S$ with $d(p,q) \geq c$. By construction, Algorithm 4.2 finds a site $r \in S$ whose global routing table stores the pair (S_u, S_v) that represents (p,q) and the corresponding middle site m. This middle site exists since $d(p,q) \geq c > 2\phi$. The stack remains unchanged until now (see Figure 4.3), because the local routing information is cleared just before the global routing starts. The routing algorithm pushes $l(q)$ onto the header stack and sets m as a new intermediate target. Again by induction, the routing scheme routes the data package correctly from p to m (Property i). Immediately after the data package arrives at the intermediate target m, the target label $l(q)$ is at the top of the stack (Property ii) and Algorithm 4.2 executes Line 5 to pop the target label and routes the data package from m to q. Again by induction, the data package arrives at q and the stack remains in the same state as before pushing $l(q)$. □

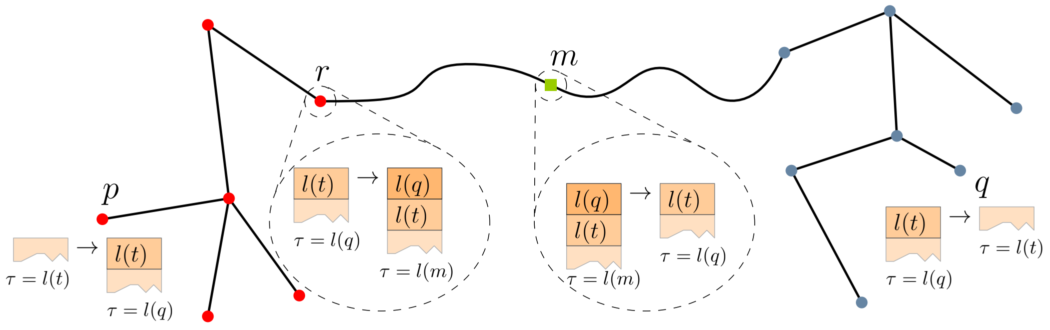

Figure 4.3: *This figure shows how the intermediate target τ and the header stack changes during the routing.*

Stretch factor

After we proved the correctness, we next evaluate how efficient the routing scheme is. The analysis of the stretch factor requires technical work. First, we justify the properties of a middle site.

Lemma 4.4. *Let p and q be two sites in S with $d(p,q) \geq c \geq 24\phi$ and let $(S_u, S_v) \in \Xi$ be the well-separated pair that represents (p,q). If m is a middle site of a shortest path from p to a_v in $\mathrm{DG}(S)$, then we have*

(i) $d(p,m) + d(m,q) \leq (1 + 2/c)d(p,q)$ *and*

(ii) $d(p,m), d(m,q) \leq 0.625 d(p,q)$.

Proof. For the first inequality we use Lemma 3.6 and the fact that m lies on a shortest path from p to a_v. The remaining steps follow by the triangle inequality. Thus, we have

$$\begin{aligned}
d(p,m) + d(m,q) &\leq d(p,m) + d(m,a_v) + d(a_v,q) \\
&= d(p,a_v) + d(a_v,q) \\
&\leq d(p,q) + 2d(a_v,q) \\
&\leq d(p,q) + 2\,\mathrm{diam}(S_v) \\
&\leq \left(1 + \frac{2}{c}\right) d(p,q).
\end{aligned}$$

The other inequalities are much more involved. To prove them, consider a shortest path π from p to a_v that contains the middle site m. Furthermore let m' be the point on π with distance $d(p,a_v)/2$ from p and from a_v. It is possible that m' is not a site in S, because it lies on an edge between two sites. But one of these two sites, let us say $\tilde{m}$, has to satisfy $d(m', \tilde{m}) = |\overline{m'\tilde{m}}| \leq \phi$, since the length of the edges in $\mathrm{DG}(S)$ are at most 2ϕ. Furthermore, by Lemma 3.6, the triangle inequality and our assumption that $d(p,q) \geq c$ we have

$$d(p,a_v) \geq d(p,q) - d(a_v) \geq d(p,q) - \mathrm{diam}(S_v) \geq d(p,q)(1 - 1/c) \geq c - 1.$$

Now we can estimate

$$\begin{aligned}
\max(d(p,\tilde{m}), d(\tilde{m},a_v)) &\leq \frac{d(p,a_v)}{2} + \phi \\
&\leq \frac{d(p,a_v)}{2} + \phi\frac{d(p,a_v)}{c-1} \\
&= \left(\frac{1}{2} + \frac{\phi}{c-1}\right) d(p,a_v) \\
&= \frac{2\phi + c - 1}{2(c-1)} d(p,a_v) \\
&\leq \frac{13c - 12}{24(c-1)} d(p,a_v). && (c \geq 24\phi)
\end{aligned}$$

In the next step we can bound the distances to and from the middle site m. They are at most

$$\max(d(p,m), d(m,a_v)) \leq \max(d(p,\tilde{m}), d(\tilde{m},a_v)) \leq \frac{13c-12}{24(c-1)} d(p,a_v).$$

By applying the triangle inequality and Lemma 3.6 once more, we see $(1-1/c)d(m,q) \leq d(m,a_v)$ and $d(p,a_v) \leq (1+1/c)d(p,q)$. Using the last three estimations we can finally assess

$$\begin{aligned}\max(d(p,m), d(m,q)) &\leq \left(1+\frac{1}{c-1}\right)\max(d(p,m), d(m,a_v))\\ &\leq \left(1+\frac{1}{c-1}\right)\left(\frac{13c-12}{24(c-1)}\right) d(p,a_v)\\ &\leq \left(1+\frac{1}{c-1}\right)\left(\frac{13c-12}{24(c-1)}\right)\left(1+\frac{1}{c}\right) d(p,q)\\ &= \frac{13c^2+c-12}{24c^2-48c+24} d(p,q).\end{aligned}$$

Since $c \geq 24\phi \geq 24$ we have $\max(d(p,m), d(m,q)) \leq 0.625 d(p,q)$. □

Lemma 4.5. *Let $p, q \in S$ with $d(p,q) \geq c$. The total distance travelled by the packet during the local routing phase before the WSPD-pair representing (p,q) is discovered is at most $(162\phi^3)/c \cdot d(p,q)$.*

Proof. Let $(S_u, S_v) \in \Xi$ be the pair that represents (p,q) and let $u_0 u_1 ... u_k = u$ be the path in H from the leaf u_0 corresponding to p to u. Furthermore, let T_i and S_i be the corresponding subtrees of T and sites of S for $0 \leq i \leq k$. The local routing algorithm iteratively performs an Euler tour of each T_i. The tour of T_k may stop early. Additionally, an Euler tour in T_i takes $2|S_i| - 2$ steps . The length of each step is at most 2ϕ. As described in Chapter 3, Algorithm 3.1 guarantees

$$|S_i| \leq |S_{i+1}| - \frac{|S_{i+1}|}{\alpha+1} = \frac{\alpha}{\alpha+1}|S_{i+1}|.$$

Using the geometric progression we can bound the total distance of the local routing to at most

$$\sum_{i=0}^{k} 2\phi(2|S_i|-2) \leq 4\phi \sum_{i=0}^{k} |S_i| \leq 4\phi |S_k| \sum_{i=0}^{k} \left(\frac{\alpha}{\alpha+1}\right)^i \leq 4\phi(\alpha+1)|S_k|.$$

Once again, we use Lemma 3.6 to have $d(p,q) \geq 2\phi c(|S_u|-1)$ and since $S_k = S_u$ we can bound the total distance to

$$4\phi(\alpha+1)|S_u| \leq 4\phi(\alpha+1)\left(\frac{d(p,q)}{2\phi c}+1\right) \leq 4\phi(\alpha+1)\left(1+\frac{1}{2\phi}\right)\frac{d(p,q)}{c},$$

where the last inequality holds by $d(p,q) \geq c$. Finally, since $\alpha \leq 26\phi^2$ by Lemma 2.3 the distance during the local routing is at most $(162\phi^3)/c \cdot d(p,q)$. □

Last but not least, we can bound the stretch factor.

