Geometric Optimization and Querying
Exact & Approximate

Domagoj Matijević

Thesis
for obtaining the degree of a
Doctor of the Engineering Sciences (Dr.-Ing.)
of the natural-technical faculties of
Saarland University

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurswissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken
September, 2007

Tag des Kolloquiums:	07 September, 2007	
Dekan:	Prof. Dr.-Ing. Thorsten Herfet	
Prüfungsausschuss:	Prof. Dr. Reinhard Wilhelm	(Vorsitzender)
	Dr. Stefan Funke	(Berichterstatter)
	Prof. Dr. Friedrich Eisenbrand	(Berichterstatter)
	Prof. Dr. Kurt Mehlhorn	(Berichterstatter)
	Dr. Khaled Elbassioni	

Abstract

This thesis has two main parts. The first part deals with *the stage illumination problem*. Given a stage represented by a line segment L and a set of lightsources represented by a set of points S in the plane, assign powers to the lightsources such that every point on the stage receives a sufficient amount, e.g. one unit, of light while minimizing the overall power consumption. By assuming that the amount of light arriving from a fixed lightsource decreases rapidly with the distance from the lightsource, this becomes an interesting geometric optimization problem. We present different solutions, based on convex optimization, discretization and linear programming, as well as a purely combinatorial approximation algorithm. Some experimental results are also provided.

In the second part of this thesis, we are concerned with two different geometric problems whose solutions are based on the construction of a data structure that would allow for efficient queries. The central idea of our data structures is the *well-separated pair decomposition*. The first problem we address is the *k-hop restricted shortest path under the power-euclidean distance function*. Given a set P of n points in the plane and the distance function $|pq|^{\delta} + C_p$ for some constant $\delta > 1$, nonnegative offset cost C_p and $p, q \in P$, where $|pq|$ denotes the Euclidean distance between p and q, we consider the problem of finding paths between any pair of points that minimize the lenght of the path and do not use more than some constant number k of hops. Known exact algorithms for this problem required $\Omega(n \log n)$ per query pair (p,q). We relax the exactness requirement and only require approximate $(1+\varepsilon)$ solutions which allows us to derive schemes which guarantee constant query time using linear space and $O(n \log n)$ preprocessing time. The dependence on ε is polynomial in $1/\varepsilon$. We also develop a tool that might be of independent interest: For any pair of points $p, q \in P$ report in constant time the cluster pair (A,B) representing (p,q) in a well-separated pair decomposition of P. The second problem in this part is so-called *cone-restricted nearest neighbor*. For a given point set in Euclidean space we consider the problem of finding (approximate) nearest neighbors of a query point but restricting only to points that lie within a fixed cone with apex at the query point. We investigate the structure of the Voronoi diagram induced by this notion of proximity and present approximate and exact data structures for answering cone-restricted nearest neighbor queries. In particular, we develop an approximate Voronoi diagram of size $O((n/\varepsilon^d)\log(1/\varepsilon))$ that can be used to answer cone-restricted nearest neighbor queries in $O(\log(n/\varepsilon))$ time.

Zusammenfassung

Diese Arbeit besteht aus zwei Teilen. Der erste Teil behandelt das *Stage Illumination Problem*. Hierbei möchte man eine Bühne, die durch ein Geradenstück repräsentiert ist, durch Lichtquellen, die durch Punkte in der Ebene repräsentiert sind, so beleuchten, dass jeder Punkt der Bühne genügend Licht erhält und dabei möglichst wenig Energie verbrauchen. Wenn man annimmt, dass die Lichtintensität stark mit der Entfernung zur Lichtquelle abnimmt, so stellt dies ein interesanntes geometrisches Optimierungsproblem dar. Wir geben verschiedene Lösungen an, die sowohl auf konvexer Optimierung, Diskretisierung und Linearer Programmierung basieren, als auch einen kombinatorischen Approximationsalgorithmus. Es werden auch experimentelle Resultate angegeben.

Im zweiten Teil dieser Arbeit behandeln wir zwei verschiedene geometrische Probleme, deren Lösungen auf einer Datenstruktur basieren, die effiziente Anfragen beantworten kann. Die zentrale Idee unserer Datenstruktur ist die *well-separated pair decomposition WSPD*. Das erste Problem, das wir ansprechen ist das *k-hop restricted shortest path under the power-euclidean distance function*. Für n Punkte in der Ebene möchte man den kürzesten Pfad zwischen zwei beliebigen Punkten finden, der nicht mehr als k Kanten benötigt. Bekannte exakte Algorithmen für dieses Problem benötigen $\Omega(n \log n)$ Zeit pro Anfrage (p,q). Wir lockern die Exaktheitsforderung und verlangen nur eine $(1+\varepsilon)$-Approximation. Dies erlaubt uns eine Methode zu entwickeln, die konstante Zeit pro Anfrage garaniert und nur linearen Platz benötigt bei einer Vorverarbeitungszeit von $O(n \log n)$. Die Abhängigkeit von ε ist polynomiell in $1/\varepsilon$. Außerdem entwickeln wir eine Methode, die davon unabhängig von Interesse ist. Für ein Punktepaar $p,q \in P$ bestimmen wir in konstanter Zeit das Cluster-paar (A,B), das (p,q) in einer WSPD von P bestimmt. Das zweite Problem in diesem Teil ist das sogenannte *cone-restricted nearest neighbor problem.* Für eine gegebene Menge von Punkten im Euklidischen Raum betrachten wir das Problem den nächsten Nachbarpunkt zu bestimmen, der in einem Kegel liegt, dessen Spitze ein beliebiger Anfragepunkt ist. Wir untersuchen das dazugehörige Voronoi-Diagramm und entwickeln effiziente Datenstrukturen sowohl für exakte als auch für approximative cone-restricted nearest neighbor-Anfragen. Im speziellen entwickeln wir ein approximatives Voronoi-Diagramm der Größe $O((n/\varepsilon^d)\log(1/\varepsilon))$, das dazu benutzt werden kann, Anfragen in der Zeit $O(\log(n/\varepsilon))$ zu beantworten.

Kurzzusammenfassung

Diese Arbeit befasst sich mit geometrischen Optimierungsproblemen und besteht aus zwei Teilen. Der erste Teil behandelt das *Stage Illumination Problem*. Wir geben verschiedene Lösungen und experimentelle Resultate an.

Im zweiten Teil dieser Arbeit behandeln wir zwei verschiedene geometrische Probleme, deren Lösungen auf einer Datenstruktur basieren, die effiziente Anfragen beantworten kann. Das erste Problem, das wir ansprechen ist das *k-hop restricted shortest path under the power-euclidean distance function*. Wir entwickeln eine Methode, die $(1+\varepsilon)$-approximative Lösungen in konstanter Zeit pro Anfrage garaniert und nur linearen Platz benötigt. Das zweite Problem in diesem Teil ist das sogenannte *cone-restricted nearest neighbor problem.* Wir entwickeln effiziente Datenstrukturen sowohl für exakte als auch für approximative cone-restricted nearest neighbor-Anfragen.

Acknowledgements

I could not have written this thesis without the help of many people. First of all, I would like to thank my advisor and friend Dr. Stefan Funke for his guidance and advice throughout my doctoral study. I am also grateful to Prof. Dr. Kurt Mehlhorn. It has been a great pleasure and an honor to be a member of the Algorithms and Complexity group at the Max-Planck-Institut für Informatik in Saarbrücken. Many thanks to Prof. Dr. Friedrich Eisenbrand for agreeing to co-referee this thesis.

During the last four years I had many fruitful discussions with almost every member of our group. In particular, I am grateful to my coauthors Andreas Karrenbauer, Dr. Theocharis Malamatos, Prof. Dr. Peter Sanders and Dr. Nicola Wolpert.

I would also like to express my gratitude to friends who also contributed to this work, though not directly, yet more than they might think: Stefan Canzar, Khaled Elbassioni, Sören Laue, Debapriyo Majumdar, Nabil Mustafa, Rouven Naujoks, Ralf Osbild and Ružica Piskač.

Table of Contents

List of Figures

List of Tables

Introduction

Concepts of computational geometry can be found in many places of everyday life. For example, imagine being in a big city and looking for a post office. There are several post offices in the city, but naturally you would prefer to know which one is closest to your current position. This problem was mentioned for the first time in [Knu73] and is known in the literature as *Knuth's Post Office Problem*.

A city map which shows the exact position of each post office would be helpful, but still, there might be several post offices which are similarly close to your current position. To find exactly the post office which is closest, it would be nice if the map was subdivided into areas which have one particular post office as closest. Now the question is, how do these areas look like and how can one compute them?

Even though you probably do not really care if you go to a post office which is not the closest but almost as close, this problem describes a fundamental geometric concept, which has many applications. The subdivision of the map is so-called Voronoi diagram. It can be used to model areas of influence of radio transmitters, guide robots, and even to describe and simulate the growth of crystals. There are many more geometric concepts which can be found in everyday life.

In this thesis we shall present results on three different geometric problems using techniques of *geometric optimization and querying.*

Geometric Optimization

STAGE ILLUMINATION PROBLEM[1]. Suppose you are a director of a theatre play, and you would like that the stage on which your actors perform is well illuminated. Namely, you would like to assign powers to a fixed set of light sources placed in the theatre such that there are no dark areas on the stage. However, you don't want the electric bill to be too high at the end of the month. Thus, you want to minimize the cost

[1]This problem was posed in the open-problem session of the 17th Canadian Conference on Computational Geometry [DO06] by Joseph O'Rourke as 'Illuminating with True Lightbulbs'.

of electricity under the constraint that the whole stage is adequately illuminated. By assuming that the amount of light arriving from a fixed light source decreases rapidly with the distance from the light source[2], this becomes an interesting geometric optimization problem. For a 2D variant of the stage illumination problem, see Figure 1.

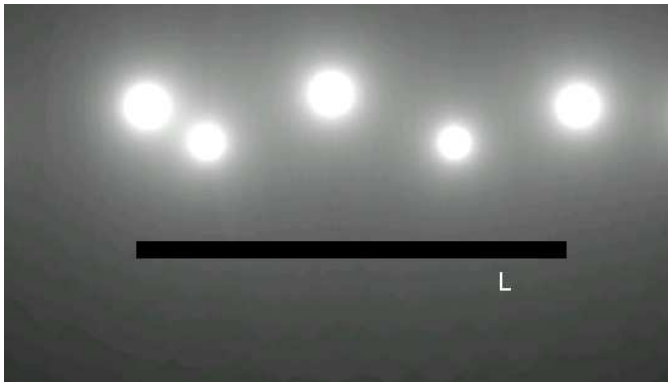

Figure 1: *Given a stage represented by a line segment L and a set of light sources in the plane, assign powers to the light sources such that every point on the stage receives a sufficient amount – e.g. one unit – of light while minimizing the overall power consumption.*

The stage illumination problem is just one of many problems that can be formulated as a geometric optimization problems, which has made this a very attractive area of research during the last twenty years.

In Part I of the thesis, we deal with the stage-illumination problem informally introduced above and present different solutions, some based on convex/linear programming, as well as a purely combinatorial approximation algorithm. We use a dual-fitting technique for bounding the quality of solutions of our combinatorial algorithm. We also provide some experimental results in order to stress the difference in running time and quality of solutions of approaches relying on linear programming as opposed to combinatorial solutions. This work has been published in the *21st Annual Symposium on Computational Geometry* (see [EFKM05]).

Geometric Querying

In Part II of the thesis we present efficient query data structures whose construction is based on Well-Separated Pair Decomposition (WSPD) for the following two geometric problems:

[2] In physics, a point light source illuminates according to an inverse-square law.

k-HOP RESTRICTED SHORTEST PATH IN POWER-EUCLIDEAN NETWORKS. Suppose you are a manager of an airline company and you would like to minimize the fuel consumption of your airplanes routed between the airports. Namely, suppose you want to plan a path of a flight between source and destination airport. The longer the plane flies without a stop the more fuel needs to be filled in. As a consequence, the weight of the plane grows which in turn increases the cost of a mile of flight. The cost per mile in such cases can be mathematically described with a function that increases superlinearly with the distance of the flight. Thus, you might find it cheaper to stop at some intermediate airports, even if this increases the total length of the path (see Figure 2). But in turn, stopping by at too many airports will account for the additional

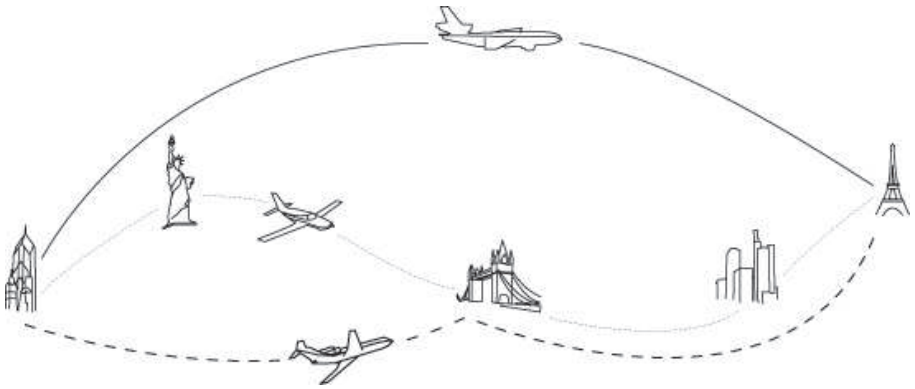

Figure 2: *Economical route planning between source and destination airport with* $1,2,4$*-hops different paths*

and undesirable distance independent overhead (e.g. the take-off phase can consume considerably more fuel). To avoid such unwanted scenarios you can plan the most economical route between source and destination airport with the constraint that not more than a certain number of airports, say k, can be used in between.

This motivation was first proposed by [EHP98, CE01] to give more insight into the k-hop restricted shortest path problem. The positive cost of the flight between airports p and q is mathematically described via a function $\omega(p,q) = |pq|^{\delta} + C_p$ for some constant $\delta > 1$ and nonnegative offset C_p that accounts for distance independent costs. $|pq|$ denotes the Euclidean distance between p and q.

Our main contribution in this part is a data structure of linear size based on WSPD that outputs $(1+\varepsilon)$ approximate k-hop restricted shortest path solutions extremely quickly (e.g. query time is completely independent of the input size). This work has appeared in the *11th Annual European Symposium* (see [FMS03]) and in the follow-up and more experimental paper that has been published in the *1st International Workshop on Algorithms for Wireless and Mobile Networks* (see [FMS04]).

CONIC NEAREST NEIGHBOR (CNN). Suppose we are given a set S of n points in the plane. A cone C having its apex at the point q is a wedge bounded by two rays emanating from q that make an angle of at most π. We want to determine the nearest neighbor $s_q \in S$ that is contained in the cone C with apex at q. Point s_q we shall call a

conic nearest neighbor of q.

We show how to construct approximate conic Voronoi diagram (cAVD) of near-linear size for a set S of points in a low-dimensional Euclidean space which allows for poly-logarithmic query time. The construction of cAVD is based on WSPD. We also examine the structure of the exact linear size conic Voronoi diagram induced by the notion of conic proximity in 2 and 3 dimensions and develop a data structure such that exact cone queries can be answered in sublinear time. This result has been published in the *18th Canadian Conference on Computational Geometry* (see [FMMW06]).

In the k-hop restricted shortest path problem as well as in cNN problem, we are interested in answering many different queries. Thus, we allow some *preprocessing* on a given instance of the problem, such that the desired computation of either k-hop shortest path or cNN can be performed 'quickly'. In general, classes of problems induced by a collection of geometric objects where we want to preprocess a given instance of a problem to a query data structure we shall call **geometric querying problems**. Note that this is conceptually different to the 'Stage illumination problem', where we have been concerned with finding a solution to a particular instance of the problem.

Part I

Geometric Optimization

Chapter 1

Energy-Aware Stage Illumination

1.1 Introduction

Illumination and guarding problems have been popular in mathematics and computer science for several decades. One instance in this class of problems is the Art Gallery Problem posed by Victor Klee : *"How many guards are necessary, and how many are sufficient to guard the paintings and works of art in an art gallery with n walls?"*, where the art gallery is represented by a simple polygon and every two points x and y in a simple polygon can see (guard) each other if the open line segment xy lies entirely within the interior of the polygon. While this particular problem has been solved shortly after by Chvatal proving a tight $\lfloor \frac{n}{3} \rfloor$ bound, many other variants in this problem class have appeared in the literature, e.g. see Ref. [BMKM05], [CEHP04], [CCRCU98], [CT98], [Eid02], [EHP02], [FT77], [GB01], [KKK83], [LL90], [ST03], [Tot00]; also see Ref. [Urr00] for a general survey of the topic.

On one hand, people have restriced the allowable 'floor plans', i.e. special classes of polygons like orthogonal polygons, or looked at the problem of guarding a set of buildings from the outside. Kahn et al. have shown (see Ref. [KKK83]) that any orthogonal polygon with n vertices can be guarded with $\lceil \frac{n}{4} \rceil$ guards, and $\lceil \frac{n}{4} \rceil$ are sometimes necessary. Fejes Toth (see Ref. [FT77]) has shown that for any family $\{S_1, \ldots, S_n\}$ of n disjoint compact convex sets in the plane, one can illuminate the boundaries of the sets by $4n-7$ light sources in the complement of $S_1 \cup \cdots \cup S_n$ and sometimes that many light sources are necessary. Common to these results is the fact that they assume that guards or light sources cover a 360^o field of view, and distance does not affect guarding or illumination abilities.

So other people have come up with models for less powerful guards and light sources, for example by requiring the guards to be placed at the vertices or edges of the polygon. Another restriction is to limit the field of view of the guards to an angle of 180^o, or incorporate the used field of view of the guards or illumination angle of the light sources into the objective to be optimized. For example Lee and Lin (see Ref. [LL90])

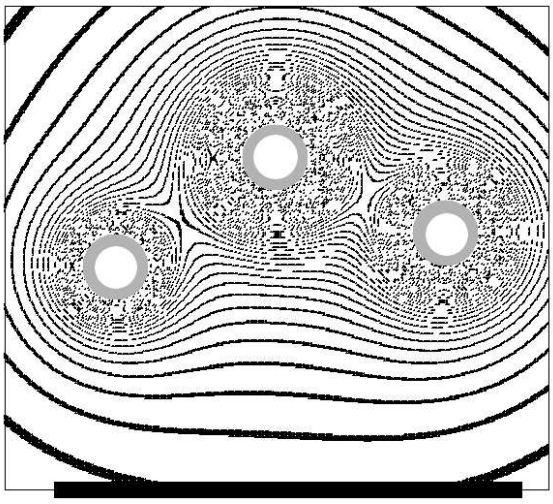

Figure 1.1: *Isolines of light intensities induced by 3 light sources (unfortunately with moiré effects close to the light sources).*

have shown that finding the *minimum* number of vertex guards for a polygon is *NP*-hard. Toth (see Ref. [Tot00]) has shown that $\lfloor \frac{n}{3} \rfloor$ light sources with illumination angle π suffice to illuminate any polygon with n vertices (light sources need not be placed on vertices of the polygon). Given a line segment L (the stage) and a set of n points $p_1, \ldots, p_n$ (light sources), Czyzowicz et al. (see Ref. [CCRCU98]) have proved that it is possible to find in $O(n \log n)$ time a set of floodlights $f_1, \ldots f_n$ with apexes in the set $\{p_1, \ldots, p_n\}$ and angles of illumination $\alpha_1, \ldots \alpha_n$ such that the stage L is illuminated and the sum $\alpha_1 + \cdots + \alpha_n$ is minimized.

The model we propose in this Chapter is in the spirit of the angle restriction employed by Czyzowicz et al., since we also aim to allow only less powerful light sources. But while Czyzowicz et al. disallow omnidirectional light sources (modelling floodlights in the real world), we take into account that the light emitted from a light source spreads with increasing distance, so the amount of light arriving at a fixed area patch decreases with the distance from the light source (see also Figure 1.1, where we have sketched the isolines of light intensities induced by 3 light sources). The rationale behind our model is that it seems rather unrealistic for a guard (or a light source) to monitor (or illuminate) things that are arbitrarily far away, even in the absence of occlusions.

Our contribution

This Chapter proposes to reconsider the large collection of classical illumination problems under a light attenuation model, where the amount of light arriving from a particular light source decreases rapidly with the distance. As a first example, we consider the simple problem of illuminating a stage using a fixed set of light sources, where the goal is to minimize the total amount of power assigned to the light sources while

ensuring a sufficient illumination of the stage. Several approaches in decreasing order of weight of the employed machinery are presented, namely

- a polynomial-time solution based on a convex programming formulation
- a $(1+\varepsilon)$-approximate solution based on discretization and linear programming
- a purely combinatorial $O(1)$ approximate solution with running time $O(n^2)$

We also present some experimental results suggesting that the performance analysis of the combinatorial algorithm is overly pessimistic, leaving an improved analysis and the consideration of other illumination problems in this model as open problems.

1.2 Preliminaries

Consider the following setting: We are given a closed line segment $L \subset \mathbb{R}^2$ and a set of points $S \subset \mathbb{R}^2$, $|S| = n$. L denotes the *stage*, S a set of light sources. Our goal is to assign powers x_s to each light source $s \in S$ such that any point of the stages receives a 'sufficient' amount of light – we will be more precise about that after quickly introducing the physical light model.

1.2.1 The physical model

For the physical model we consider the setting in three dimensions, treating the light sources as points that emit their energy isotropically. Thereby, the energy that hits concentric spheres around the light source is always the same but its density decreases with growing radius. Since the energy is homogeneously distributed over the surface of such a sphere we get

$$F = \int_0^{2\pi} \int_0^{\pi} \frac{E}{4\pi} \sin\theta \, d\theta d\varphi$$
$$= 2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \underbrace{\frac{E}{4\pi} \cdot \frac{z}{(x^2+y^2+z^2)^{3/2}}}_{F(x,y,z)} dxdy$$

where the integrand is the flux F through an infinitesimal patch on a plane at distance z from the light source. This is not a contradiction to the commonly known $\frac{1}{d^2}$ dependence for the intensity of a point light source since the latter counts for beams perpendicular to the patch. If we rotate the patch by an angle α orthogonally to the incident beam, we have to multiply the intensity by $\cos\alpha = z/\sqrt{x^2+y^2+z^2}$ yielding the same result as above.

Note that F is an additive quantity, i.e. its value can be expressed by a sum over all light sources. We will use reduced units such that for a point p on the stage at distances $d(p,s)$ from each light source $s \in S$ we have a requirement of 1 while assuming the supply is expressed in the form $F(p) = \sum_{s \in S} \frac{x_s}{d^\delta(p,s)}$. We may choose $\delta = 2$, if the size

of the stage is small with respect to the distance of the light sources to the plane in which the stage is embedded, i.e. if $\cos\alpha$ is nearly 1 for all light sources. Otherwise, we set $\delta = 3$ and implement the distance z_s in the variable for the power x_s of each light source. Moreover, we scale all distances such that the minimal value of z_s is 1 for every s in S.

Remark: In case of an illumination problem, it is intuitive to actually add up the arriving energy from all light sources when considering some point $p \in L$. Unfortunately this cannot easily be interpreted in the context of a guarding problem. If there are two guarding cameras watching for example an expensive painting in a museum, but due to their distance and limited resolution, each of the cameras can only tell with 50% confidence whether there is someone near the painting, this does not mean that by using both cameras, one can tell with 100% confidence what is happening near the painting. So, individual 'confidence ratings' do not simply add up; a reasonable model could in this concrete example assign a confidence rate of 75%, since some information might be gained by using both cameras instead of only one. Note though, that the attenuation model introduced does make sense also in this interpretation. If an object doubles its distance to the camera, it covers only a quarter of a digital camera's CCD sensor or film emulsion. So in the above attenuation model, we would have an exponent of $\delta = 2$.

Very recently, Bilò et. al. in Ref. [BCKK05] have presented an $(1+\varepsilon)$-approximate solution for the variant of the problem where individual energies or confidences for a point do not add up, but rather only the maximum energy or confidence is considered. Here the goal is to cover all points of a stage subject to the constraint that for each point the maximum energy arriving from a single light source is above some threshhold. This variant of the problem probably has more applications than the one considered in this Chapter, e.g., when designing wireless networks. Our results can be viewed as the fractional version of the ones presented in Ref. [BCKK05].

1.3 Algorithms in $\mathbb{R}^2$

In this section we propose several ways to solve the problem in $\mathbb{R}^2$. The approaches will employ less and less heavy machinery, starting with a convex programming formulation, going over a combination of discretization and linear programming, to finally presenting a very simple combinatorial algorithm. In spite of the derivation of an attenuation exponent of $\delta = 3$ in the previous section, we will assume in the following any exponent $\delta \geq 2$, i.e. a point p receives a $\frac{1}{|ps|^\delta}$ fraction of the light emitted from light source s. To allow for a simpler presentation, most calculations and proofs will be in terms of $\delta = 2$, though generalization for larger (but constant) values of δ are straightforward.

Before presenting these algorithms we first make a simple observation which allows us to reduce the number of light sources that have to be considered for the following steps.

1.3.1 Pruning light sources

In this part we show that under the assumption that light sources can be assigned arbitrarily high powers, only certain light sources are of interest for our problem. Namely, we show that all light sources whose Voronoi cell does not intersect the stage L can be replaced by light sources whose Voronoi cells do, without incurring a larger cost in terms of the overall power used (see Figure 1.2). Let us state this claim more formally

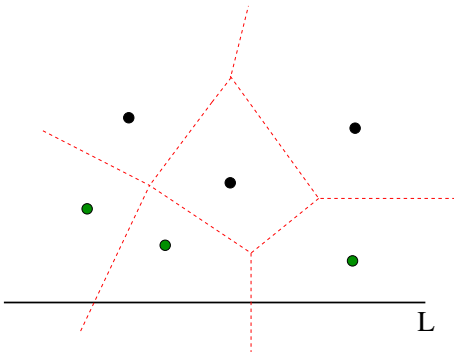

Figure 1.2: *Only lights whose Voronoi cells intersect the stage are useful.*

in the following lemma.

Lemma 1.1 *Consider the order of the light sources in S induced by the vertical projection on the line supporting the stage L. Let $s \in S$ be some light source whose Voronoi cell does not intersect the stage, $s_L, s_R \in S$ be the first neighbors to the left and right in this ordering whose Voronoi cells intersect the stage.*

Then there always exists a power assignment x_{s_L} and x_{s_R} such that for any $p \in L$

$$\frac{x_{s_L}}{|s_L p|^2} + \frac{x_{s_R}}{|s_R p|^2} \geq \frac{x_s}{|sp|^2} \tag{1.1}$$

and $x_{s_L} + x_{s_R} \leq x_s$.

Proof:

In the following we will exhibit a power assignment x_{s_L}, x_{s_R} with $x_{s_L} + x_{s_R} = x_s$. Thus, we can express x_{s_L} and x_{s_R} as $x_{s_L} = \alpha \cdot x_s$ and $x_{s_R} = (1-\alpha) \cdot x_s$ for some nonnegative $\alpha \leq 1$. So we can rewrite (1.1) as

$$\frac{\alpha}{|s_L p|^2} + \frac{(1-\alpha)}{|s_R p|^2} \geq \frac{1}{|sp|^2}, \quad \alpha \leq 1. \tag{1.2}$$

Now the goal is to show that there exists an $\alpha \leq 1$ independent of the position of the point $p \in L$ and such that inequality (1.2) holds.

Let v denote the intersection of the Voronoi edge between s_L and s_R and the stage L. Note that we can always move the light-source s perpendicularly toward the stage until $|s_L v| = |sv| = |s_R v|$ (see Figure 1.3) since this is the worst case scenario for the claim of

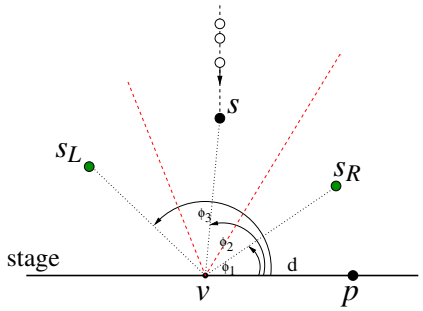

Figure 1.3: *Light source moved towards the stage until $|sv| = |s_L v| = |s_R v|$.*

the lemma. In the case when p lies to the left of v, $|s_L p| < |sp| < |s_R p|$ and analogously when p lies to the right of v, $|s_L p| > |sp| > |s_R p|$. For the sake of simplicity suppose $|s_L v| = |sv| = |s_R v| = 1$ and let $d = |vp|$, $\phi_1 = \angle p v s_R$, $\phi_2 = \angle pvs$ and $\phi_3 = \angle p v s_L$. We can express the distances $|s_L p|$, $|sp|$ and $|s_R p|$ in (1.2) with help of the law of cosines and obtain:

$$\frac{\alpha}{1 + d^2 \pm 2d\cos\phi_3} + \frac{(1-\alpha)}{1 + d^2 \pm 2d\cos\phi_1}$$

$$\geq \frac{1}{1 + d^2 \pm 2d\cos\phi_2}$$

and hence

$$\alpha \cdot \left(\frac{\pm(\cos\phi_1 - \cos\phi_3)}{|s_L p|^2}\right)$$

$$\geq \frac{\pm(\cos\phi_1 - \cos\phi_2)}{|sp|^2} \tag{1.3}$$

were '+' holds if p lies to the left of v and '-' if p lies to the right of v. Choosing $\alpha = \frac{\cos\phi_1 - \cos\phi_2}{\cos\phi_1 - \cos\phi_3} \leq 1$ for $0 \leq \phi_1 < \phi_2 < \phi_3 \leq \pi$ and keeping in mind that $|s_L p| > |sp|$ if p lies to the right of v and $|s_L p| < |sp|$ if p lies to the left of v, one can easily verify inequality (1.3) and therefore conclude the proof of the lemma. □

1.3.2 A convex programming formulation

The following convex program clearly solves our problem:

$$\begin{array}{lll} \min & \sum_{s\in S} x_s & \\ s.t. \quad \forall p \in l: & \sum_{s\in S} x_s / d^{\delta}(p,s) & \geq 1 \\ & x_s & \geq 0 \end{array} \tag{1.4}$$

Here the second line exactly expresses the constraint that for every point p on the stage, when summed over all light sources, 'enough' light should arrive at p. If light source s is powered up with x_s, the fraction of light arriving at p is proportional to $1/d^{\delta}(p,s)$. Here δ is the attenuation exponent as derived to be $\delta = 3$ in the previous section or $\delta = 2$ as commonly used. Note, that in this formulation one could also incorporate *upper bounds* on the light intensity. Later we will refer to this convex program when considering only a finite number of constraints (and hence being a linear program) as *lighting LP*.

This formulation is not a linear program since the number of constraints is (uncountably) infinite, so in fact our setting is an optimization problem over a convex body rather than a simple linear program. There are numerous algorithms for optimizing over a convex body, most of which rely on an efficient method of determining whether a point $x \in \mathbb{R}^n$ is contained in the convex body, which in our case basically reduces to determining whether some degree n polynomial has a root. In the following we describe the method in detail for $\delta = 2$. The following notions can be found in Ref. [GLSv88]. Let $K \subseteq \mathbb{R}^n$ be a convex body, $\varepsilon > 0$. The set $S(K,\varepsilon)$ is the set of points which have at most distance ε from K, $S(K,\varepsilon) = \{x \in \mathbb{R}^n \mid \|x-y\| \leq \varepsilon \text{ for some } y \in K\}$. The set $S(K,-\varepsilon)$ is the set of points in K whose ε-environment is completely contained in K, $S(K,-\varepsilon) = \{x \in K \mid \|x-y\| \leq \varepsilon \text{ implies that } y \in K \text{ for all } y \in \mathbb{R}^n\}$.

We now recall the definitions of the weak membership and optimization problem over convex bodies, see Ref. [GLSv88].

Definition 1.1 *The* weak membership problem *for K is the following:*

Given a vector $x \in \mathbb{Q}^n$ and a rational number $\delta > 0$, either

1. *assert that $x \in S(K,\delta)$, or*

2. *assert that* $x \notin S(K, -\delta)$.

Definition 1.2 *The* weak optimization problem *for K is the following: Given a vector $c \in \mathbb{Q}^n$ and a rational number $\varepsilon > 0$, either*

1. *find a vector $x^* \in \mathbb{Q}^n$ such that $x^* \in S(K, \varepsilon)$ and $c^T x \leq c^T x^* + \varepsilon$ for all $x \in S(K, -\varepsilon)$, or*
2. *assert that $S(K, -\varepsilon)$ is empty.*

Grötschel, Lovász and Schrijver (Corollary (4.3.12) in Ref. [GLSv88]) prove the following theorem.

Theorem 1.1 *There exists an oracle polynomial time algorithm that solves the weak optimization problem for every convex body K, given by a weak membership oracle, where the convex body contains a ball of radius r around a point a_0 and is contained in a ball or radius R around* 0.