Lemma 4.6. *For any two sites p and q we have $d_\rho(p,q) \le (1+\epsilon)d(p,q)$.*

Proof. We show by induction on $d_\rho(p,q)$ that there is a constant $t > 0$ with

$$d_\rho(p,q) \le \left(1 + (t\phi^3/c)\log d(p,q)\right) d(p,q).$$

The claim then follows from our choice of $c = (t\phi^3/\epsilon)\log D$, because $d(p,q) \le \operatorname{diam}(\mathrm{DG}(S))$. There is nothing to prove for $d(p,q) < c$, since Lemma 4.2 shows that the routing path is an optimal path. Now consider $d(p,q) \ge c$. Then Algorithm 4.2 performs a local routing to find the site r that stores a well-separated pair (S_u, S_v) representing (p,q) in its global routing table. Afterwards, the data package is routed recursively from r to the corresponding middle site m and from m to q. Now we can bound the total routing distance using Lemma 4.5 to $d_\rho(p,q) \le (162\phi^3/c)d(p,q) + d_\rho(r,m) + d_\rho(m,q)$. By induction, we gain

$$\begin{aligned} d_\rho(p,q) \le (162\phi^3/c)d(p,q) &+ \left(1 + (t\phi^3/c)\log d(r,m)\right) d(r,m) \\ &+ \left(1 + (t\phi^3/c)\log d(m,q)\right) d(m,q). \end{aligned}$$

The middle site m lies on a shortest path from r to a_v in $\mathrm{DG}(S)$. Thus, we can apply Lemma 4.4 to get

$$\begin{aligned} d_\rho(p,q) \le (162\phi^3/c)d(p,q) &+ \left(1 + (t\phi^3/c)\log d(r,q) + (t\phi^3/c)\log 0.625\right) d(r,m) \\ &+ \left(1 + (t\phi^3/c)\log d(r,q) + (t\phi^3/c)\log 0.625\right) d(m,q). \end{aligned}$$

Lemma 4.4 again and the fact that $\log 0.625 \le -0.5$ imply

$$d_\rho(p,q) \le (162\phi^3/c)d(p,q) + \left(1 + (t\phi^3/c)\log d(r,q) - t\phi^3/(2c)\right)(1 + 2/c)d(r,q).$$

Moreover, we know that $p, r \in S_u$ and by triangle inequality we obtain $d(r,q) \le d(p,q) + \operatorname{diam}(S_u)$. Applying Lemma 3.6 we see

$$\begin{aligned} d_\rho(p,q) &\le (162\phi^3/c)d(p,q) + \left(1 + (t\phi^3/c)\log d(r,q) - t\phi^3/(2c)\right)(1 + 2/c)(1 + 1/c)d(p,q) \\ &\le (162\phi^3/c)d(p,q) + \left(1 + (t\phi^3/c)\log d(r,q) - t\phi^3/(2c)\right)(1 + 4/c)d(p,q) \\ &= \left(162\phi^3/c + \left(1 + (t\phi^3/c)\log d(r,q) - t\phi^3/(2c)\right)(1 + 4/c)\right) d(p,q) \end{aligned}$$

for $c > 2\phi \ge 2$. Now for $t \ge 648$ we have $162\phi^3/c - (1+4/c)t\phi^3/(2c) \le -(1+4/c)t\phi^3/(4c)$ and hence we can derive

$$d_\rho(p,q) \le \left(1 + (t\phi^3/c)\log d(r,q) - t\phi^3/(4c)\right)(1 + 4/c)d(p,q).$$

This ensures $c \ge 648$. Now we use Lemma 4.4 once again and the fact that $\log(1 + 1/c) \le 1/8$ to obtain

$$\begin{aligned}
d_\rho(p,q) &\le \left(1 + t\phi^3/c \log d(p,q) - t\phi^3/(8c)\right)(1+4/c)d(p,q)\\
&= (1 + t\phi^3/c \log d(p,q))d(p,q)\\
&\quad + \left(4/c(1 + t\phi^3/c\log d(p,q)) - t\phi^3/(8c)(1+4/c)\right)d(p,q)
\end{aligned}$$

It remains to show that $4/c(1 + t\phi^3/c \log d(p,q)) \le t\phi^3/(8c)(1+4/c)$. For $t \ge 648$ we have

$$1 + t\phi^3 \log d(p,q) \le 1 + \epsilon \le 2 \le t\phi^3/32 \le t\phi^3/32(1+4/c).$$

This finally finishes the proof. □

Now we get following theorem.

Theorem 4.7. *Let S be a set of n sites in the plane with density δ and $r\colon S \to \mathbb{R}^+$ a function which assigns to each site a radius $r \in \mathcal{O}(1)$. For any $\epsilon > 0$, we can preprocess S into a routing scheme for $\mathrm{DG}(S)$ with labels of size $\mathcal{O}(\log n)$ bits and routing tables of size $\mathcal{O}(\delta\epsilon^{-2}\log^2 n \log^2 D)$ bits, where D is the diameter of $\mathrm{DG}(S)$. For any two sites p, q the routing scheme produces a routing path with distance at most $(1+\epsilon)d(p,q)$. During the routing the maximum header size is $\mathcal{O}(\log n \log D)$. The preprocessing time is $\mathcal{O}(n^2 \log n + \delta n^2 + \delta\epsilon^{-2} n \log n \log^2 D)$.*

Proof. The proof can be conducted by evaluating the header's size. The rest follows from Lemma 4.1 and Lemma 4.6. During the local phase of the routing algorithm we store the label of the point where we start the Euler tour, together with a counter l. Their binary representations need $\mathcal{O}(\log n)$ bits.

For the global routing part we need to store $\mathcal{O}(\log n)$ bits for one intermediate target. Since S has diameter D and the middle sites split a path from p to q in a balanced way, the recursion depth of the global routing part has to be $\mathcal{O}(\log D)$. Thus, the maximum header size is $\mathcal{O}(\log n \log D)$. □

4.4 Extension to arbitrary Density

Let $1 + \epsilon$ be the desired stretch factor with $\epsilon > 0$. We extend the routing scheme to site sets of unbounded density and follow the strategy described in Section 3.3. We first compute an ϵ_0-net C of S and a set of bridges B to get $Z = C \cup B$ using Algorithm 3.3 and 3.4. We clarify the choice of ϵ_0 later on. Next, we perform the preprocessing routine with ϵ_0 as the stretch parameter for the set Z. We assign new labels to the sites in $S \setminus Z$ and extend the label $l(p)$ of each site in $p \in S$, such that it also contains the label of the next site in C. Thus, the label size remains $\mathcal{O}(\log n)$.