Polynomial time here means, polynomial in the dimension n and the binary encoding lengths of c, ε, a_0, r and R.

Observe that the feasible region of the system (1.4) is not bounded. However, we can easily compute a bounding parameter M, such that an optimal solution is contained in $0 \leq x \leq M$. We simply let M be the largest power value, which has to be assigned to a single light source in order to lighten the stage by itself. If we then impose the additional constraint $0 \leq x \leq 2M$ to the system (1.4), the convex set is bounded and contained in the ball around 0 with radius $2Mn$ and contains the ball around $M\mathbf{1}$ with radius M. In the following we denote the set of feasible solutions by K.

Next we show that the weak membership problem for a power assignment x' can be solved in polynomial time. For this we assume that the light sources are located on the Euclidean plane with nonnegative component in the y-axis and that the stage is the interval $[-L, L]$ on the x-axis. Suppose we are given a power assignment x'. The exact membership problem is to decide whether there exists a point $(p, 0)$ on the stage such that $\sum_{s \in S} x_s / \left((p - X_s)^2 + Y_s^2\right) < 1$ holds.

We solve the weak membership problem for any $\varepsilon > 0$ in the following way. We decide whether there exists a $p \in [-L, L]$ such that $\sum_{s \in S} x_s / \left((p - X_s)^2 + Y_s^2\right) = 1$ holds. If yes, we can assert that $x' \notin S(K, -\delta)$ for each $\delta > 0$. Otherwise, we determine whether $\sum_{s \in S} x_s / \left((L - X_s)^2 + Y_s^2\right) > 1$ holds. If yes, we can assert that $x' \in K$. If not, we can assert that $x' \notin K$.

Thus we can solve the weak membership problem for K if we can determine in polynomial time, whether there exists a $p \in [-L, L]$, such that the following holds:

$$\sum_{s \in S} x_s / \left((p - X_s)^2 + Y_s^2\right) = 1 \tag{1.5}$$

Eq. (1.5) can be written as $f(p) = 0$, where $f(p)$ is a rational polynomial, whose binary encoding length is polynomial in the encoding length of the positions of the light sources. The problem now reads as follows. Given a polynomial $f(p) \in \mathbb{Q}(p)$ and an integer L, determine, whether $f(p)$ has a root in $[-L, L]$. This can be done in polynomial time, after $f(p)$ is decomposed into squarefree factors, with the method of *Sturm*, see Ref. [vG99], p. 87. So the weak optimization problem (1.4) can be solved in polynomial time in the *encoding length* of the light source placements and the *encoding length* of the error parameter ε.

Theorem 1.2 *Given a set S of light sources in the Euclidean plane and an* $\varepsilon > 0$*, one can compute a feasible point* x^* *for the optimization problem (1.4) in polynomial time such that* $\sum_{s \in S} x_s^* \leq \sum_{s \in S} \bar{x}_s + \varepsilon$ *for any feasible* $\bar{x}$*.*

Thus the most energy efficient illumination can be approximated with an *additive error* $\varepsilon > 0$ in polynomial time.

1.3.3 A $(1+\varepsilon)$-approximation scheme

One obvious approach to obtain an approximation to our problem is to discretize the stage using a finite number of *guards*[1], solve the linear program with constraints only for the guards and then power up all light sources sufficiently such that all points on the stage which were not 'guarded' by a constraint for sure also get enough light. The efficiency of this approach depends on the choice of a suitable discretization which allows for few guards but still requires only a moderate 'power up' of the light sources to guarantee sufficient overall coverage.

Definition 1.3 *For every point* $p \in L$*, we define the function* $ens(p)$ *– the* empty neighborhood size *– to be the distance to the closest light source, i.e.* $ens(p) = \min_{s \in S} d(p, s)$.

The following observation is not hard to see since the function ens(.) is defined to be the minimum over some distance functions.

Observation 1.1 *Function* $ens(.)$ *is* 1-Lipschitz*, that is* $ens(p) \leq ens(q) + 1 \cdot |pq|$.

Our discretization will now be based upon the empty neighborhood size, in particular we will have more guards in areas where ens(.) is small and fewer guards in areas where ens(.) is large. Similar discretization approaches occur in several other places in literature: For example, Amenta et al. (see Ref. [ABE98]) use the so-called *local feature size* to classify discrete samples from a continuous surface. Papadimitriou and

[1] Note that in the following we use the term *guard* as a point on the stage that ensures sufficient lighting at that point. That notion differs from the use of *guard* in other work in that area, where the guard is a point which covers or watches the scene.

Aleksandrov et al. also use a related discretization for the purpose of shortest path computations, see Ref. [Pap85] and Ref. [AMS00] for more details.

The crucial property for our set G of chosen guards is the following:

Definition 1.4 *A set $G \subset L$ of points satisfying*

$$\forall p \in L \; \exists g \in G : d(p,g) \leq \varepsilon \cdot ens(p)$$

is called a ε-good set of guards.

Note that assuming a minimum distance of 1 of each light source to the stage, it is trivial to obtain an ε-good set of guards of size D/ε by placing guards at equal distance ε all along the stage. Here D denotes the length of the stage L. In the following we will show that one can do considerably better.

Before we show that using an ε-good set G we can obtain a $(1+\varepsilon)$-approximation to our original problem, let us first convince ourselves that a reasonably small set G of guards exists. For that consider one light source s. Assuming that we have pruned the set of light sources as in the previous section, there is a point $p_0 \in L$ for which s is the closest light source. We start constructing a set G_s by first adding p_0 to G_s. We then extend G_s by adding guards $p_{-1} \in L$ (for first guard left of p_0) and $p_1 \in L$ (first guard right of p_o) at distance $2\varepsilon ens(p_0)$ from p_0. Iteratively we place the next guard p_{i+1} at distance $2\varepsilon ens(p_i)$ to the right of p_i (and accordingly $p_{-(i+1)}$ to the left of p_{-i}). We now claim the following:

Lemma 1.2 *The set G_s constructed above is a ε-good set of guards for the single light source s and furthermore $|G_s| = O(\frac{\log D}{\varepsilon})$ where D denotes the length of the stage.*

Proof: Assume the contrary, i.e. there exists a point $p \in L$ s.t. $\nexists p_i \in G_s$ with $d(p_i,p) \leq \varepsilon ens(p)$. W.l.o.g. assume p lies between p_i and p_{i+1} (the same argumentation holds when it lies between p_{-i} and p_{-i-1}). We have $\forall p'$ between p_i and p_{i+1}: $ens(p') \geq ens(p_i)$, since we are moving away from the light source. Furthermore we have clearly $\min\{d(p,p_i), d(p,p_{i+1})\} \leq |p_i p_{i+1}|/2$. But since $|p_i p_{i+1}| \leq 2\varepsilon ens(p_i)$ by construction we get $\min\{d(p,p_i), d(p,p_{i+1})\} \leq \varepsilon ens(p_i) \leq \varepsilon ens(p)$ which contradicts our assumption.

Let us now turn to the size of G_s. Look at the situation in Figure 1.4. Clearly for all $p \in \overline{P_l P_r}$ we have $ens(p) \geq d$, hence the distance between two adjacent guards between P_l and P_r is at least $2\varepsilon d$, hence there are $O(1/\varepsilon)$ guards placed at $P_l P_r$.

We are now interested in the guards outside $\overline{P_l P_r}$. We claim that for consecutive guards p_i, p_{i+1} we have $ens(p_{i+1}) \geq ens(p_i) \cdot (1+\varepsilon)$. This follows easily from the law of cosine since we have $ens(p_{i+1})^2 = ens(p_i)^2 + (2\varepsilon ens(p_i))^2 - 2ens(p_i) \cdot 2\varepsilon ens(p_i) \cos(\alpha) \geq ens(p_i)^2[1 + 4\varepsilon^2 + 2\varepsilon] \geq ens(p_i)^2(1+\varepsilon)^2$ where the last two inequalities follow from the fact that $\alpha \geq \frac{3}{4}\pi$. Hence the distance between adjacent guards outside $\overline{P_l P_r}$ grows

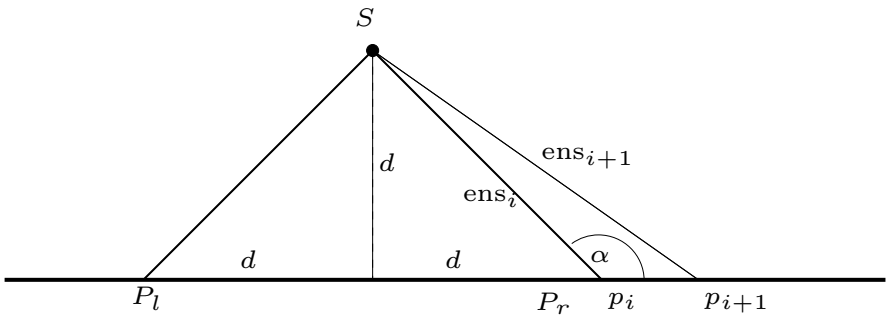

Figure 1.4: *Bounding the number of guards for S.*

at least by a factor of $a = (1+\varepsilon)$ in each iteration. We now establish an upper bound on the number of guards in terms of this factor a.

$$\begin{aligned} D &\geq 2\varepsilon \sum_{i=1}^{|G_s|} a^i = 2\varepsilon a \frac{a^{|G_s|}-1}{a-1} \\ \Rightarrow \quad a^{|G_s|} &\leq \frac{D}{2\varepsilon} \cdot \frac{a-1}{a} + 1 \\ \Rightarrow \quad |G_s| &\leq \frac{\log(\frac{D}{2\varepsilon} \cdot \frac{a-1}{a} + 1)}{\log a} \end{aligned}$$

Since the number of guards contained in $\overline{P_l P_r}$ is also $O(1/\varepsilon)$ we can conclude that the total number of guards generated by our procedure is $O(\frac{\log D}{\varepsilon})$ □

For a fixed length D of the stage, this estimation is tight as can be seen from Definition 1.4. It establishes an upper bound on the distance of two guards. Having a stage of length D and a light source at distance 1 from the stage, ens$(\cdot)$ is at most $\sqrt{1+D^2}$. Therefore, we have to partition the stage into at least $\Omega(\frac{1}{\varepsilon})$ parts.

Obtaining a set of ε-good guards G could be easily achieved by computing G_s for all light sources s and taking the union of those sets. It is clear that the resulting set is ε-good for the set of all light sources, since for any $p \in L$ there is a guard within distance $\varepsilon \cdot \text{ens}(p)$ in the set G_s where s is the light sources closest to p. The resulting union then contains $O(\frac{n}{\varepsilon} \log D)$ guards. We can do better though:

Lemma 1.3 *There exists a ε-good set of guards G of size $O(\frac{n}{\varepsilon} \log[1+\frac{D}{n}])$.*

Proof: We may consider each Voronoi cell of the light sources on its own since the ens$(\cdot)$ of its points on the stage are determined by their distance to this particular light source. The stage is partitioned into n pieces $D_1, \ldots, D_n$ corresponding to the respective Voronoi regions of the light sources. Let d_i denote the length of piece D_i, i.e. $\sum_{i=1}^n d_i = D$. For each piece D_i we construct the sample set as before and get overall $O((\sum_{i=1}^n \log[1+d_i])/\varepsilon)$ many guards. This sum is maximized when all parts have equal length, i.e. $d_i = \frac{D}{n}$. □

Note that this bound tends to $O(\frac{D}{\varepsilon})$ if n goes to infinity, since $n\log[1+\frac{D}{n}] = \log[1+\frac{D}{n}]^n \longrightarrow \log e^D$, i.e. in the limit the number of guards does not depend on n anymore, but only on D (but linearly, not in the logarithm).

The last lemma in this section shows that given a ε-good set of guards, we can obtain a $(1+\varepsilon)$-approximate solution to the lighting problem. But before proving that, we show a small auxiliary Lemma which gives an upper bound on the distance between two consecutive guards in a ε-good set of guards.

Lemma 1.4 *Let G be an ε-good set of guards, p and q two guards in G that appear consecutively on the stage. Then $|pq| \leq \frac{2\varepsilon}{1-\varepsilon} ens(p)$.*

Proof: Let $z \in L$ such that $|pz| = |zq|$. By the definition 1.4 we know that $|zq| \leq \varepsilon \cdot \text{ens}(z)$. Using the 1-Lipschitz property of the local feature size $\text{ens}(z)$ we can write down $|pq| = 2|zq| \leq 2\varepsilon \cdot (\text{ens}(q) + |pq|/2)$ which in turn implies the claim of the lemma. □

Lemma 1.5 *Let $\{x_s\}$ be an optimal solution of the lighting LP (1.4) with respect to an ε-good set of guards G. Then powering up every light source by a factor $(1+6\varepsilon)$ ensures that* every *point on the stage receives enough light.*

Proof: For the power assignment $\{x_s\}$ we know that for all $p' \in G$, the lighting constraints are fulfilled, i.e. they receive enough light. Consider some point $p \in L$, $p \notin G$. Let $p' \in G$ be the closest guard to p, hence $|pp'| \leq \frac{\varepsilon}{1-\varepsilon}\text{ens}(p')$ according to Lemma 1.4. We want to show that after powering up all light sources by a sufficiently large factor $\psi = 1 + O(\varepsilon)$, p also receives enough light. Namely, we are looking for ψ such that

$$\sum_{s\in S} \frac{\psi \cdot x_s}{(|sp'| + \text{ens}(p') \cdot \varepsilon/(1-\varepsilon))^2} \geq 1 \tag{1.6}$$

Observe that all light sources have distance at least $\text{ens}(p')$ to p' just by definition of $\text{ens}(\cdot)$ and keeping in mind that p' receives enough light, inequality (1.6) holds if ψ is chosen such that:

$$\sum_{s\in S} \frac{\psi \cdot x_s}{|sp'|^2(1+\varepsilon/(1-\varepsilon) \cdot \underbrace{\text{ens}(p')/|sp'|}_{\leq 1})^2} \geq \frac{\psi}{(1+\varepsilon/(1-\varepsilon))^2} \geq 1$$

Therefore, for $\varepsilon < 1/2$, powering up all light sources by a factor of $\psi = (1+2\varepsilon)^2 < 1+6\varepsilon$ makes sure that p receives at least as much light as p' received before powering up all light sources. □

We summarize by stating the main theorem of this part:

Theorem 1.3 *Given a stage of length D and a set of light sources S where each light source has at least unit distance from the stage, one can compute a power assignment*

$\{x_s\}_{s\in S}$ such that each point on the stage receives at least one unit of light and $\sum x_s \leq (1+\varepsilon)\sum x_s^{\text{opt}}$ where x_s^{opt} denotes an optimal power assignment. $\{x_s\}_{s\in S}$ can be found in polynomial time by solving a linear program with $O(\frac{n}{\varepsilon}\log\frac{D}{n})$ constraints and n variables.

1.3.4 Pruning guards – a simple $O(1)$-approximation algorithm

Even though the previous section provided a rather simple $(1+\varepsilon)$-approximation algorithm for our problem, it relied on solving a linear program which – in spite of being polynomial-time – is still quite time-consuming. Furthermore there was still a – even though only logarithmic – dependence on the length D of the stage. In the following we will propose a very simple $O(1)$-approximation algorithm that can be easily implemented to run in $O(n^2)$ time.

Similarly to the previous section we will first relax our problem by restricting to a small – here $O(n)$ size – set of guards. This set is chosen such that any solution for this reduced set transfers to a solution for the original problem incurring only a $O(1)$ overhead in terms of the quality of the solution.

Consider the function ens$(.)$ on the stage L (see Figure 1.5). This continuous function

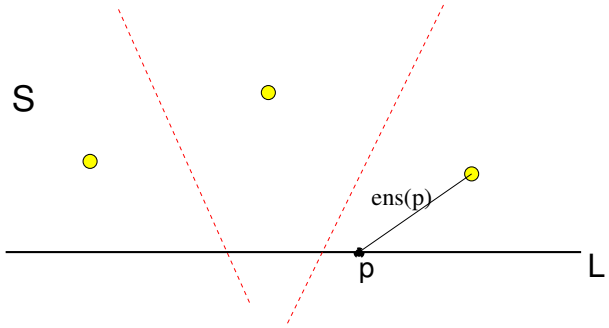

Figure 1.5: *For every $p \in L$, empty neighborhood size of p is Euclidean distance to its closest lightsource.*

consists of several arcs, each corresponding to one light source and their respective Voronoi cell. ens$(.)$ is differentiable except for the positions where two adjacent arcs are joined, that is at the boundary between two Voronoi cells. ens$(.)$ has local maxima at all the intersection points between Voronoi edges of $\mathcal{V}(S)$ and L, and potentially at the endpoints of L – depending on the location of the left-most and right-most light source.