As a next step, we describe the extension of the routing algorithm. First, we assume two points $p, q \in S$ and check whether there is an edge between them. In this case we route the data package straight from p to q. Otherwise, we have $d(p,q) > 2$. Let p' and q' be the clusterheads of p and q. Since we extend the labels of sites in S such that we always know the corresponding clusterhead, we can go the direct way from p to p'. Then, we use the low density algorithm to route from p' to q' and finally go the last step from q' to q. Let $d_{\rho Z}(\cdot,\cdot)$ be the routing distance in $\mathrm{DG}(Z)$. Then we have $d_\rho(p, q) \leq |\overline{pp'}| + d_{\rho Z}(p', q') + |\overline{qq'}|$. We use Lemma 4.6, Lemma 3.11 and the triangle inequality twice to obtain

$$\begin{aligned} d_\rho(p, q) &\leq \epsilon_0(r_{p'} + r_{q'}) + (1 + \epsilon_0)d_Z(p', q') \\ &\leq 2\phi\epsilon_0 + (1 + \epsilon_0)((1 + 6\phi\epsilon_0)d(p', q') + 6\phi\epsilon_0) \\ &\leq 2\phi\epsilon_0 + (1 + \epsilon_0)((1 + 6\phi\epsilon_0)(d(p, q) + 2\phi\epsilon_0) + 6\phi\epsilon_0). \end{aligned}$$

With rearranging we obtain

$$d_\rho(p, q) \leq 10\phi\epsilon_0 + (8\phi + 12\phi^2)\epsilon_0^2 + 12\phi^2\epsilon_0^3 + (1 + (6\phi + 1)\epsilon_0 + 6\phi\epsilon_0^2)d(p, q).$$

Since we have $d(p, q) > 2$, we see

$$d_\rho(p, q) \leq \left(1 + (1 + 11\phi)\epsilon_0 + (10\phi + 6\phi^2)\epsilon_0^2 + 6\phi^2\epsilon_0^3\right) d(p, q).$$

Thus, we have $d_\rho(p, q) \leq (1 + \epsilon)d(p, q)$ if we choose

$$\epsilon_0 \leq \frac{\epsilon}{12\phi^2 + 21\phi + 1}.$$

This discussion discussion leads us to our main theorem.

Theorem 4.8. *Let S be a set of n sites in the plane and $r\colon S \to \mathbb{R}^+$ a function which assigns to each site a radius $r \in \mathcal{O}(1)$. For any $\epsilon > 0$, we can preprocess S into a routing scheme for $\mathrm{DG}(S)$ with labels of size $\mathcal{O}(\log n)$ bits and routing tables of size $\mathcal{O}(\epsilon^{-6} \log^2 n \log^2 D)$ bits, where D is the diameter of $\mathrm{DG}(S)$. For any two sites p, q the routing scheme produces a routing path with distance at most $(1 + \epsilon)d(p, q)$. During the routing the maximum header size is $\mathcal{O}(\log n \log D)$. The preprocessing time is $\mathcal{O}(n^2 \log n + \epsilon^{-4}n^2 + \epsilon^{-6}n \log n \log^2 D)$.*

Proof. Since we choose $\epsilon_0 \in \mathcal{O}(\epsilon)$, the set Z has density $\mathcal{O}(\epsilon^{-4})$ by Lemma 3.8. The theorem then follows from Theorem 4.7 and Lemma 3.10. □

CHAPTER 5

Routing in Polygons

In this chapter, we present the first routing scheme for polygons with holes. In contrast to the routing scheme for disk graphs, we do not have a recursive approach and hence do not need any additional information in the header. Moreover, our routing scheme guarantees that each vertex of the polygon is visited at most once. In the following section we describe a partition of the visibility polygon of a vertex p. This partition is then used for the routing scheme with arbitrarily small stretch.

5.1 Cones in Polygons

Let P be a polygon with h holes and n vertices. Furthermore, let $t > 0$ be a parameter. In this section, we will use a technique of Yao [42], to subdivide the visibility polygon of a site into rays and cones with a certain apex angle.

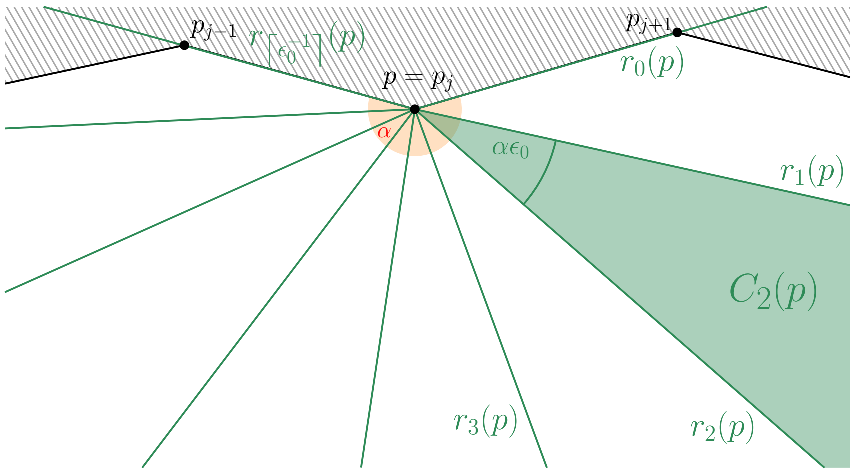

Figure 5.1: *The cones and rays of a vertex p with inner angle α.*

Let $p = p_j$ be a *vertex* in the visibility polygon. Thus, p lies on the boundary of P or on the boundary of a hole in P. We assume that the indices of the outer boundary are increasing clockwise, whereas the indices of the hole boundaries are increasing counterclockwise. Then we denote with $r_0(p)$ the *ray* emanating from p through p_{j+1}. Next, we can rotate this initial ray by certain angles. Let α be the inner angle at p and $\epsilon_0 = \frac{2\pi}{\alpha t}$. We set

$$r_i(p) := \text{rotate } r_0(p) \text{ clockwise by angle } \alpha \cdot \min(i \cdot \epsilon_0, 1)$$

for $1 \le i \le \lceil \epsilon_0^{-1} \rceil$. Using the rays, we can define the *cones* of p. $C_i(p)$ is the cone with apex p and boundary $r_{i-1}(p)$ and $r_i(p)$ (see Figure 5.1). Furthermore, $\mathcal{C}(p)$ is the set of all cones with apex p whose boundaries emanate into P that is $\mathcal{C}(p) = \{C_i(p) \mid 1 \le i \le \lceil \epsilon_0^{-1} \rceil\}$.