Lemma 1.6 *Let G_V be the set of guards consisting of all points p of the stage L where ens$(.)$ has a local maximum. Furthermore let x_V^* be a feasible power assignment to the light sources S w.r.t. the set G_V of guards.*

Then $x^* = 4 \cdot x_V^*$ is a valid power assignment w.r.t. to all points on the stage.

Proof: Consider any point $p \in L$, $p \notin G_V$. Let $p_V \in G_V$ be a guard such that p is contained in the circle centered at p_V with radius $\text{ens}(p_V)$. Such a guard p_V always exists since each point on the stage is contained at least in one of the Voronoi circles around its left and right neighbors in the set G_V. Obviously all light sources have distance at least $\text{ens}(p_V)$ to p_V. But on the other hand we have $|pp_V| \leq \text{ens}(p_V)$ by choice of p_V. Therefore all light sources satisfying p_V's demand are at most a factor 2 further away from p, hence powering up all light sources by a factor of 4 ensures that p receives a sufficient amount of light. □

An immediate consequence is the following corollary:

Corollary 1.1 *A 4-approximation to the lighting problem can be obtained by solving the lighting LP consisting of $n+1$ constraints.*

In other words, if we are only aiming for a $O(1)$-approximation, we can obtain a solution in time independent of the length of our stage D (remember in case of the $(1+\varepsilon)$-approximation we had a $\log D$ dependence on the length of the stage).

In the following we will work on this set of guards G_V as defined above. Essentially we order them according to decreasing $\text{ens}(.)$ and one-by-one increase the power of their respective nearest light source such that they all get satisfied. By a dual fitting argument we then show that the used amount of power does not exceed a constant times the optimum.

Our analysis relies on a special property of the set of guards, namely we want that the density of the guards is proportional to the local value of $\text{ens}(.)$, in particular we want the distance between two adjacent guards g_i, g_j on the stage L to be lower bounded by $|g_i g_j| \geq C \cdot \max\{\text{ens}(g_i), \text{ens}(g_j)\}$ for some constant $C > 0$. This need not be the case in general, e.g. consider a set of light sources on a line parallel to but far away from the stage. Hence we need to prune the set of guards beforehand to ensure this property.

Pruning Guards

Let $\alpha > 0$ be some constant. Then the following algorithm prunes a set of guards G_V to a set G_P:

1. Compute for each guards p_i its empty neighborhood size $\text{ens}(p_i)$
2. Sort the guards in decreasing order of $\text{ens}(p_i)$, i.e. $\text{ens}(p_1) \geq \text{ens}(p_2) \geq \ldots \text{ens}(p_n)$
3. for $i = 1 \ldots n$
   - if p_i has not been removed yet, remove all guards p_j at distance $\leq \alpha \cdot \text{ens}(p_i)$ (but not p_i itself)
4. return the set of guards that have not been removed as G_P

In the following we will show that for constant α, even after this pruning step, we can obtain a $O(1)$-approximation using the pruned set of guards.

Lemma 1.7 *Let x_P^* be a feasible assignment of powers to the light sources such that all guards in G_P are satisfied. Then $x_V^* = (1+\alpha)^2 \cdot x_P^*$ is a valid power assignment for the set of guards $G_V \supseteq G_P$.*

Proof: Let $p_i \in G_V, \notin G_P$ be a guard that has been removed during the pruning step, $p_j \in G_P$ the guard responsible for the removal. Then we have $|p_i p_j| \leq \alpha \text{ens}(p_j)$, and hence powering up all light sources by a factor of $(1+\alpha)^2$ ensures that p_i receives enough light due to the same reasoning as in Lemma 1.6. □

Furthermore, our desired property is obviously fulfilled:

Lemma 1.8 *For any two guards in the pruned set $p_i, p_j \in G_P$, we have $|p_i p_j| \geq \alpha \max\{ens(p_i), ens(p_j)\}$.*

Proof: Assume otherwise, then either p_i or p_j would have been pruned away when considering the other guard. □

An immediate corollary of Lemma 1.7 is the following:

Corollary 1.2 *A $4(1+\alpha)^2$-approximation to the lighting problem can be obtained by solving the lighting LP w.r.t. to the pruned set of guards G_P.*

It is now time to describe the algorithm which we will use to derive a power assignment to the light sources. For that let us denote by s_i the light source that is closest to guard p_i, x_i its assigned power for all guards $p_i \in G_P$. Without loss of generality we assume that no light source is the closest for more than one guard (our derived bounds only get better if we remove this assumption). The algorithm works as follows:

1. Compute the set of guards G_V (via the Voronoi diagram of S)
2. Prune the set of guards G_V with pruning constant α to obtain G_P, $|G_P| = m$.
3. Let G_P be ordered such that $\text{ens}(p_1) \geq \text{ens}(p_2) \geq \cdots \geq \text{ens}(p_m)$
4. for all $i = 1 \ldots m$
   - $x_i = \max\{0, |p_i s_i|^2 \cdot \left(1 - \sum_{j=1}^{i-1} \frac{x_j}{|s_j p_i|^2}\right)\}$

Informally speaking this algorithm takes the guards one-by-one in decreasing order of their ens(.) value and increases the power of their closest light source just sufficiently such that they receive enough light. It can be trivially implemented to run in $O(n^2)$ time.

The crux of the analysis will be to show that no guard receives more than a constant amount of excessive light. This property will then allow us to use a dual fitting argument bounding the quality of our solution.

Let P_i be the amount of light experienced by some guard p_i after the execution of the algorithm. Let us write this as $P_i = P_i^< + P_i^= + P_i^>$ where $P_i^<$ denotes the power received from light sources s_j, $j < i$, $P_i^=$ the power received from light source s_i and $P_i^>$ the power received from light sources s_j, $j > i$. Clearly $P_i^= > 0 \Leftrightarrow P_i^< < 1$, that is s_i will only be used if p_i did not already receive enough light from light sources which were switched on before in the course of the algorithm. In the following we will bound $P_i^>$ and $P_i^<$ and show that they are at most some constant ($P_i^= \leq 1$ is obvious).

Lemma 1.9 $P_i^> \leq 4$

Proof: Assume w.l.o.g. that all guards p_j, $j > i$ lie to the right of p_i (at the end we simply multiply the obtained bound by 2 to obtain a bound for all p_j). We have

$$P_i^> = \sum_{j=i+1}^{m} \frac{x_j}{|p_i s_j|^2} \leq \sum_{j=i+1}^{m} \frac{\mathrm{ens}(p_j)^2}{((\sum_{l=i+1}^{j} \alpha \cdot \mathrm{ens}(p_{l-1})) - \mathrm{ens}(p_j))^2}$$

following from Lemma 1.8 and since each light source s_j is at most powered up to $\mathrm{ens}(p_j)^2$. Assuming $\alpha \geq 2$ we can continue with

$$\leq \sum_{j=i+1}^{m} \frac{\mathrm{ens}(p_{j-1})^2}{(\sum_{l=i}^{j-1} \mathrm{ens}(p_l))^2}$$

But this sum is of the form $\sum_{i=1}^{n} \frac{\delta_i^2}{(\sum_{j=1}^{i} \delta_j)^2}$ with $\delta_i \geq \delta_{i+1}$ (new indices here !). We then get

$$\sum_{i=1}^{n} \frac{\delta_i^2}{(\sum_{j=1}^{i} \delta_j)^2} \leq \sum_{i=1}^{n} \frac{\delta_i^2}{(i\delta_i)^2} \leq \frac{\pi^2}{6} \leq 2$$

by our decreasing ordering of the δ_i.

□

Furthermore we have for the energy collected from the light sources assigned previously in the course of the algorithm:

Lemma 1.10 $P_i^< \leq 6$.

Proof: Consider the first guard p_j, $j < i$ to the left of p_i whose light source is switched on, i.e. $x_j > 0$. Clearly we have $P_j^= + P_j^< = 1$ by definition of the algorithm. Furthermore due to the previous Lemma we know that p_j receives at most 2 units of light from the left from light sources $k > j$. Hence at most 3 units of light can arrive at p_i from the left, which makes 6 units of light overall considering both the contributions from the left and the right. □

From these two Lemmas and the observation that $P_i^= \leq 1$ we can derive the following

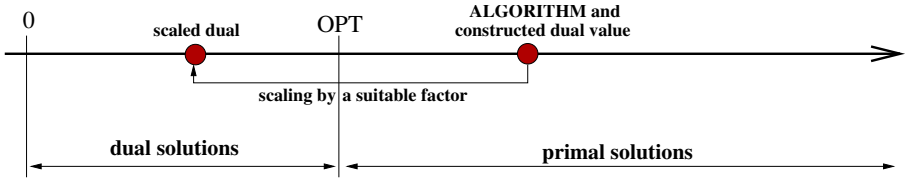

Figure 1.6: *Scaling the dual by a suitable (constant) factor gives the approximation guarantee for the combinatorial algorithm.*

Corollary 1.3 *If powers are assigned to the light sources according to our algorithm, we have for every guard $p_i \in G_p$:* $1 \leq P_i = P_i^{<} + P_i^{=} + P_i^{>} \leq 4+1+6 = 11$.

which says that any guard receives between 1 and 11 units of light.

Bounding the Quality of the Solution

In the following we will argue with the help of dual fitting that the solution x^* obtained with respect to the pruned set of guards G_P is almost optimal, i.e. only a constant factor away from the optimal solution (w.r.t. G_P). Recall that x^* can be easily extended to a feasible solution for the whole stage incurring an additional cost factor of at most $4 \cdot (1+\alpha)^2$ according to Corollary 1.2. The method of dual fitting, assuming the minimization problem as in our case, relies on the linear programming and its dual. The basic idea is to interpret the combinatorial algorithm as an algorithm that iteratively makes primal and dual updates such that the primal feasible solution computed at the end by the algorithm is upper bounded by that of the constructed dual[2]. Since the dual assignment is not feasible, the main step in the analysis consists of scaling the dual by a suitable (constant) factor and showing that the scaled dual is feasible. The scaled dual is then a lower bound on OPT, and the scaling factor gives the approximation guarantee for the algorithm (see Fig. 1.6).

Let us rewrite the linear program w.r.t. the pruned set of guards G_P and light sources S_P:

$$\begin{array}{lll} \min & \sum_{s \in S_P} x_s & \\ s.t. & \forall p \in G_P: \quad \sum_{s \in S_P} x_s / d^2(p,s) & \geq 1 \\ & x_s & \geq 0 \end{array} \tag{1.7}$$

The dual of this program looks as follows:

[2] In our case the objective values will be the same

$$\begin{array}{lll} \max & \sum_{p\in G_P} y_p & \\ s.t. \quad \forall s \in S_P: & \sum_{p\in G_P} y_p/d^2(p,s) & \leq 1 \\ & y_p & \geq 0 \end{array} \tag{1.8}$$

The interpretation of the dual is the following: assign weights y_p to each guard $p \in G_p$ such that for each light source, the 'influence' of the guards does not exceed 1.

To show that the power assignment constructed by our algorithm is not too far off the optimum, it suffices to exhibit a feasible solution to the dual program which has about the same objective function value. Weak duality then tells us that the optimum solution to the primal program is sandwiched between the solution of our algorithm and the feasible dual solution. Making use of the fact that the distance between a light source s_i and a guard p_j is essentially the same as the distance between light source s_j and p_i (after the α-pruning), we can use the amount of light arriving at each guard p_i as value for y_i and after scaling by a constant factor obtain a feasible solution to the dual program.

Lemma 1.11 *For any light source s_i and guard p_j in the pruned set of guards and light sources G_P and S_P we have:*

$$|s_j p_i| \cdot (1 - \frac{2}{\alpha - 1}) \leq |s_i p_j| \leq |s_j p_i| \cdot (1 + \frac{2}{\alpha - 1})$$

Proof: We show the right inequality, the left works analogously. We have by triangle inequality $|s_i p_j| \leq |p_i p_j| + \text{ens}(p_i) \leq |s_j p_i| + \text{ens}(p_i) + \text{ens}(p_j) \leq |s_j p_i| + 2E$ for $E = \max\{\text{ens}(p_i)\text{ens}(p_j)\}$. But since $E \leq \frac{|s_j p_i| + E}{\alpha}$ we get after rearranging $E \leq \frac{|s_j p_i|}{\alpha - 1}$ which yields the desired bound. □

In other words, for $\alpha \geq 3$ the distances $|s_i p_j|$ and $|s_j p_i|$ can differ by at most a factor of two.

Lemma 1.12 *Let x^* be the solution to the primal LP (1.7) as computed by our algorithm, c_p^* its objective function value. Then there exists a feasible solution y^* to the dual LP (1.8) with objective value $c_d^* \geq c_p^*/44$.*

Proof: Let us set for every $p_i \in G_P$: $y_i^* = x_i^*/44$. We need to verify that $\forall s \in S_P$: $\sum_{p\in G_P} y_p^*/|ps|^2 \leq 1$. But according to Corollary 1.3 and together with Lemma 1.11 we have for $\alpha \geq 3$:

$$\sum_{i=1}^{m} \frac{y_i^*}{|s_j p_i|^2} \leq \frac{2^2}{44} \left(\sum_{i=1}^{m} \frac{x_i^*}{|s_i p_j|^2} \right) \leq 1$$

□

So we have established a dual feasible solution with function value at least $c_p^*/44$, i.e. the optimum value c_{opt} must lie between $c_p^*/44$ and c_p^* which implies that x_p^* is no worse than a 44-approximation for the LP w.r.t. the pruned set of guards. And because an optimum solution w.r.t. the pruned set of guards can be extended to a feasible solution for the original problem at a cost of an additional $4 \cdot (1+\alpha)^2$ factor, we conclude with the following main theorem of this section:

Theorem 1.4 *Given a stage L and n light sources, one can compute in $O(n^2)$ time a power assignment to the light sources such that any point on the stage receives at least one unit of light. The solution produced requires at most $O(1)$ times the optimal amount of energy.*

Questions

It is not clear whether the pruning is indeed necessary for the analysis of the algorithm. As the experiments later on show, even without pruning the algorithm achieves a rather good approximation ratio, so it might be possible that the pruning was only necessary due to our inability to give a more precise analysis.

1.4 Generalizations and Open Problems

In the following we will list some extensions and other open illumination problems that are worth analysing within our light attenuation model.

1.4.1 Generalization to higher dimensions ($\mathbb{R}^3$)

There is a straightforward way of extending the described model to a 3-dimensional setting by assuming the stage L to be some bounded two-dimensional surface patch in $\mathbb{R}^3$. The definition of ε-good sampling (Def. 1.4) can still be used, and ε-good sample sets can be derived in a similar manner as described for the 2-dimensional case. Their sizes then depend on the area of the two-dimensional surface to be sampled (again, a logarithmic instead of linear dependence is achievable). The LP-based solution strategy can still be applied, whereas the constant approximation probably requires some more work.

1.4.2 Open Problems

There is a vast number of variations of the basic illumination and guarding problems that have been considered in the past. Many of them can also be considered in our light attenuation model, for example:

Art Gallery Illumination with k light sources

Given some fixed number k, determine position and power assignments of k light sources such that any point on the boundary (or also in the interior) of a polygon with n vertices receives at least one unit of light. Minimize the sum of assigned powers.

Floodlight Illumination

Given a stage L and a set $F = \{f_1, \ldots, f_n\}$ of floodlights of angle sizes $\alpha_1, \ldots, \alpha_n$ such that their apexes are located at some fixed points on the plane, all on the same side of L. Decide if it possible to rotate them such that every point on L is illuminated, and if yes, determine the rotations and power assignments, such that the stage is sufficiently illuminated at every point and the overall power assigned is minimized.

Stage Illumination with Obstacles

The same problem as considered in this Chapter can be examined also in the presence of obstacles. In this case, neither the pruning of light sources nor the discretization can be applied immediately, though some similar approach seems doable.

1.5 Experimental Results

In this experimental section we want to investigate the actual behavior of the proposed algorithms, since we believe that our analysis of the approximation ratios is overly pessimistic. We have implemented the LP-based approximation algorithms using a ε-good set of guards and the local maxima of the ens$(.)$ function, as well as the combinatorial $O(1)$-approximation algorithm based on the pruned set of Voronoi vertices. We have run the algorithms for different lengths D of the stage also varying the number of relevant (in the sense of Section 1.3.1) light sources (which were randomly generated around the stage [3]).