Observation 5.1. *Each vertex p with inner angle α has $\lfloor \epsilon_0^{-1} \rfloor$ cones, with apex angle $\alpha\epsilon_0$ and $\lceil \epsilon_0^{-1} \rceil - \lfloor \epsilon_0^{-1} \rfloor \le 1$ cones with apex angle $\alpha(1 - \epsilon_0 \cdot \lfloor \epsilon_0^{-1} \rfloor) \le \alpha\epsilon_0$. Thus, the apex angle of each cone is at most $\alpha\epsilon_0 = 2\pi/t$.*

Lemma 5.2. *Let p be a vertex in P and $\{p, q\}$ an edge of $\text{VG}(P)$ in the cone $C_i(p)$. Furthermore, let r be the closest vertex in $C_i(p)$ to p. Then the following inequality holds:*

$$d(r, q) \le |\overline{pq}| - \left(1 - 2\sin\frac{\pi}{t}\right)|\overline{pr}|.$$

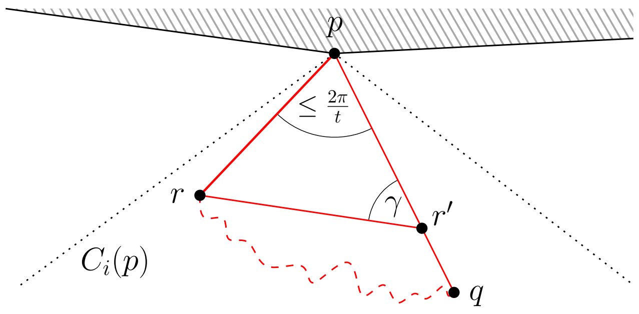

Figure 5.2: *Illustration of Lemma 5.2. The points r and r' have the same distance to p. The dashed line represents the shortest path from r to q.*

Proof. First of all, let r' be the point on $\overline{pq}$ such that $|\overline{pr'}| = |\overline{pr}|$ (see Figure 5.2). Since $p \leftrightsquigarrow q$, we have also $r' \leftrightsquigarrow q$. Furthermore, by construction we have $r \leftrightsquigarrow r'$, because if the line segment $\overline{rr'}$ would intersect the boundary of P, we could find a vertex r'' contained in $C_i(p)$ with $|\overline{pr}| > |\overline{pr''}|$. Now the triangle inequality yields $d(r, q) \le |\overline{rr'}| + |\overline{r'q}|$. Let β be the angle at p in the triangle $\Delta(p, r, r')$. This angle has to be at most $2\pi/t$. Thus, the angle γ at r' is at least $\pi/2 - \pi/t$. After using the Sine Theorem and $\sin 2x = 2\sin x \cos x$ we get

$$|\overline{rr'}| = |\overline{pr}| \cdot \frac{\sin\beta}{\sin\gamma} \le |\overline{pr}| \cdot \frac{\sin\frac{2\pi}{t}}{\sin\left(\frac{\pi}{2} - \frac{\pi}{t}\right)} = |\overline{pr}| \cdot \frac{2\sin\frac{\pi}{t}\cos\frac{\pi}{t}}{\cos\frac{\pi}{t}} = 2|\overline{pr}|\sin\frac{\pi}{t}.$$

Furthermore, the triangle inequality provides $|\overline{r'q}| \le |\overline{pq}| - |\overline{pr'}| = |\overline{pq}| - |\overline{pr}|$ and thus, we have

$$d(r, q) \le 2|\overline{pr}|\sin\frac{\pi}{t} + |\overline{pq}| - |\overline{pr}| = |\overline{pq}| - \left(1 - 2\sin\frac{\pi}{t}\right)|\overline{pr}|.$$

□

5.2 The Routing Scheme

Let P be a polygon with n vertices and h holes. Additionally, let $\epsilon > 0$ be a parameter. We describe a routing scheme for $\mathrm{VG}(P)$ with stretch factor $1 + \epsilon$. The idea is to compute for each vertex p the corresponding set of cones $\mathcal{C}(p)$ and save a certain set of indices for each cone in the routing table of p. These sets of indices form intervals. If an interval of a cone $C_i(p)$ contains a target vertex q we walk to the next vertex contained in $C_i(p)$ (see Figure 5.3). This will provide a small stretch.

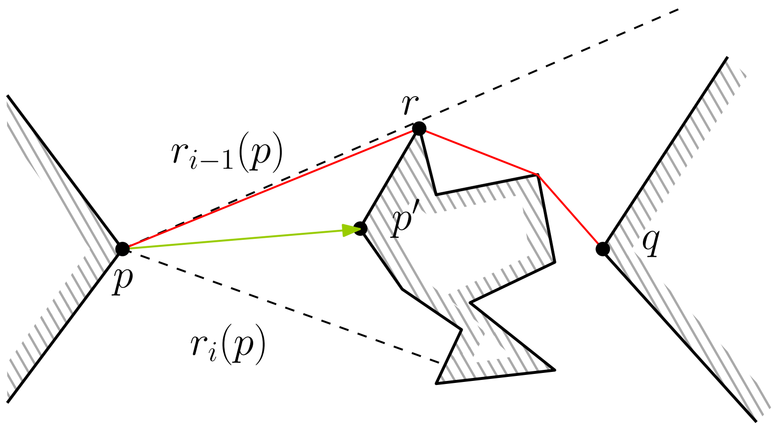

Figure 5.3: *This figure shows the idea of the routing scheme. The first edge on a shortest path from p to q (red) is contained in $C_i(p)$. The routing algorithm will route the package from p to p' (green), the closest vertex to p in $C_i(p)$.*

For the preprocessing we first compute the labels $l(p)$ of each vertex p. These labels consist of the indices of the numbering in P. The first part of $l(p)$ represents one of the $h + 1$ components of the boundary, whereas the second part describes the index of the vertex on the fixed boundary. All the labels are distinct binary strings and have length $\mathcal{O}(\log n)$. It would also be possible to use one index instead of two. This one index should be a binary representation of a number between 1 and n. Using this kind of representation we would need an extra table in the routing table of p. This table shows for each hole the corresponding interval of indices of the vertices contained on the boundary of the hole. Using this table we can map each single piece index to the mentioned two piece index. The table's size is $\mathcal{O}(h \log n)$, because we have to store an interval for each hole. Later on, we will see that our routing table needs asymptotically more information. Thus, the mapping table carries no weight. In fact, we assume without loss of generality that the labels of the vertices have two parts: the number of the hole and the number of the vertex on the boundary of the hole.

For the routing table we do the following: We compute the shortest path tree T_p starting from p. Let $T_p(r)$ be a subtree of T_p with root r. Now let $e = (p, r)$ be a directed edge in T_p between the starting vertex and a neighbour r in $\mathrm{VG}(P)$. For this edge e we can look at all vertices q, where e is the first edge on a shortest path from p to q. That means, we look at all nodes in $T_p(r)$. For $1 \leq i \leq h$ we set $I_r(i) = \{p_{i,j} \mid r \text{ is on the shortest path from } p \text{ to } p_{i,j}\}$. Now we can prove the following lemma.