Performance according to the Analysis

In Table 1.1 we have listed the sum of the power assignments made by our algorithms for varying values of D and $|G_V|$ (the number of light sources whose Voronoi cells actually intersect the stage). Columns $LP_{1+\varepsilon}$ ($\varepsilon = 0.01$), LP_V^*, and C_0^* contain the results for the $(1+\varepsilon)$-, 4-, and $O(1)$-approximation algorithms, *including* the power-up of all light sources to satisfy all points on the stage as proved in the previous sections.

	$LP_{1+\varepsilon}$	LP_V^*	C_0^*
$D = 70$, $\|G_V\| = 6$	390.97	1547.96	39776
$D = 70$, $\|G_V\| = 15$	106.03	416.04	11671.44
$D = 70$, $\|G_V\| = 22$	82.28	324.12	8883.60
$D = 140$, $\|G_V\| = 8$	690.88	2507	66584.32
$D = 140$, $\|G_V\| = 19$	314.59	1227.44	33846.56
$D = 140$, $\|G_V\| = 41$	172.40	675.96	20138.80

Table 1.1: *Energy costs for different stage lengths (D), number of guards ($|G_V|$), and algorithms (LP-based $(1+\varepsilon)$-approximation, LP-based 4-approximation, combinatorial $O(1)$-approximation).*

Not surprisingly, the differences in the approximation ratios of the latter two algorithms are dominated by the 'power-up' factor. Note though, that this completely disregards to which degree some point on the stage gets insufficient light. This will be taken into account in our next experiment.

Actual Performance

To assess the real amount of 'power-up' required to satisfy all points on the stage, we evaluated the power assignments on a ε-good set of guards with $\varepsilon = 0.01$ and determined the 'worst' guard, i.e. the guard that received the least amount of light. We

[3] We have not investigated how to generate worst-case instances, so our results are not meant to be a rigorous experimental analysis.

D, $\lvert G_V\rvert$	$LP_{1+\varepsilon}$	LP'_V	C'_0	APX$_V$	APX$_{\text{COMB}}$
70, 6	414.43	561.13	1071.24	1.43	2.73
70, 15	112.40	139.37	167.11	1.31	1.57
70, 22	87.22	102.10	162.52	1.24	1.97
140, 8	732.34	814.775	1112.26	1.11	1.51
140, 19	333.47	527.79	615.39	1.58	1.95
140, 41	182.75	239.96	299.79	1.39	1.74

Table 1.2: *Energy costs for different stage lengths, number of guards, and algorithms, but with adaptive power-up.*

D, $\lvert G_V\rvert$	$\alpha = 0$	$\alpha = 1$	$\alpha = 3$	$\alpha = 5$
70, 14	3.39	3.31	3.19	3.16
70, 20	2.75	2.76	1.54	1.51
70, 31	5.09	5.09	4.98	3.92
140, 21	6.81	7.03	1.75	1.74
140, 30	10.15	10.06	9.98	9.88
140, 65	5.45	5.40	2.64	2.54

Table 1.3: *Maximum excess of light for different stage lengths, number of guards, and values of* α.

use this to compute an appropriate power-up factor (for all light sources; one could improve here by powering up light sources locally only). The results can be found in Table 1.2; here LP'_V and C'_0 denote the power assignments resulting from this more careful power-up strategy for the 4- and the $O(1)$-approximation scheme.

We have also included in the last two columns the resulting approximation factor of the solutions (taking the $(1+\varepsilon)$ solution with $\varepsilon = 0.01$ as a lower bound). It turns out that in fact, by using this more refined power-up strategy, the combinatorial $O(1)$ as well as the 4-approximation get much closer to the optimum solution than guaranteed by the pessimistic theoretical analysis.

Further Observations

In the proof of the approximation ratio of the $O(1)$ algorithm, we employed a pruning procedure with some parameter α to actually be able to bound the amount of excess light at any guard. But we suspected that this was only necessary because of our inability to come up with a better analysis. In Table 1.3, the worst amount of excess light at any guard is stated for different values of α. It seems that even without pruning (i.e. $\alpha = 0$) the amount of excessive light is bounded by a small constant; in our proof we were only able to bound it for $\alpha \geq 3$.

In a last experiment we have run the simple combinatorial algorithm on a ε-good set

D, $\lvert G_V \rvert$	$LP_{1+\varepsilon}$	C_0^*	$C_{\text{SIMPLE}}^{1+\varepsilon}$	$\text{APX}_{\text{SIMPLE}}^{1+\varepsilon}$
70, 6	565.28	54175.6	613.63	1.08
70, 16	130.94	14386.3	180.66	1.38
70, 32	55.77	6106.18	72.90	1.31
140, 15	581.27	63846.1	684.78	1.18
140, 38	235.43	37361.8	322.78	1.37
140, 54	117.95	13528	154.83	1.31

Table 1.4: *Energy costs of the LP-based* $(1+\varepsilon)$*-approximation, the combinatorial algorithm as described, and the combinatorial algorithm using a* ε*-good set of guards. The last column shows resulting approximation factor of* $C_{\text{SIMPLE}}^{1+\varepsilon}$ *outcome.*

of sample points. Even though we cannot prove any better approximation ratio than for the original algorithm, the results look quite promising as can be seen in Table 1.4. Here we denote by $C_{\text{SIMPLE}}^{1+\varepsilon}$ the outcome of running the $O(1)$ algorithm on an ε-good set of guards, including the required power-up.

1.6 Conclusions

In this Chapter we have introduced a light attenuation model under which the large class of illumination problems can be considered. Our model also takes into account the decrease of light intensity with distance – something that had not been regarded in classical models for illumination problems. As a concrete example we have examined the problem of illuminating a stage using a fixed set of light sources minimizing the overall energy.

Although we showed that the problem can be approximated up to the arbitrary small additive error (see Section 1.3.2), the exact version of the problem is not known to be in NP.

Looking at our solutions, we believe this new model creates quite a number of new and interesting open questions in the context of illumination problems for which only a combination of geometric reasoning and techniques from combinatorial optimization will lead to provable results.

Acknowledgements

We want to thank Arno Eigenwillig for helpful comments and suggestions regarding Section 1.3.2.

Part II

Efficient Geometric Queries using WSPD

Chapter 2

Preliminaries

In this chapter we will introduce the so-called *well-separated pair decomposition* (WSPD) due to Callahan and Kosaraju ([CK92], [CK95a], [CK95b]).

The overall picture of the decomposition is as follows: given a set of n points in d-dimensional Euclidean space, if the dimension d is fixed one can cluster points in *linearly* many cluster-pairs such that

i) for every two points, there exists a (unique) cluster-pair separating them,

ii) the distance between two points from different sets/clusters can be 'approximated' by the distance between the two clusters,

iii) points belonging to the same cluster are 'close', as compared to points in the opposite clusters.

In other words, a WSPD can be seen as a compressed representation to approximate n^2 pairwise distances of n points.

Above properties make the WSPD a powerful geometric concept that has been shown to be of use in many practical and theoretical problems. In fact, later on in Chapter 3 and Chapter 4 we present query data structures based on the WSPD that would allow for fast and efficient queries for two different geometric problems, the k-hop restricted shortest path problem and the cone restricted nearest neighbor problem.

2.1 The Well-Separated Pair Decomposition

Let P be a set of n points in $\mathbb{R}^d$. For two sets $A, B \subseteq P$, let $d(A,B)$ denotes the distance between the centers of their minimum enclosing boxes $R(A)$ and $R(B)$, respectively. By $r(A), r(B)$ we denote the radii of the minimum enclosing balls C_A, C_B of $R(A), R(B)$, respectively.

Definition 2.1 *A pair of point sets A and B are said to be* **well-separated** *if and only if $d(A,B) > s \cdot r$, for an arbitrary positive constant s and where r denotes the radius of the larger of the two minimum enclosing balls C_A, C_B of $R(A), R(B)$ (see Fig. 2.1).*

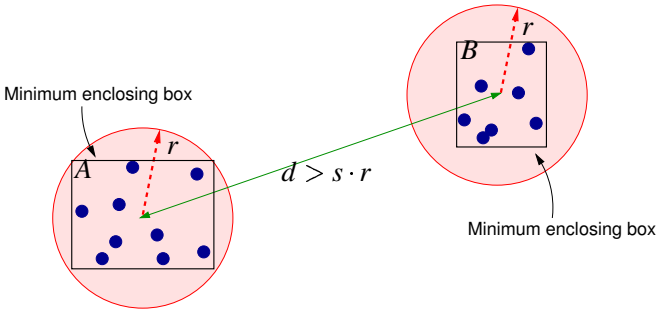

Figure 2.1: *Clusters A and B are 'well-separated' if $d > s \cdot r$.*

s is called a *separation constant* and, in order to keep the minimum enclosing balls C_A, C_B disjoint, s will never be less than 2. Furthermore, the definition of two sets being well-separated captures some very nice properties. More precisely, for $a, a' \in A$ and $b, b' \in B$ we make the following two observations:

Observation 1 Points belonging to the same cluster are 'close', as compared to points in the opposite clusters.

$$|aa'| < 2r \leq \frac{2}{s} \cdot d(A,B) \leq \frac{2}{s} \cdot (|ab| + 2r) \leq \frac{4}{s} \cdot |ab|, \text{ for } s \geq 4$$

Observation 2 All distances between points in opposite clusters are 'almost equal'.

$$|a'b'| \leq |a'a| + |ab| + |bb'| < (1 + \frac{8}{s}) \cdot |ab|$$

Definition 2.2 (WSPD) *A well-separated pair decomposition of P for a given parameter $s \geq 2$ is a sequence $(A_1, B_1), \ldots, (A_m, B_m)$, where $A_i, B_i \subseteq P$ such that*

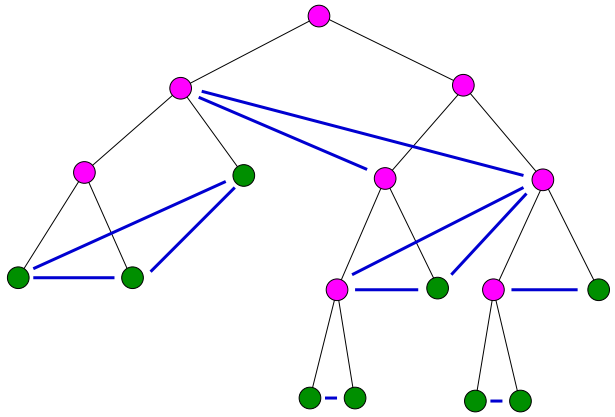

Figure 2.2: *Example of split tree with additional blue edges.*

1. *A_i, B_i are well-separated with respect to separation constant s, for all $1 \leq i \leq m$,*
2. *for all $p \neq q$ in P there exists a unique pair (A_i, B_i) such that $p \in A_i, q \in B_i$ or $q \in A_i, p \in B_i$.*

The first question we could ask ourself is whether such a decomposition exists at all. The answer is 'yes', since we could simply use all singleton pairs $(\{a\},\{b\})$. This would yield a sequence of $m = \Theta(n^2)$ length. The big question is whether we could do better than that. Namely, can we construct a WSPD of P that will have e.g. linearly many pairs? The answer to this question is given by the following theorem due to Callahan and Kosaraju ([CK92],[CK95a]):

Theorem 2.1 *Given a set P of n points in $\mathbb{R}^d$ and a separation constant $s > 2$, a WSPD of P with $O(s^d d^{d/2} n)$ many pairs can be computed in $O(dn \log n + s^d d^{d/2} n)$ time.*

Corollary 2.1 *For a fixed separation constant s and dimension d, a WSPD in d-dimensional Euclidean space of size $O(n)$ can be computed in $O(n \log n)$ time.*

The rest of the section will be devoted to the proof of the Theorem 2.1 that follows the main ideas from Callahan and Kosaraju ([CK92]) paper.

Construction of WSPD: Let $L_i(R(A))$ denote the ith side of the minimum bounding box of $A \subseteq P$ (in the rest of the section we will simplify notation and instead of $L_i(R(A))$ usually just write $L_i(A)$). Furthermore, $L_{\max}(A)$ ($L_{\min}(A)$) will denote the largest (smallest) dimension of the minimum bounding box $R(A)$. The *split-tree* of a set P of points in $\mathbb{R}^d$ is the tree constructed by the following recursive algorithm:

SplitTree(P)

1. if size(P)=1 then return leaf(P),

2. partition P into sets P_1 and P_2 by halving its minimum enclosing box $R(P)$ with hyperplane along $L_{\max}(P)$ side,

3. return a node with children (SplitTree(P_1), SplitTree(P_2)).

Although such a tree might have linear depth and therefore a naive construction as above takes quadratic time, Callahan and Kosaraju in [CK92] have shown how to construct such a binary tree in $O(n \log n)$ time.

With every node u of that tree we can conceptually associate the set P_u of all points contained in its subtree as well as their minimum enclosing box $R(P_u)$. In order to obtain the WSPD of P with respect to the separation constant s, for each internal node u of the split tree with children v, w (and v_l, v_r and w_l, w_r denoting left and right child of v, w) invoke the following procedure:

FINDPAIRS(v, w)

if P_v and P_w are well-separated with respect to s
then add additional *blue* edge (v, w) to the tree.
else if $L_{\max}(P_v) > L_{\max}(P_w)$
then FINDPAIRS(v_l, w), FINDPAIRS(v_r, w)
else FINDPAIRS(v, w_l), FINDPAIRS(v, w_r)

Note that the additional *blue* edges for the split tree are added such that

- the point sets associated with the endpoints of a blue edge are well-separated with respect to separation constant s,

- for any pair of leaves (a, b), there exists exactly one blue edge that connects two nodes on the paths from a and b to their lowest common ancestor $lca(a, b)$ in the split tree.

Thus, for a given split tree, the FINDPAIRS procedure terminates (e.g. in the worst case, the blue edges will always connect the pairs of leaves of the tree) and computes the WSPD represented as an additional set of edges, called *blue* edges, of the split tree.

Analysis: What is left to argue about is the size of the WSPD (number of blue edges). The main idea is to argue that every node in the tree can be incident to only constant number of blue edges. Since the size of the split tree is $O(n)$ this will immediately imply that the WSPD is of $O(n)$ size.

Let a and b are two nodes in the split tree. Suppose that (a, b) is the blue edge since P_a and P_b are well-separated after FINDPAIRS(u, v) call (see Figure 2.3). But then either $P_{\pi(a)}, P_b$ or $P_a, P_{\pi(b)}$ are by construction not well-separated, where $\pi(\cdot)$ denotes parent node. Without loss of generality, suppose $P_{\pi(a)}, P_b$ are not well-separated. Then the following Lemma should be easy to see:

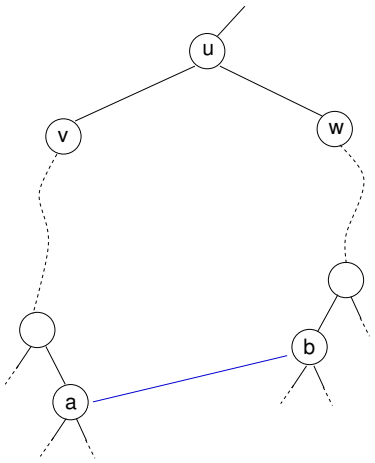

Figure 2.3: *Since P_a and P_b are well-separated, blue edge (a,b) is reported.*

Lemma 2.1 *Tests made by* FINDPAIRS(u,v) *imply*

$$L_{\max}(b) \leq L_{\max}(\pi(a)) \leq L_{\max}(\pi(b)).$$

Proof: Note that left-hand side of the inequality holds trivially. For the right-hand side of the inequality observe that either $L_{\max}(\pi(a))$ or some of the predecessors on the path from v to $\pi(a)$ have to be less or equal than $L_{\max}(\pi(b))$. □

Imagine balls of radius $\frac{\sqrt{d}}{2}L_{\max}(\pi(a)) := r$ around centers x and y of $R(\pi(a))$ and $R(b)$, respectively. Note that $R(b)$ fits in ball of radius r because of left-hand side inequality in Lemma 2.1. Furthermore, since $P_{\pi(a)}$ and P_b are not well-separated we have that

$$d(P_{\pi(a)}, P_b) = |xy| < s \cdot r,$$

which in turn directly implies the following Lemma:

Lemma 2.2 *For a given node a in the split tree, let $(a,b_1),\ldots,(a,b_k)$ denote the sequence of blue edges computed by* FINDPAIRS *routine and y_i center of $R(\pi(y_i))$, $1 \leq i \leq k$. Then each y_i has to be contained in hypercube of size $2s \cdot r$ centered at x. Furthermore, each $R(B_i)$ is completely contained in hypercube of size $4s \cdot r$.*

Note that each $R(b_i), R(b_j)$ are disjoint, for $i \neq j$, since separated by hyperplanes. Thus, in order to bound k (number of blue edges incident to a) we will use a packing argument. For this, we need a lower bound of the volume of $R(b_i)$. Since that doesn't really seem possible, we define an outer box (container) $R_0(u)$ of $R(u)$ and lower bound its volume. Namely, for each node u of the split tree the splitting hyperplane along the $L_{\max}(u)$ defines two outer boxes $R_0(v), R_0(w)$ corresponding to its leaves u and v such that $R_0(v) \cup R_0(w) = R(u)$ (see Figure 2.4). Furthermore, we assume that the outer box of the root is a minimal hypercube enclosing all the points in P.

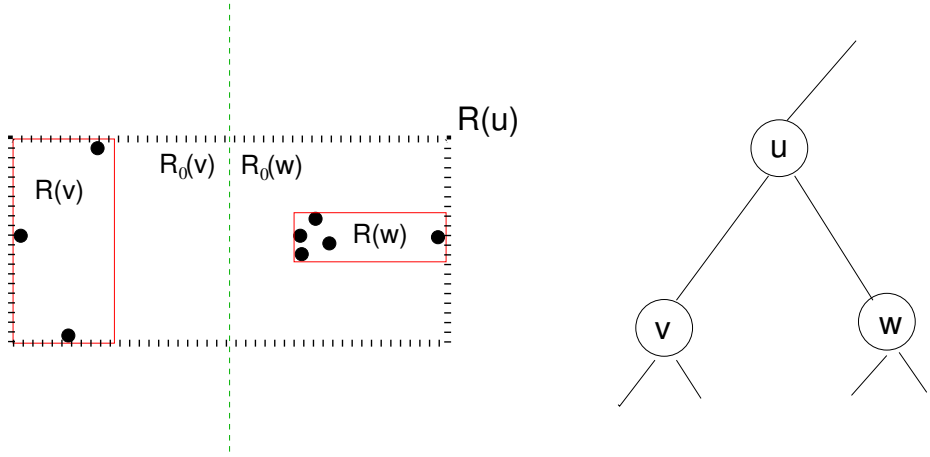

Figure 2.4: *Example of the containers $R_0(v)$ and $R_0(w)$ of $R(v)$ and $R(w)$.*

Lemma 2.3 *For every node $b \neq$ root of the split tree, we have that*

$$L_{\min}(R_0(b)) \geq \frac{1}{2} L_{\max}(\pi(b)).$$

Proof: Proof goes by induction on the level of the tree.

Suppose $\pi(b) =$ root. Since the outer box of the root is a minimal hypercube enclosing P, $L_{\min}(R_0(b)) = \frac{1}{2} L_i(R_0(\text{root}))$ for $1 \leq i \leq d$. Since $L_i(R_0(\text{root})) = L_{\max}(R(\text{root}))$ the claim follows.

If $\pi(b) \neq$ root, we distinguish between two cases:

CASE 1: $L_{\min}(R_0(b)) = L_{\min}(R_0(\pi(b)))$. But then, by induction, we have that

$$L_{\min}(R_0(\pi(b))) \geq \frac{1}{2} L_{\max}(\pi(\pi(b))) \geq \frac{1}{2} L_{\max}(\pi(b))$$

.

CASE 2: $L_{\min}(R_0(b)) < L_{\min}(R_0(\pi(b)))$. But then the splitting of $R(\pi(b))$ must have happened along the minimum dimension – say ith dimension – of the $R_0(b)$. But then,

$$L_{\min}(R_0(b)) = L_i(R_0(b)) = \frac{1}{2} L_i(\pi(b)).$$

Since the ith dimension was the splitting one, $L_i(\pi(b)) = L_{\max}(\pi(b))$ which completes the proof. □

Since, $R_0(b_i), R_0(b_j)$ are also disjoint for the same reason (separated by a hyperplane), we will try to pack as many as possible of them in the hypercube of size $4s \cdot r$.

Note that the volume of the hypercube, call it U, of size $4s \cdot r$ is $(4s \cdot r)^d = (4s \cdot \frac{\sqrt{d}}{2} L_{\max}(\pi(a)))^d := \text{Vol}(U)$. Furthermore, by Lemma 2.1 we have that $\text{Vol}(U) \leq$

$(4s \cdot \frac{\sqrt{d}}{2} L_{\max}(\pi(b)))^d$. Number of $R_0(b_i)$, for $1 \leq i \leq k$, lying completely inside the hypercube U is then at most

$$\frac{\left(4s \cdot \frac{\sqrt{d}}{2} L_{\max}(\pi(b))\right)^d}{\left(\frac{1}{2} L_{\max}(\pi(b))\right)^d} = O(s^d d^{d/2}). \tag{2.1}$$

We can conclude that the number of blue edges incident to any node in the split tree is bounded by $O(s^d d^{d/2})$ which concludes the proof of the Theorem 2.1.

The split tree together with its additional blue edges is what we will usually refer to as *well-separated pair decomposition*.

Chapter 3

Approximate k-hop Restricted Shortest Paths in Power-Euclidean Networks

The *shortest-path* problem in networks is a well studied fundamental problem for which an efficient solution is well-known. The best-known and most commonly used shortest path algorithm is that of Dijkstra [Dij59], which solves the single-source shortest paths problem for directed graph with n nodes and m edges and non-negative edge weights in $O(m+n\log n)$ time. However, better solutions exist if we allow precomputation and construction of a data structure that would allow for *approximate shortest paths* queries instead. For a constant $t > 1$, we say that a path $\pi(p,q)$ from p to q is a t-approximate shortest path if and only if $\omega(\pi(p,q)) \leq \omega(\pi'(p,q)) \cdot t$, for any other path $\pi'(p,q)$ from p to q and function $\omega(\cdot)$ outputs the length/distance of the path. Not insisting on the *exact* shortest path/distance computation has allowed for drastic improvement in the running time. For general undirected graphs and for any integer $r \geq 1$, Thorup and Zwick [TZ01] show how to construct a distance oracle which answers $(2r-1)$-approximate queries in $O(r)$ time using a data structure that takes $O(rmn^{1/r})$ (expected) time and essentially optimal $O(rn^{1+1/r})$ space. They also show that the $(2r-1)$-approximate paths can be produced in constant time per edge. Since the query time is essentially bounded by a constant (if r is constant), Thorup and Zwick also refer to their queries as approximate distance oracles. Their algorithms are also simple and easy to implement efficiently.

In this work, we consider a following variant of the shortest path problem: Given a set P of n points in $\mathbb{R}^2$ we define the *Power-Euclidean network* as the complete graph $G = (P, P \times P)$ with edge weight

$$\omega(p,q) = |pq|^{\delta} + C_p \tag{3.1}$$

for some constant $\delta > 1$, nonnegative offset cost C_p and $p,q \in P$, where $|pq|$ denotes the Euclidean distance between p and q. We define the k-hop restricted shortest path,

Figure 3.1: *Example of very common and compact wireless units.*

for some positive integer k, to be a shortest path between two given points in the Power-Euclidean network subject to the constraint that at most k edges are used in the path.

This work focuses on constructing a linear-size data structure from a given instance P based on a well-separated pair decomposition (see Theorem 2.1) such that the $(1+\varepsilon)$-approximate k-hop restricted shortest path in Power-Euclidean network can be retrieved in constant time, for arbitrary small values of $\varepsilon > 0$ and a constant k.

Motivation

The main motivation for our problem comes from an application in wireless networks. If a user, application or company wishes to make data pervasive and accessible then wireless networking is probably what they are looking for. Wireless communication links between computing devices have become standard. For example, using a mobile phone, receiving a message on a pager or checking an email from a PDA is a very common scenario in everyday life of most people. New extremely compact wireless units allow for small and cheap stations that can be deployed almost everywhere (see Figure 3.1). However, such wireless gadgets can be found in a variety of different places and their geometric location strongly determines the quality of wireless communication. Thus, the geometric aspect turns to be much more important for wireless communication than for the wired world.

A particularly interesting scenarios in wireless networking is to deploy a wireless network completely independent of any wired infrastructure. For example, suppose that any kind of fixed wired infrastructure, such as the Internet or LAN, is not available either because it may not be economically practical or physically possible to provide the necessary infrastructure. In such situations a collection of hosts with wireless network interfaces may form a temporal network without the aid of any established

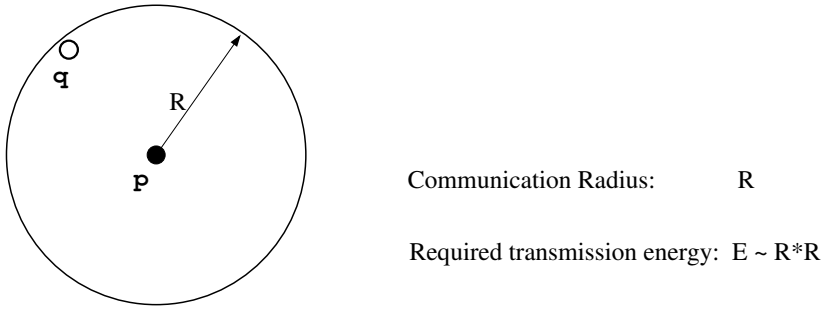

Figure 3.2: *Station p is able to send a message to other stations within its communication range R. The energy required is proportional to the communication area.*

infrastructure. *Radio Networks* are an attractive way to quickly build a communication infrastructure without slow and expensive deployment of a cable backbone. In such a network the finite set of radio stations are distributed over some geographical area. Every station consists of the radio transmitter/receiver pairs and a limited power supply. Communication takes place by a station broadcasting an omnidirectional signal over a fixed range, the size of which is proportional to the power expanded by the station's transmitter. Any other station within this range can directly (e.g. by one hop) receive messages from the sending station (e.g. in Figure 3.2, station q is within the communication range of p).

The energy needed by the station p in order to send a message to some other station q is mathematically modelled with the weight function (3.1). Exponent values of δ are typically between 2 and 6. Namely, for $\delta = 2$ the edge weights reflect the exact energy requirement for free space communication. For larger values of δ we get a popular heuristic model for absorption effects [Rap95, Pat00]. The *offset cost* C_p accounts for the distance independent energy consumption of the wireless stations (e.g. the energy consumption of the signal processing during sending and receiving).

Communication between two stations that are not within their respective ranges can be achieved by multi-hop transmission, namely a station in the range of the sending station receives the message and forwards it to the next station and so on until the destination station is reached. However, even when two stations are within communication range, they still might prefer to use a multi-hop transmission in order to minimize the overall energy needed (see Figure 3.3). In fact, in our idealistic model we are going to assume that every station is within communication range of others and will be mainly concerned with using the least possible energy needed to transmit messages between two different stations. However, a drawback of wireless communications is that the QoS (Quality of Service) is not guaranteed. For example, multi-hop transmission with too many hops/stations will increase the probability of link failure as well as increase the latency of the communication (e.g. time needed for encoding and decoding signals). One possible solution to that is to limit the number of hops in the transmission to some small integer k. And, as we will see, limiting the number of hops to k helps to speed up the calculation of paths.

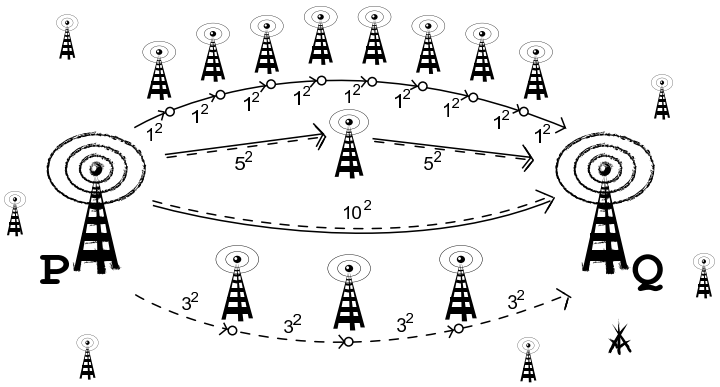

Figure 3.3: *A Radio Network example with* $9,4,2,1$*-hop paths from P to Q with costs 9, 36, 50, 100*

Hence, following the discussion above, computing the geometric shortest path from p to q under the weight function $\omega(p,q) = |pq|^{\delta} + C_p$ with bounded number of hops can be interpreted in radio networks as finding the energy optimal route between stations p and q.

Another application of k-hop shortest path computation comes from the area of economical airplane route planning. Chan, Efrat, and Har-Peled [EHP98, CE01] use similar concepts to model the fuel consumption of airplanes routed between a set P of airports as already discussed in the introduction of the thesis. The positive cost of the flight is usually mathematically described via a superlinear function, e.g. $\delta > 1$. The *offset cost* C_p in this case will also account for distance independent costs (e.g. take off phase will consume significantly more fuel compared to maintaining the constant altitude of the plane).

Related Work

The first attempt, by our knowledge, to cope with Dijkstra's quadratic complexity of the exact shortest path problem in a complete geometric graph with $\omega(p,q) = f(|pq|^{\delta})$ distance function is by Chan, Efrat, and Har-Peled [EHP98, CE01]. They observe that for $\omega(p,q)$ and $\delta \geq 2$, exact geometric shortest paths are equivalent to shortest paths in the Delaunay triangulation of P, i.e., optimal paths can be computed in time $O(n \log n)$. This is rather easy to see since a non-Delaunay edge e on a shortest path contains points inside the smallest circle enclosing the edge (replacing e by two edges going via such a point will yield a smaller cost for $\delta \geq 2$). Note that this approach completely collapses for k-hop paths because most Delaunay edges are very short. The main contribution of Chan et al. is a sophisticated $O(n^{4/3+\gamma})$ time algorithm for computing exact geometric shortest paths for monotone cost functions $\omega(p,q) = f(|pq|)$ where γ is any positive constant.

Beier, Sanders and Sivadasan [BSS02] give an algorithm which for the special case $\delta = 2$ computes the optimal k-hop path in time $O(kn \log n)$ by dynamic programming and applies geometric nearest neighbor search structures to speed up the update of the dynamic programming table. They also construct an $O(n^{(1+\varepsilon)})$ time algorithm for the case of an unrestricted number of hops. For two hop paths they furthermore reduce the query time to $O(\log n)$ using only linear space and $O(n \log n)$ preprocessing time. Their algorithms are based on geometric data structures ranging from simple 2-dimensional Delaunay triangulation to more sophisticated proximity data structures that exploit the special structure of the problem.

Our results

We start studying the k-hop restricted shortest path problem by presenting a very generic algorithm in Section 3.1 for computing an optimal solution for the problem.

In Section 3.2 and Section 3.3 we present our main result. Namely, we present a data structure that uses linear space and can be built in time $O(n \log n)$ for any constants k, $\delta > 1$, and $\varepsilon > 0$. In constant time it allows us to compute k-hop paths between arbitrary query points that are within a factor $(1+\varepsilon)$ far from optimal. When k, δ, and ε are considered variables, the query time remains constant and the preprocessing time is bounded by a polynomial in k, δ, and $1/\varepsilon$.

The algorithm has two main ingredients that are of independent interest. The first part, discussed in Section 3.2, is based on the observation that for approximately optimal paths it suffices to compute a shortest path for a constant size subset of the points — one point for each square cell in some grid that depends on the query points. This subset can be computed in time $O(\log n)$ using well known data structures supporting (approximate) quadratic range queries [dBvKOS97, AM00]. These data structures and in particular their space requirement are independent of k, δ, and ε. Some variants even allow insertion and deletion of points in $O(\log n)$ time. Section 3.3 discusses the second ingredient. Well separated pair decompositions [CK92] allow us to answer arbitrary approximate path queries by precomputing a linear number of queries. We develop a way to access these precomputed paths in constant time using hashing. This technique is independent of path queries and can be used for retrieving any kind of information stored in well separated pair decompositions. Since the query time is bounded by a constant, our result can be seen as an approximate *path oracle* for Power-Euclidean networks.

In Section 3.4 we experimentally study our algorithms and compare them to some simple heuristics. Section 3.5 discusses further generalizations and open problems.

REMARK For reasons of simplicity and clearness in the rest of the chapter we will consider the offset cost C_p to be zero. Note that our results stay applicable under more general weight function with a strictly positive offset cost C_p, though.