Lemma 5.3. *Let p be a vertex in P and $e = (p, p')$ an edge in T_p. Then the indices of the vertices in $I_{p'}(i)$ form an interval. Furthermore, let $f = (p, p'')$ be another edge in T_p, such that the cone with apex p spanned by the edges e and f does not contain a third edge of T_p starting at p. Then the indices of the vertices in $I_{p'}(i) \cup I_{p''}(i)$ are again an interval.*

Proof. For the first part of the lemma, consider two distinct vertices $q, r \in I_{p'}(i)$, with lowest common ancestor l in the shortest path tree. We assume that p and q are not contained in an interval. Thus, we can find two vertices $a, b \notin I_{p'}(i)$, such that they lie on different polygonal chains with endpoints q and r induced by the boundary of the i-th hole (see Figure 5.4). Then there has to be a neighbour $\tilde{p}$ of p such that the following holds:

- $\tilde{p}$ lies on the shortest path from p to a or p to b in T_p and
- the shortest path from $\tilde{p}$ to a or the shortest path from $\tilde{p}$ to b crosses the first time either the shortest path from l to q or the shortest path from l to r in T_p.

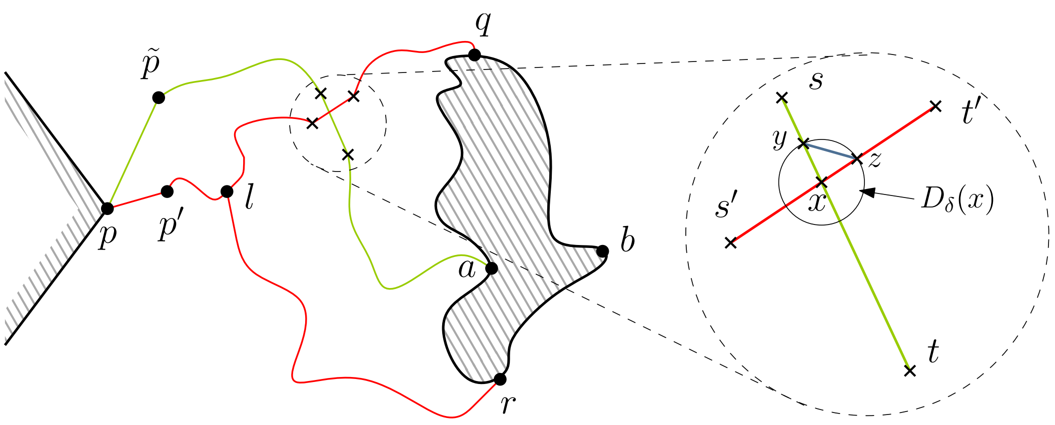

Figure 5.4: *The green line is the shortest path from p to a, whereas the red paths are the "shortest" paths from p to q and r. The blue line segment gives a shortcut from s to t'.*

We consider always the first case and derive a contradiction. The other cases are analogously. That means $\tilde{p}$ lies on the shortest path π from p to a and it crosses the shortest path π' from l to q. Let x be the first intersection of π and π'. The intersection x is not a vertex of P, because otherwise T_p would have a cycle.

Assume that x is the intersection of two directed edges (s, t) and (s', t') in T_p, where s and t lie on π and (s', t') on π'. Again, we have two cases and will only discuss the first one: $d(p, s) + |\overline{sx}| \leq d(p, s') + |\overline{s'x}|$ and vice versa. Since (s, t) and (s', t') are edges in T_p and $x \in \operatorname{int} P$, these edges hit the boundary of P only in their endpoints. Thus, we can find a $\delta > 0$ such that $D_\delta(x)$ is contained in P. Now consider the intersection y of $\overline{sx}$ and the boundary of $D_\delta(x)$ and the intersection z of $\overline{t'x}$ and the boundary of $D_\delta(x)$ (see again Figure 5.4). We have $\overline{yz} \subset D_\delta(x) \subset P$ and the triangle inequality yields $|\overline{xy}| + |\overline{xz}| > |\overline{yz}|$. Thus, the path $syzt'$ is a shortcut from s to t'. This shortcut gives us a shorter path from p to q using the vertex $\tilde{p}$. Hence, we have $q \in I_{\tilde{p}}(i) \neq I_{p'}(i)$ which contradicts the assumption.

For the second part of the lemma, the main argument remains the same. We consider two vertices $q, r \in I_{p'}(i) \cup I_{p''}(i)$ and two vertices $a, b \notin I_{p'}(i) \cup I_{p''}(i)$ that lie on two different polygonal chains with endpoints q and r induced by the boundary of the i-th hole. Again, there has to be a neighbour $\tilde{p}$ of p. This neighbour satisfies the conditions above, because it is not contained in the cone with apex p spanned by the edges e and f. Then we derive again a contradiction and hence the claim follows. □

This powerful lemma gives the main idea for the routing table. We subdivide the surface around p into cones with a certain angle. Thus, we set

$$t := \pi / \arcsin\left(0.5/\left(1+\epsilon^{-1}\right)\right)$$

and use the subdivision described in the previous section. This subdivision provides a set $\mathcal{C}(p)$ with a certain amount of cones. The following lemma specifies this amount.

Lemma 5.4. *For t as above, we have $t \leq 2\pi\left(1+\epsilon^{-1}\right)$.*

Proof. We have $\sin 0 = 0$, $\sin' 0 = 1$ and $\sin' x < 1$ for $0 < x < 0.5$. Thus we have $\sin x \leq x$ for $0 < x < 0.5$. Now for $z > 2$, we obtain the inequality $\sin(1/z) \leq 1/z$ and by the monotonicity of the sine function in the interval $(0, 0.5)$ we have $\arcsin(1/z) \geq 1/z$ for $z > 2$. Thus, we can substitute $z = 2(1+\epsilon^{-1})$ to obtain the desired inequality. □

This lemma and Observation 5.1 show immediately $|\mathcal{C}(p)| \in \mathcal{O}(\epsilon^{-1})$. For each cone $C_j(p) \in \mathcal{C}(p)$ and each hole we collect all the indices $I_{p'}(i)$ with $p' \in C_j(p)$. With Lemma 5.3 it is known that these indices form an interval. This interval contains all indices of vertices whose shortest path starts in the cone $C_j(p)$. To use less space, we save for each cone the intervals of $h+1$ holes and the label of the vertex that has the shortest distance to p in $C_j(p)$. An interval needs $\mathcal{O}(\log n)$ bits, because we store only the boundaries of the interval. Furthermore, in each cone we have $\mathcal{O}(h)$ intervals and one label with size $\mathcal{O}(\log n)$. Thus, the size of the routing table is $\mathcal{O}(\epsilon^{-1} h \log n)$.

The routing function is quite obvious. Starting at a vertex p, we search the label $l(q)$ of the target vertex q in the routing table $\rho(p)$. The search of the label gives a cone and the label of the corresponding shortest neighbour of p. We travel then from p to this neighbour (see again Figure 5.3).

5.3 Analysis

In this section we want to prove the stretch factor and give an upper bound on the preprocessing time. Thus, let P be a polygon with n points and h holes. Additionally, let $1+\epsilon$, $\epsilon > 0$, be the desired stretch factor and let t be as in the last section. First, we want to show that during the routing each step shortened the distance to the target vertex. However, it is not obvious that the routing scheme terminates.