3.1 Computing Optimal k-hop Paths

We consider the following problem: Given a set P of n points in $\mathbb{Z}^2$ and some constant k, report for a given query pair of points $p,q \in P$, a polygonal path $\pi = \pi(p,q) = v_0v_1v_2\dots v_l$, with vertices $v_i \in P$ and $v_0 = p, v_l = q$ which consists of at most k segments, i.e. $l \leq k$, such that its weight $\omega(\pi) = \sum_{0\leq i<l} \omega(v_i, v_i+1)$ is minimized. By $\pi_{opt} = \pi_{opt}(p,q)$ we denote an optimal path from p to q under this criterion.

Lemma 3.1 *For the optimal path π_{opt} connecting p and q we have $\frac{|pq|^\delta}{k^{\delta-1}} \leq |\pi_{\text{opt}}| \leq |pq|^\delta$.*

Proof: As we can connect p and q directly using one hop, the upper bound follows immediately. For the lower bound observe that the cheapest way to connect p and q is to divide the segment pq into k subsegments of equal length, which yields the lower bound. □

Definition 3.1 *We define the axis-aligned square of side-length l centered at the midpoint of a segment pq as the* frame *of p and q, $F(pq,l)$.*

Lemma 3.2 *The optimal path π_{opt} connecting p and q has to be contained within the frame $F(pq, k^{(\delta-1)/\delta}|pq|)$ of p and q.*

Proof: Assume that the optimal path visits some point r outside the frame F. Note that such a path has Euclidean length at least $|pr| + |rq| > k^{(\delta-1)/\delta}|pq|$ and therefore the cost is lower bounded by $\frac{(|pr|+|rq|)^\delta}{k^{\delta-1}} > \frac{(k^{(\delta-1)/\delta}|pq|)^\delta}{k^{\delta-1}} = |pq|^\delta$ which in turn implies that doing one direct hop from p to q is even better. □

Layered Graph Construction

In the following we will present first a very generic algorithm for computing an exact k-hop shortest paths. Namely, the algorithm works for arbitrary connected graphs (not necessary geometric ones) and an arbitrary weight function.

We consider the complete graph with all edge weights explicitly stored and then use the following construction:

Lemma 3.3 *Given a connected graph $G(V,E)$ with $|V| = n$, $|E| = m$ with weights on the edges and one distinguished node $p \in V$, one can compute for all $q \in V - \{p\}$ the path of minimum weight between p and q using at most k edges in time $O(km)$.*

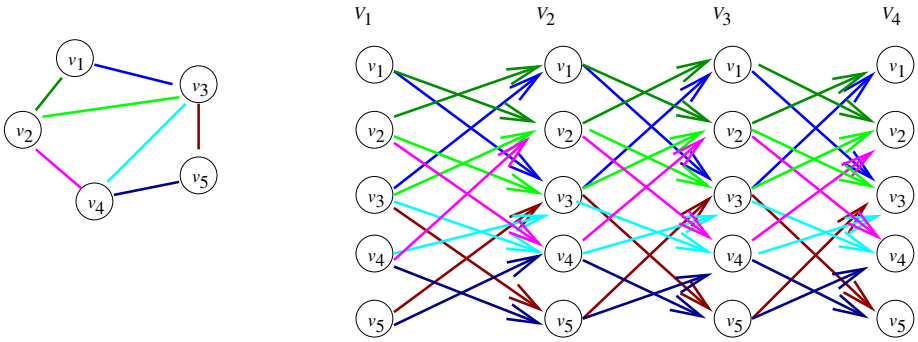

Figure 3.4: *Layered Graph example*

Proof: We assume that the graph G has self-loop edges (v,v) with assigned weight 0. Construct $k+1$ copies $V^{(0)},V^{(1)},\ldots,V^{(k)}$ of the vertex set V and draw a directed edge $(v^{(i)},w^{(i+1)})$ iff $(v,w)\in E$ with the same weight (Figure 3.4). Compute the distances from $p^{(0)}$ to all other nodes in this layered, acyclic graph. This takes time $O(km)$ as each edge is relaxed only once. □

Note that the above approach performs well in practice only if the set of points is very small. However, in practice we could speed up the above naive, essentially 'brute force', approach by figuring out edges that are too long to be part of the optimal solution for a given p and q (e.g. edges outside of the frame $F(pq,k^{(\delta-1)/\delta}|pq|)$). This simple heuristic will be discussed in more detail in experimental Section 3.4.

Recall that for the special case of $\delta=2$ there is an algorithm that computes the optimal k-hop shortest path in Power-Euclidean networks in time $O(kn\log n)$ (see [BSS02]).

3.2 Approximate k-hop Path Queries

In this section we will be concerned with computation of k-hop shortest path that is 'almost' optimal. Namely, for a given pair of query points p,q and an arbitrarily small constant $\varepsilon > 0$ our primal objective will be to output a polygonal path $\pi(p,q)$ such that its weight is at most $(1+\varepsilon)\cdot\omega(\pi_{opt})$.

In the following we assume that the weight function ω is of the form $\omega(a,b) = |ab|^{\delta}$ with $\delta > 1$ (the case $\delta \leq 1$ is trivial as we just need to connect p and q directly by one hop).

Preliminaries

Before we introduce our procedure for reporting approximate k-hop paths, we need to refer to some standard data structures from Computational Geometry which will be used in our algorithm.

Theorem 3.1 (Exact Range Query) *Given a set P of n points in $\mathbb{Z}^2$ one can build a data structure of size $O(n\log n)$ in time $O(n\log n)$ which for a given axis aligned query rectangle $R = [x_l,x_u]\times[y_l,y_u]$ reports in $O(\log n)$ time either that R contains no point or outputs a point $p \in P\cap R$.*

The data structure can be maintained dynamically such that points can be inserted and deleted in $O(\log n\log\log n)$ amortized time. The preprocessing time then increases to $O(n\log n\log\log n)$ and the query time to $O(\log n\log\log n)$. All the $\log\log n$ *factors can be removed if only* either *insertions* or *deletions are allowed.*

Proof: We use the standard 2-level range-tree construction. From the resulting $O(\log^2 n)$ query time we can get rid of one $\log n$ by fractional cascading, see [dBvKOS97]. The whole construction can be maintained dynamically allowing insertions and deletions using the results in [MN90]. □

In fact, the algorithm we will present will also work with an approximate range reporting data structure such as the one presented in [AM00, AMN+98]. The part of their result relevant for us can be stated in the following theorem:

Theorem 3.2 (Approximate Range Query) *Given a set P of n points in $\mathbb{Z}^2$ one can build a data structure of size $O(n)$ in time $O(n\log n)$ which for a given axis aligned query rectangle $R = [x_l,x_u]\times[y_l,y_u]$ with diameter ω reports in $O(\log n + \frac{1}{\alpha})$ time either that the rectangle $R' = [x_l+\alpha\omega, x_u+\alpha\omega]\times[y_l+\alpha\omega, y_u+\alpha\omega]$ contains no point or outputs a point $p \in P\cap R'$.*

The data structure can be maintained dynamically such that points can be inserted and deleted in $O(\log n)$ time.

Basically this approximate range searching data structure works well if the query rectangle is fat; and since our algorithm we present in the next section will only query square rectangular regions, all the results in [AM00] and [AMN+98] apply. In fact we do not even need α to be very small, $\alpha = 1$ turns out to be good enough. So the use of an approximate range searching data structure helps us to get rid of the $\log n$ factor in space and some $\log\log n$ factors for the dynamic version. But to keep presentation simple we will assume for the rest of this chapter that we have an exact range searching data structure at hand. Furthermore, let us briefly cite two well known inequalities that turn out to be of use in the following analysis.

Minkowski's inequality For some $\delta > 1$ and a_i, $b_i > 0$, Minkowski's sum inequality states that

$$\left(\sum_{i=1}^{k}(a_i+b_i)^{\delta}\right)^{\frac{1}{\delta}} \leq \left(\sum_{i=1}^{k}a_i^{\delta}\right)^{\frac{1}{\delta}} + \left(\sum_{i=1}^{k}b_i^{\delta}\right)^{\frac{1}{\delta}}.$$

Equality holds iff the sequences $a_1, a_2, \ldots$ and $b_1, b_2, \ldots$ are proportional.

Hölder's inequality Let

$$\frac{1}{p}+\frac{1}{q}=1$$

with $p, q > 1$. Then, for some a_i, $b_i > 0$ Hölder's inequality for sums states that

$$\sum_{i=1}^{k}a_i b_i \leq \left(\sum_{i=1}^{k}a_i^{p}\right)^{\frac{1}{p}}\left(\sum_{i=1}^{k}b_i^{q}\right)^{\frac{1}{q}}.$$

Equality holds when $b_i = ca_i^{p-1}$, for some $c > 0$.

Computing Approximate k-hop Paths

We will now focus on how to process a k-hop path query for a pair of points p and q assuming that we have already constructed the data structure for orthogonal range queries (which can be done in time $O(n\log n)$).

We are now armed to state our algorithm to compute a k-hop path which, as we will later show, is a $(1+\varepsilon)$ approximation to the optimal k-hop path from p to q.

K-HOP-QUERY(p, q, ε)

1. Put a grid of cell-width $\alpha \cdot |pq|/k$ on the frame $F(pq, k^{(\delta-1)/\delta}|pq|)$ with $\alpha = \frac{\ln(2)}{2\sqrt{2}} \cdot \frac{\varepsilon}{\delta}$.
2. For each grid cell C perform an orthogonal range query to either certify that the cell is empty or report one point inside which will serve as a representative for C.
3. Compute the optimal k-hop path $\pi(p, q)$ with respect to all representatives and $\{p, q\}$.

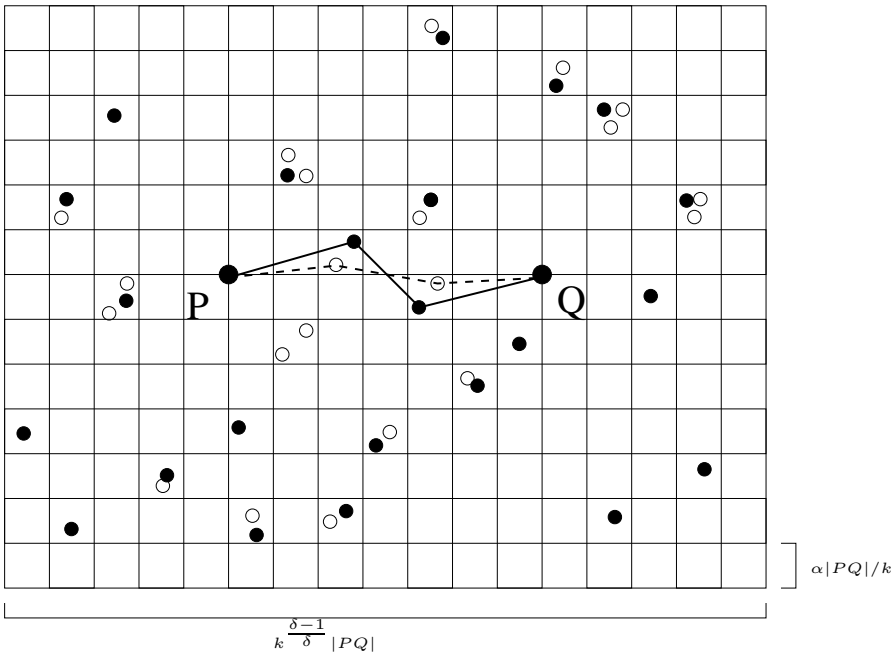

Figure 3.5: 3-*hop-query for P and Q: representatives for each cell are denoted as solid points, the optimal path is drawn dotted, the path computed by the algorithm is drawn solid*

4. Return $\pi(p,q)$

Please look at Figure 3.5 for a schematic drawing of how the algorithm computes the approximate k-hop path. It remains to argue about correctness and running time of our algorithm. Let us first consider its running time.

Lemma 3.4 K-HOP-QUERY(p, q, ε) *can be implemented to return a result in time* $O(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\varepsilon^2} \cdot T_R(n) + T_{k,\delta}(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\varepsilon^2}))$*, where* $T_R(n)$ *denotes the time for one 2-dimensional range query on the original set of n points and* $T_{k,\delta}(x)$ *denotes the time for the exact computation of a minimal k-hop path for one pair amongst x points under the weight function* $\omega(pq) = |pq|^\delta$.

Proof: By the choice of frame and cell width, it is easy to see that in the first step of our algorithm we generate a grid of size $O((\frac{\delta \cdot k^{(2\delta-1)/\delta}}{\varepsilon})^2)$ which is $O(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\varepsilon^2})$. For each of the cells we perform an orthogonal range query each of which takes $T_R(n)$ time. For all the representatives which we have found, we run an exact minimum k-hop path algorithm which takes $T_{k,\delta}(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\varepsilon^2})$. □

Let us now turn to the correctness of our algorithm, i.e. for any given positive ε, we want to show that our algorithm returns a k-hop path of weight at most $(1+\varepsilon)$ times the weight of the optimal path. We will show that only using the representatives of all the

grid cells there exists a path of at most this weight. In the following we assume that the optimal path π_{opt} consists of a sequence of points $p_0 p_1 \dots p_j$, $j \leq k$ and $l_i = |p_{i-1} p_i|$.

Before we get to the actual proof of this claim, we need to state a small technical lemma.

Lemma 3.5 *For any* $\delta > 1$ *and* $l_i, \xi > 0$ *the following inequality holds*

$$\frac{\sum_{i=1}^{k}(l_i+\xi)^\delta}{\sum_{i=1}^{k} l_i^\delta} \leq \left(\frac{\sum_{i=1}^{k}(l_i+\xi)}{\sum_{i=1}^{k} l_i}\right)^\delta$$

Proof: Note that Minkowski's inequality directly implies

$$\left(\sum_{i=1}^{k}(l_i+\xi)^\delta\right)^{\frac{1}{\delta}} \leq \left(\sum_{i=1}^{k} l_i^\delta\right)^{\frac{1}{\delta}} + \left(\sum_{i=1}^{k}\xi^\delta\right)^{\frac{1}{\delta}} = \left(\sum_{i=1}^{k} l_i^\delta\right)^{\frac{1}{\delta}} + k^{\frac{1}{\delta}} \cdot \xi$$

which in turn gives

$$\left(\frac{\sum_{i=1}^{k}(l_i+\xi)^\delta}{\sum_{i=1}^{k} l_i^\delta}\right)^{\frac{1}{\delta}} \leq 1 + \frac{k^{\frac{1}{\delta}} \cdot \xi}{(\sum_{i=1}^{k} l_i^\delta)^{\frac{1}{\delta}}}.$$

Moreover, note that Hölder's inequality implies

$$\left(\frac{\sum_{i=1}^{k} l_i^\delta}{k}\right)^{\frac{1}{\delta}} \geq \frac{\sum_{i=1}^{k} l_i}{k}$$

for $b_i = 1$, $i = 1, \dots, k$, $p = \delta$ and $q = \frac{\delta}{\delta-1}$, which completes the proof. □

Lemma 3.6 K-HOP-QUERY(p, q, ε) *computes a k-hop path from p to q of weight at most* $(1+\varepsilon)\omega(\pi_{\text{opt}}(p,q))$ *for* $0 < \varepsilon \leq 1$.

Proof: Using Lemma 3.5, with ξ denoting the possible error incurred by taking the respective representative edge, it is sufficient to show that

$$\left(\frac{\sum_{i=1}^{k}(l_i+\xi)}{\sum_{i=1}^{k} l_i}\right)^\delta \leq 1+\varepsilon.$$

Knowing that the absolute 'detour'[1] incurred by replacing its endpoints by the respective representatives in their grid cell is bounded by $\xi \leq 2\sqrt{2}\alpha\frac{d}{k}$ we have

$$\left(1 + \frac{k \cdot 2\sqrt{2}\alpha\frac{d}{k}}{\sum_{i=1}^{k} l_i}\right)^\delta \leq 1+\varepsilon.$$

[1] Here the analysis has to be changed slightly when using approximate range reporting data structures: for $\alpha = 1$ we might get twice the 'detour' assumed here.