Lemma 5.5. *Let p and q be two vertices in P. Furthermore, let p' be the next vertex computed by the routing scheme that routes a data package from p to q. Then we have $d(p', q) \leq d(p, q) - |\overline{pp'}|/(1+\epsilon)$.*

Proof. First, we look at the routing table of p to find the label $l(q)$. This query supplies a cone and the label of the corresponding vertex p'. Thus, we walk to the vertex p'. However, it is possible that this vertex p' is not located on a shortest path from p to q. But due to our construction, we know that the next vertex q' on the shortest path from p to q is contained in the same cone as p'. We can apply the triangle inequality and Lemma 5.2 which yields

$$\begin{aligned} d(p',q) &\le d(p',q') + d(q',q) \\ &\le |\overline{pq'}| - \left(1 - 2\sin\frac{\pi}{t}\right)|\overline{pp'}| + d(q',q) \\ &= d(p,q) - \left(1 - 2\sin\frac{\pi}{t}\right)|\overline{pp'}| \\ &= d(p,q) - \left(1 - \frac{1}{1+1/\epsilon}\right)|\overline{pp'}| \\ &= d(p,q) - |\overline{pp'}|/(1+\epsilon). \end{aligned}$$

□

This lemma immediately implies the correctness of the routing scheme, because the distance from the vertices to q decreases strictly ($d(p',q) < d(p,q)$) in each step and since there is a finite number of vertices in the polygon, we can find an $m \le n$ such that $\pi = p_0p_1 \dots p_m$ is the path computed by the routing scheme with $p = p_0$ and $q = p_m$. Finally, we can bound the stretch of the routing scheme.

Lemma 5.6. *Let p and q be two vertices of P. Then we have* $d_\rho(p,q) \le (1+\epsilon)d(p,q)$.

Proof. Let $\pi = p_0p_1 \dots p_m$ be the path from $p = p_0$ to $q = p_m$ computed by the routing scheme. By Lemma 5.5 we have $d(p_{i+1},q) \le d(p_i,q) - |\overline{p_ip_{i+1}}|/(1+\epsilon)$. Thus, we have

$$\begin{aligned} d_\rho(p,q) &= \sum_{i=0}^{m} |\overline{p_ip_{i+1}}| \\ &\le (1+\epsilon)\sum_{i=0}^{m}\left(d(p_i,q) - d(p_{i+1},q)\right) \\ &= (1+\epsilon)\left(d(p_0,q) - d(p_m,q)\right) \\ &= (1+\epsilon)d(p,q). \end{aligned}$$

This finishes the proof for the bound of the stretch. □

Finally, we have to discuss the details of the preprocessing and its time complexity.

Lemma 5.7. *The preprocessing time for the scheme described above is* $\mathcal{O}(n^3 + nh\epsilon^{-1})$.

Proof. Let p be a vertex of P. First of all, we compute the shortest path tree T_p. Using the algorithm of Dijkstra and Fibonacci Heaps [10], this can be done in amortized time $\mathcal{O}(n^2)$. Now we make a post-order traversal of T_p to compute the intervals for each child of p. Given a node q in T_p we look at all children of q. The post-order traversal provides at most h different intervals.

For each hole we unify the intervals among the children. Lemma 5.3 shows that the union of these intervals is again an interval. This union can be done in time $\mathcal{O}(h \cdot \operatorname{outdeg}(q))$, where $\operatorname{outdeg}(q)$ is the number of children of q in T_p. Summarizing among all sites in T_p this post-order traversal needs $\mathcal{O}(hn)$ time and is thus insignificant, because $h \leq n$.

Let $q_1 \dots q_k$ be the children of p and $\alpha_1 \dots \alpha_k$ the corresponding angles spanned by the ray $r_0(p)$ and the edge (p, q_j). We sort the q_j by increasing angle α_j. This needs time $\mathcal{O}(n \log n)$. Moreover, we insert into this sorted sequence L the rays $r_i(p)$. By Lemma 5.4 and Observation 5.1 the sequence L contains $\mathcal{O}(\epsilon^{-1} + \operatorname{outdeg}(p))$ elements. Next, we walk through L. Between two rays $r_{i-1}(p)$ and $r_i(p)$ we join all the corresponding intervals. Again by Lemma 5.3 these unions are intervals. Furthermore, we compute the point that is closest to p. Finally, we store the label of the closest point and the h intervals in the i-th row of the routing table $\rho(p)$. This last step takes time $\mathcal{O}(h(\epsilon^{-1} + \operatorname{outdeg}(p))) = \mathcal{O}(h\epsilon^{-1} + hn)$.

Thus, the computation time for one point is $\mathcal{O}(n^2 + h\epsilon^{-1})$ and we need preprocessing time $\mathcal{O}(n^3 + nh\epsilon^{-1})$. □

Combining the last two lemmas and the discussion in Section 5.2 we obtain the following theorem.

Theorem 5.8. *Let P be a polygon with n vertices and h holes. For any $\epsilon > 0$ we can preprocess P into a routing scheme for $\mathrm{VG}(P)$ with labels of size $\mathcal{O}(\log n)$ bits and routing tables of size $\mathcal{O}(\epsilon^{-1} h \log n)$ bits. For any two sites $p, q \in P$, the scheme produces a routing path with stretch $\leq (1+\epsilon)d(p,q)$. The header is always empty and the preprocessing time is $\mathcal{O}(n^3 + nh\epsilon^{-1})$.*

Proof. Again, we only have to say something about P and $\mathrm{VG}(P)$. If P is the input of our routing scheme, we first have to compute $\mathrm{VG}(P)$. Naively, this needs time $O(n^3)$, because for each pair of vertices p, q on the boundary of P we can decide in time $\mathcal{O}(n)$ if there is an edge e, such that $\overline{pq}$ hits the interior of e. In this case, $\{p, q\}$ is not an edge in $\mathrm{VG}(p)$. □

5.4 Improvement for simple Polygons

Let P be a simple polygon with n vertices. Again, let $1+\epsilon$, $\epsilon > 0$, be the stretch factor. In this last section we would like to find an improvement of the preprocessing time for polygons without holes. The problem of the previous section consists in the computation of each shortest path tree, which needs time $\mathcal{O}(n^3)$. In polygons without holes, we can use another technique to avoid this large computation. The routing function, the labelling of the sites and the structure of the routing table remain the same. Certainly, the computation of the routing table is different. Thus, we have to prove that the preprocessing provides the same properties as in the previous section.

First of all, let p be a vertex of P. We compute its visibility polygon $\operatorname{vis}(p)$. This computation provides a sequence of consecutive points $v_0 v_1 \dots v_k$ with $p = v_0 = v_{k+1}$. Some points of this sequence are perhaps not vertices of P. We assume that the sequence is sorted clockwise. Thereby, the points are sorted by increasing angle α_j that is spanned by the ray $r_0(p)$ and the edge $\{p, v_j\}$ $(1 \leq j \leq k)$.

Now s_i is the point of the intersection of $r_i(p)$ and $\operatorname{vis}(p)$ with smallest distance to p. The corresponding edge $e_i \subset P$ with $s_i \in e_i \cap r_i(p)$ can be found by using the sorted sequence of vertices of $\operatorname{vis}(p)$ (see Figure 5.5).

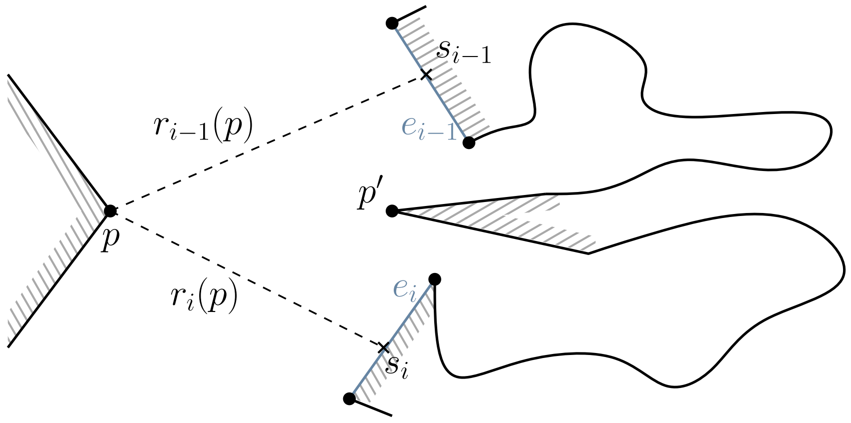

Figure 5.5: *The boundaries of $C_i(p)$ hit ∂P in the points s_{i-1} and s_i. The site p' is the point in $C_i(p)$ with shortest distance to p.*

Now let $C_i(p) \in \mathcal{C}(p)$ be a cone. Recall that the boundaries of $C_i(p)$ are the rays $r_{i-1}(p)$ and $r_i(p)$. The vertices related to this cone accrue in the following manner: starting from s_{i-1} we walk along the boundary of P until we meet s_i and collect all the visited vertices. Obviously, this collection forms a (possibly empty) interval, called $I(i)$. Moreover, we look for the vertex p' with smallest distance to p among the points of $I(i)$. Analogue to the version of our problem with holes, we store the interval boundary of $I(i)$ together with the label of the vertex p' in the routing table of p. This needs $\mathcal{O}(\log n)$ bits as well. Again, by Lemma 5.4, the size of the routing tables are $\mathcal{O}(\epsilon^{-1} \log n)$ bits. This result is the same as in the previous section.

With this we obtain the following lemma.

Lemma 5.9. *Let p, q be two vertices of P and (p, p') the first edge on the shortest path from p to q. If $q \in I(i)$, then $p' \in C_i(p)$.*

Proof. Consider the opposite that is $p' \notin C_i(p)$. Since q is in $I(i)$, the shortest path π from p to q has to cross $\overline{ps_{i-1}}$ or $\overline{ps_i}$ at least twice. The first intersection is p itself. Let $s \neq p$ be the second intersection and π' the subpath of π from p over p' to s. By the triangle inequality we have $|\overline{ps}| < d(\pi')$ (see Figure 5.6). Thus, we can find a shortcut from p to s and hence a shorter path from p to q. This contradicts the assumption that π is a shortest path from p to q and finishes the proof. □

This lemma proves that the routing function behaves as our first algorithm. Thereby, we proved the correctness. Finally, we obtain the last theorem of this thesis.

Theorem 5.10. *Let P be a simple polygon with n vertices. For any $\epsilon > 0$ we can preprocess P into a routing scheme for $\mathrm{VG}(P)$ with labels of size $\mathcal{O}(\log n)$ bits and routing tables of size $\mathcal{O}(\epsilon^{-1} \log n)$ bits. For any two sites $p, q \in P$, the scheme produces a routing path with stretch $\leq (1 + \epsilon)d(p, q)$. The header is always empty and the preprocessing time is $\mathcal{O}(n^2 + n\epsilon^{-1})$.*

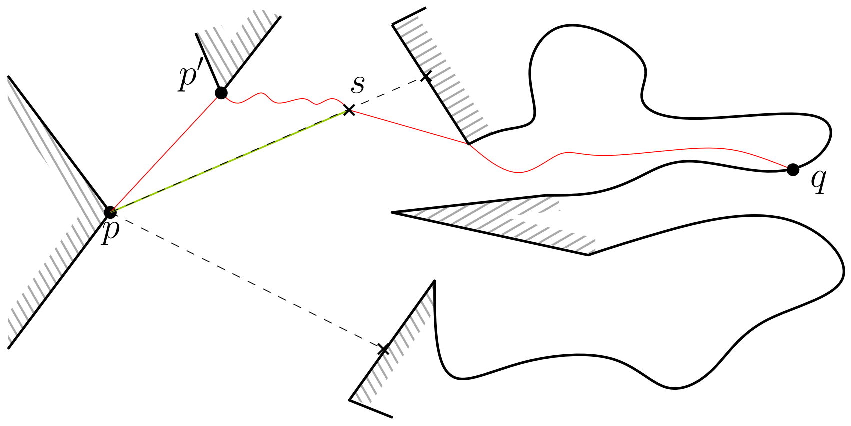

Figure 5.6: *The red line is the "shortest" path from p to q with p' as first step, whereas the green dashed line represents a shortcut from p to s.*

Proof. First, we compute the visibility graph. Using the visibility polygon for each vertex p we can do this in time $\mathcal{O}(n^2)$, because we can check in constant time for each vertex of $\text{vis}(p)$ whether it is a vertex in P.

Now, let L be the sequence of vertices of $\text{vis}(p)$ sorted by increasing angle. Using L, we can find in time $\mathcal{O}(n + \epsilon^{-1})$ all intersecting points s_i and corresponding edges e_i of P. Thus, for each ray $r_i(p)$, we can find in constant time the boundaries of e_i. Finally, let $C_i(p)$ be a cone. We can find in constant time the interval boundaries of $I(i)$ and in $\mathcal{O}(|I(i)|)$ time the point with the smallest distance in $I(i)$ to p. We store the interval and the label of the point in the i-th row of the routing table of p. Among all cones, this step costs $\mathcal{O}(n + \epsilon^{-1})$. In the end, we do the same procedure for all vertices and obtain the running time $\mathcal{O}(n^2 + n\epsilon^{-1})$. □

CHAPTER 6

Conclusion

We have presented two efficient routing schemes: one for the disk graphs and one for the visibility graph of a polygon. Both routing schemes produce a routing path whose length can be made arbitrarily close to the optimum. For the disk graph, we used the fact that the well-separated pair decomposition is small. Nevertheless, the small size of the decomposition depends on the constant radius ratio. Moreover, we can look at the hop-distance $d_h(\cdot,\cdot)$ of a graph. In this model, all edges have length 1. Now let S be a set of sites and let $\operatorname{diam}_h(S)$ denote the diameter of S using the hop-distance. Since $\operatorname{diam}_h(S) \le |S| - 1$ and $|\overline{pq}| \le 2\phi d_h(p,q)$ hold for every two sites $p, q \in S$ for disk graphs with constant radius ratio, the well-separation conditions imply also separation with respect to the hop-distance. Thus, we can find a routing scheme for disk graphs that approximates the number of hops used in the routing path.

We still have various open questions for the routing schemes for polygons. First of all, it would be interesting, if there is a routing scheme that approximates the hop-distance in polygons. Using our routing scheme we can find examples, where the stretch is in $\Omega(n)$ (see Figure 6.1). Moreover, it would be interesting to know, whether the preprocessing time or the size of the routing table can be improved, using perhaps a recursive strategy.

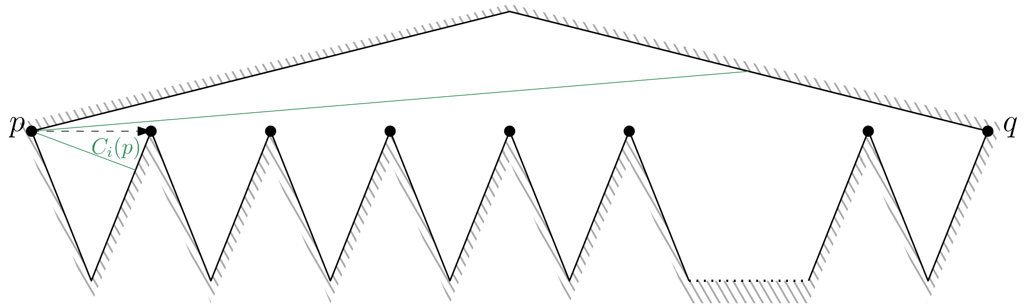

Figure 6.1: *In this polygon, the hop-distance between* p *and* q *is* 1*, because they can see each other. Our routing scheme routes from one spire to the next. Thus, the stretch is in* $\Theta(n)$.

Finally, we have many open questions for the disk graphs. First of all, it would be interesting to improve the size of the routing table. Here we have several parameters to achieve this. One problem is the large exponent of the ϵ^{-1} that we have to introduce when going from bounded to unbounded density. Perhaps there is a smarter way to pick out the bridges or to bound the number of bridges. Furthermore, the size of the routing tables depends on the size of the well-separated pair decomposition. Traditional well-separated pair decompositions have only $\mathcal{O}(c^2 n)$ pairs, while the version for unit disk graphs needs an additional logarithmic factor. Whether this factor can be avoided, is still an open problem. Any improvements on the number of pairs would decrease the size of the routing table. Not forgetting, we are interested in the size of the header. Many routing schemes do not need this header. In our case, we have a recursive approach and have to handle the associated stack. It would be interesting to know, whether it is possible to avoid this stack. Last but not least, our routing scheme works only for disk graphs with constant bounded density, because the well-separated pair decomposition provides a sub-quadratic size only for this case. If the radius ratio is unbounded, the size can be quadratic (see Figure 6.2). It would be interesting, if there are well-separated pair decompositions for general disk graphs that depend on ϕ^c with $c \approx 2$. If so, we could generalise the routing scheme to disk graphs with unbounded radius ratio.

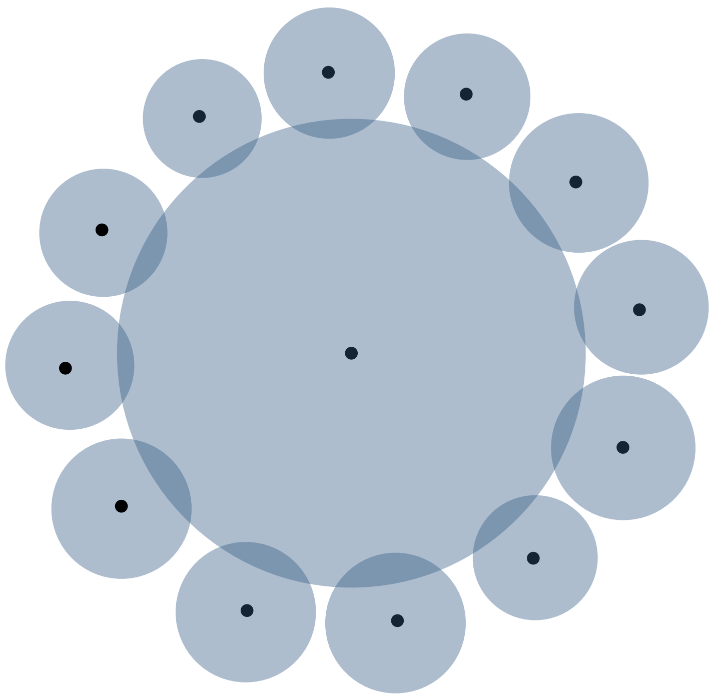

Figure 6.2: *This disk graph has unbounded density and needs $\Theta(n^2)$ well-separated pairs.*

There is one more open question for routing schemes in general: what is the running time that is needed by a data package during its travel through the graph? In particular, it would be interesting how much time a data package needs at one single node. It would be a sightly different but important measure of routing schemes.

Bibliography

[1] Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *International Symposium on Distributed Computing*, pages 404–415. Springer, 2011.

[2] Sunil Arya, David M Mount, and Michiel Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 703–712. IEEE, 1994.

[3] Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1-4):49–63, 1986.

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

[5] Reuven Bar-Yehuda and Bernard Chazelle. Triangulating disjoint jordan chains. *International Journal of Computational Geometry & Applications*, 4(04):475–481, 1994.

[6] Paul B Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 332–340, Nov 1993.

[7] Paul B Callahan and S Rao Kosaraju. Algorithms for dynamic closest pair and n-body potential fields. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 263–272. Society for Industrial and Applied Mathematics, 1995.

[8] Paul B Callahan and S Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, January 1995.

[9] Shiri Chechik. Compact routing schemes with improved stretch. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 33–41. ACM, 2013.

[10] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, third edn. 2009.

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

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

[13] Jeff Erickson. Dense point sets have sparse Delaunay triangulations: or… but not too nasty. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 125–134. Society for Industrial and Applied Mathematics, 2002.

[14] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *International Colloquium on Automata, Languages, and Programming*, pages 757–772. Springer, 2001.

[15] Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing*, 35(1):151–169, 2005.

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

[17] Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate distance oracles for geometric graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 828–837. Society for Industrial and Applied Mathematics, 2002.

[18] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[19] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). *SIAM Journal on Computing*, 34(2):453–474, 2005.

[20] John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[21] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 536–548. Springer, 2016.

[22] Sanjiv Kapoor and SN Maheshwari. Efficient algorithms for euclidean shortest path and visibility problems with polygonal obstacles. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 172–182. ACM, 1988.

[23] Sanjiv Kapoor, SN Maheshwari, and Joseph SB Mitchell. An efficient algorithm for euclidean shortest paths among polygonal obstacles in the plane. *Discrete & Computational Geometry*, 18(4):377–383, 1997.

[24] Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002.

[25] Joseph SB Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3(1):83–105, 1991.

[26] Joseph SB Mitchell. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry & Applications*, 6(03):309–332, 1996.

[27] Giri Narasimhan and Michiel Smid. Approximating the stretch factor of euclidean graphs. *SIAM Journal on Computing*, 30(3):978–989, 2000.

[28] Giri Narasimhan and Michiel Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[29] Mark H Overmars and Emo Welzl. New methods for computing visibility graphs. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 164–171. ACM, 1988.

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

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

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

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

[34] Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15(1):193–215, 1986.

[35] James A Storer and John H Reif. Shortest paths in the plane with polygonal obstacles. *Journal of the ACM (JACM)*, 41(5):982–1012, 1994.

[36] Kenneth J Supowit. The relative neighborhood graph, with an application to minimum spanning trees. *Journal of the ACM (JACM)*, 30(3):428–448, 1983.

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

[38] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10. ACM, 2001.

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

[40] Emo Welzl. Constructing the visibility graph for n-line segments in $\mathcal{O}(n^2)$ time. *Information Processing Letters*, 20(4):167–171, 1985.

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

[42] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.