Thus, observing that $(\sum_{i=1}^{k} l_i)/d$ is at least 1, we get following bound on α

$$\alpha \leq \frac{(1+\varepsilon)^{1/\delta}-1}{2\sqrt{2}}.$$

For any positive α less then the above upper bound claim of the lemma holds. However, we would like to have α as larger as possible in a given bound but also a little bit more nicely formulated. Hence, using the well known fact that $(1+1/x)^x \leq e$ for any $x > 0$, we get

$$(1+\varepsilon) = e^{\ln(1+\varepsilon)} \geq \left(\left(1+\frac{\ln(1+\varepsilon)}{\delta}\right)^{\frac{\delta}{\ln(1+\varepsilon)}}\right)^{\ln(1+\varepsilon)}$$

which easily reduces to

$$\delta \cdot \left((1+\varepsilon)^{1/\delta}-1\right) \geq \ln(1+\varepsilon).$$

Furthermore, $\ln(1+\varepsilon) \geq \ln(2)\cdot\varepsilon$ for any $0 < \varepsilon \leq 1$ and therefore it suffice to choose

$$\alpha = \frac{\ln(2)}{2\sqrt{2}} \cdot \frac{\varepsilon}{\delta}.$$

□

REMARK Note that in the case of the strictly positive offset cost C_p, the size of the frame containing the optimal path computed by Lemma 3.2 will stay the same. However, in the K-HOP-QUERY algorithm, one would have to be a little bit more careful and in step 2 of the algorithm report a point with smallest offset cost (this can be easily incorporated into the standard geometric range query data structures) rather then an arbitrary point inside the cell.

Summary

In our approximation algorithm we reduced the problem of computing an approximate k-hop path from p to q to one exact k-hop path computation of a small, i.e. constant number of points (only depending on k, δ and ε). Since with the help of Lemma 3.3 we know how to compute the k-hop optimal path, we can summarize our general result in the following Theorem (we give the bound for the case where an approximate range query data structure as mentioned in Theorem 3.2 is used).

Theorem 3.3 *We can construct a dynamic data structure allowing insertions and deletions with O(n) space and O(n log n) preprocessing time such that* $(1+\varepsilon)$ *approximate minimum k-hop path queries under the weight function* $\omega(p,q) = |pq|^{\delta}$ *can be answered in time O(*$\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\varepsilon^2} \cdot \log n + \frac{\delta^4 \cdot k^{(7\delta-2)/\delta}}{\varepsilon^4}$*).*

The query time does not change when using exact range query data structures, only space, preprocessing and update times get slightly worse (see Theorem 3.1).

In [BSS02] the authors presented an algorithm which for the special case $\delta = 2$ computes the optimal k-hop path in time $O(kn \log n)$ by dynamic programming and an application of geometric nearest neighbor search structures to speed up the update of the dynamic programming table. Applied to our problem we get the following corollary:

Corollary 3.1 *The subroutine of our algorithm to solve the exact k-hop problem on* $O(\frac{k^3}{\varepsilon^2})$ *points can be solved in time* $O(\frac{k^5}{\varepsilon^2} \cdot \log \frac{k}{\varepsilon})$ *if* $\delta = 2$.

3.3 Approximate Path Oracle

In the previous section we have seen how to answer a (p,q) query in $O(\log n)$ time (considering k, δ, ε as constants). Standard range query data structures were the only precomputed data structures used. Now we explain how additional precomputation can further reduce the query time. We show how to precompute a linear number of k-hop paths, such that for every (p,q), a slight modification of one of these precomputed paths is a $(1+\varepsilon)(1+2\psi)^2$ approximate k-hop path and such a path can be accessed in constant time. Here $\psi > 0$ is the error incurred by the use of these precomputed paths and can be chosen arbitrarily small (the size of the well-separated pair decomposition then grows, though).

3.3.1 Using the WSPD for Precomputing Path Templates

In fact the WSPD is exactly what we need to efficiently precompute k-hop paths for all possible $\Theta(n^2)$ path queries. So we will use the following preprocessing algorithm:

1. compute a well-separated pair decomposition of the point set with $s = k^{(\delta-1)/\delta} \cdot 8\delta \cdot \frac{1}{\psi}$
2. for each *blue* edge compute a $(1+\varepsilon)$-approximation to the lightest k-hop path between the centers of the associated bounding boxes

At query time, for a given query pair (p,q), it remains to find the unique blue edge (A,B) which links a node of the path from p to $lca(p,q)$ to a node of the path from q to $lca(p,q)$. We take the precomputed k-hop path associated with this blue edge, replace its first and last node by p and q respectively and return this modified path (see Figure 3.6).

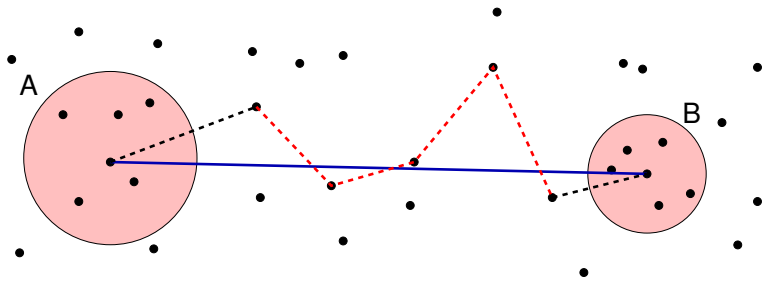

Figure 3.6: *Example of the blue-edge connecting well-separated sets A and B and the template path (red dashed line) saved with it.*

In the following we will show that the returned path is indeed a $(1+\varepsilon)(1+2\psi)^2$ approximation of the lightest k-hop path from p to q. Later we will also show that

this path can be found in constant time. For the remainder of this section let $\pi^P_{\text{opt}}(x,y)$ denote the optimal k-hop path between two points x,y not necessarily in P such that all hops have starting and end point in P (except for the first and last hop). We first start with a lemma which formalizes the intuition that the length of an optimal k-hop path does not change much when perturbing the query points slightly.

Lemma 3.7 *Given a set of points P and two pairs of points (a,b) and (a',b') with $d(a,b)=d$ and $d(a,a')\leq c$, $d(b,b')\leq c$ with $c\leq \frac{\psi d}{k^{(\delta-1)/\delta}\cdot 4\delta}$, then we have $\omega(\pi^P_{opt}(a',b'))\leq (1+2\psi)\omega(\pi^P_{opt}(a,b)$.*

Proof: Consider the path $\pi^P_{opt}(a,b)=v_0v_1\dots v_k$ with $v_0=a$, $v_k=b$ and consider its modification $\pi'=a'v_1\dots v_{k-1}b'$. For the analysis we will distinguish between the cases where the edge (a,v_1) ((v_{k-1},b) respectively) is 'long' or 'short'. If $(c+|av_1|)^\delta\leq (1+\psi)|av_1|^\delta$, we are safe and this is the case when $|av_1|$ is *long* edge, namely $|av_1|\geq \frac{c}{(1+\psi)^{1/\delta}-1}$. So let us now consider the case that (a,v_1) is short. Clearly an upper bound on the cost of $\pi^P_{opt}(a',b')$ is given by:

$$\begin{aligned}\omega(\pi^P_{opt}(a',b')) &\leq \omega(\pi')=\omega(\pi^P_{opt}(a,b))-|av_1|^\delta+|a'v_1|^\delta-|bv_{k-1}|^\delta+|b'v_{k-1}|^\delta\\ &\leq \omega(\pi^P_{opt}(a,b))-|av_1|^\delta+(|av_1|+c)^\delta-|bv_{k-1}|^\delta+(|bv_{k-1}|+c)^\delta\end{aligned}$$

And since for $0\leq x<y,\delta\geq 1$ we have $(c+x)^\delta-x^\delta\leq (c+y)^\delta-y^\delta$ and the respective edges are short, we can continue with

$$\begin{aligned}&\leq \omega(\pi^P_{opt}(a,b))+2\cdot((\frac{c}{(1+\psi)^{1/\delta}-1}+c)^\delta-(\frac{c}{(1+\psi)^{1/\delta}-1})^\delta)\\ &=\omega(\pi^P_{opt}(a,b))+2c^\delta\cdot((\frac{(1+\psi)^{1/\delta}}{(1+\psi)^{1/\delta}-1})^\delta-(\frac{1}{(1+\psi)^{1/\delta}-1})^\delta)\\ &=\omega(\pi^P_{opt}(a,b))+2c^\delta\cdot\frac{\psi}{((1+\psi)^{1/\delta}-1)^\delta}\\ &\leq \omega(\pi^P_{opt}(a,b))+2c^\delta\psi\cdot(\frac{2\delta}{\psi})^\delta\\ &\leq \omega(\pi^P_{opt}(a,b))(1+2c^\delta\psi\cdot(\frac{2\delta}{\psi})^\delta\cdot\frac{k^{\delta-1}}{d^\delta})\end{aligned}$$

So finally it just remains to choose c such that $2\cdot(\frac{2\delta c}{d})^\delta\cdot\frac{k^{\delta-1}}{\psi^{\delta-1}}\leq\psi$ which is certainly true for $c\leq\frac{\psi d}{k^{(\delta-1)/\delta}\cdot 4\delta}$ □

The following corollary of the above Lemma will be used later in the proof:

Corollary 3.2 *Given a set of points P and two pairs of points (a,b) and (a',b') with $d(a,b)=d$ and $d(a,a')\leq c$, $d(b,b')\leq c$ with $c\leq \frac{\psi d}{k^{(\delta-1)/\delta}\cdot 8\delta}$, then we have $\omega(\pi^P_{opt}(a',b'))\leq (1+2\psi)\omega(\pi^P_{opt}(a,b))$ as well as $\omega(\pi^P_{opt}(a,b))\leq(1+2\psi)\omega(\pi^P_{opt}(a',b')$.*

Proof: Clearly the first claim holds according to Lemma 3.7. For the second one observe that $d' = |a'b'| \geq d - 2 \cdot c$ and then apply the Lemma again. □

Applying this Corollary, it is now straightforward to see that the approximation ratio of the modified template path is $(1+2\psi)^2(1+\varepsilon)$.

Lemma 3.8 *Given a well separated pair decomposition of a point set $P \subset \mathbb{Z}^2$ with separation constant $s = \frac{k^{(\delta-1)/\delta} \cdot 8\delta}{\psi}$, the path $\pi(p,q)$ returned for a query pair (p,q) is a $(1+2\psi)^2(1+\varepsilon)$ approximate k-hop path from p to q.*

Proof: Let (A,B) be the unique cluster pair connected by a blue edge with $p \in A$, $q \in B$, c_A, c_B their respective cluster centers. By the choice of the separation parameter and Lemma 3.7, we know that $|pc_A|, |qc_B| \leq \frac{\psi |c_A c_B|}{k^{(\delta-1)/\delta} \cdot 8\delta}$ and therefore $\omega(\pi(p,q)) \leq (1+2\psi)(1+\varepsilon)\omega(\pi_{opt}(c_A,c_B)) \leq (1+2\psi)^2(1+\varepsilon)\omega(\pi(p,q))$. □

Adequate Choice for ψ and ε

For a given $\phi > 0$, what is the right choice for ψ and ε to obtain an arbitrary approximation quality of $(1+\phi)$? In order to answer on that, note that in the preprocessing phase for $O(s^2 n)$ many different pairs of points we have to compute approximated path in $O(\frac{\log n}{\varepsilon^2})$ time. Having in mind that separation constant $s = O(\frac{1}{\psi})$, we have that preprocessing running time depend on ε and ψ as $O(\frac{1}{\psi^2\varepsilon^2} \cdot n \log n)$. Hence, we would like to choose ψ and ε such that $\frac{1}{\psi^2\varepsilon^2}$ is minimized subject to the constraint $(1+2\psi)^2(1+\varepsilon) = (1+\phi)$. This is the classical problem of finding constrained extreme values of the function with two real variables which can be solved using *Lagrange multiplier*.

Consider the functions $f(x,y) := \frac{1}{x^2y^2}$ and $g(x,y) := (1+2x)^2(1+y) - (1+\phi)$. The Lagrangian function of $f(x,y)$ subject to the constraint $g(x,y) = 0$ is the function of three variables

$$L(x,y,\lambda) = f(x,y) - \lambda \cdot g(x,y) = \frac{1}{x^2y^2} - \lambda\Big((1+2x)^2(1+y) - (1+\phi)\Big), \quad \phi > 0$$

where λ is known as the Lagrange multiplier. Next thing is to find values of x, y and λ such that $\nabla L(x,y,\lambda) = 0$ which is equivalent to the equations

$$\begin{aligned} 0 &= \tfrac{\partial L(x,y,\lambda)}{\partial x} = -\tfrac{2}{x^3y^2} - 4\lambda(1+2x)(1+y) \\ 0 &= \tfrac{\partial L(x,y,\lambda)}{\partial y} = -\tfrac{2}{x^2y^3} - \lambda(1+2x)^2 \\ 0 &= \tfrac{\partial L(x,y,\lambda)}{\partial \lambda} = (1+2x)^2(1+y) - (1+\phi). \end{aligned}$$

The above system of nonlinear equations reduces to

$$\begin{aligned} 0 &= 4x - y + 2xy \\ 0 &= (1+2x)^2(1+y) - (1+\phi) \end{aligned} \tag{3.2}$$

which solution formulates x and y as functions of the parameter ϕ. Returning to our initial problem and replacing x and y with ψ and ε in system (3.2), respectively, we can get exact values for ψ and ε. As those expressions are fairly complicated, we would rather observe approximate values of ψ and ε. We would like to locally approximate values for ψ and ε on some small interval with linear functions $\psi' = C_1 \cdot \phi$ and $\varepsilon' = C_2 \cdot \phi$, respectively, such that

$$(1+2\psi')^2(1+\varepsilon') \leq (1+2\psi)^2(1+\varepsilon) = (1+\phi). \tag{3.3}$$

Lemma 3.9 *For $\phi \in (0,1]$, inequality (3.3) holds for $C_1 = \frac{1}{12}$ and $C_2 = \frac{2}{5}$.*

Proof: In the system (3.2), y can be easily expressed in terms of x. Concerning substitutions mentioned above we can easily obtain a new inequality following from the inequality (3.3)

$$\frac{(1+2C_1 \cdot \phi)^3}{1-2C_1 \cdot \phi} \leq (1+\phi). \tag{3.4}$$

Note that for $\phi = 0$ equality holds. The left hand side of the inequality is a monotonically increasing function while the right hand side is a linear function. Therefore inequality will hold if and only if it holds for the largest possible value of ϕ we admit. As $\phi \in (0,1]$, for $\phi = 1$ we get

$$\frac{(1+2C_1)^3}{1-2C_1} \leq 2$$

which is true for sufficiently small C_1, e.g. $C_1 \leq \frac{1}{12}$. To compute the value for C_2 we will use the inequality (3.3) again:

$$(1+2\frac{\phi}{12})^2(1+C_2 \cdot \phi) \leq (1+\phi).$$

Using the same reasoning as for inequality (3.4), setting $\phi = 1$ we easily get that inequality holds for $C_2 \leq \frac{2}{5}$. □

3.3.2 Retrieving Cluster Pairs for query points in O(1) time

In the previous paragraphs we have shown that using properties of the well-separated pair decomposition, it is possible to compute O(n) 'template paths' such that for any query pair (p,q) out of the $\Omega(n^2)$ possible query pairs, there exists a good template path which we can modify to obtain a good approximation to the lightest k-hop path from p to q. Still, we have not shown yet how to determine this good template path for a given query pair (p,q) in constant time. We note that the following description does not use any special property of our original problem setting, so it may apply to other problems, where the well-separated pair decomposition can be used to encode in O(n) space sufficient information to cover a query space of $\Omega(n^2)$ size.

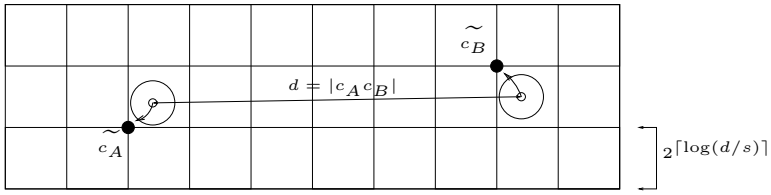

Figure 3.7: *Cluster centers c_A and c_B are snapped to closest grid points $\widetilde{c_A}$ and $\widetilde{c_B}$*

Gridding the Cluster Pairs

The idea of our approach is to round the centers c_A, c_B of a cluster pair (A,B) which is part of the WSPD to canonical grid points $\widetilde{c_A}, \widetilde{c_B}$ such that for any query pair (p,q) we can determine $\widetilde{c_A}, \widetilde{c_B}$ in constant time. Furthermore we will show that there is only a constant number of cluster pairs (A',B') which have their cluster centers rounded to the same grid positions $\widetilde{c_A}, \widetilde{c_B}$, so well-known hashing techniques can be used to store and retrieve the respective blue edge and also some additional information $I_{(A,B)}$ (in our case: the precomputed k-hop path) associated with that edge.

In the following we assume that we have already constructed a WSPD of the point set P with a separation constant $s > 4$. For any point $p \in \mathbb{Z}^2$, let $\text{snap}(p,w)$ denote the closest grid-point of the grid with cell-width w originated at $(0,0)$ and let $\mathcal{H} : \mathbb{Z}^2 \times \mathbb{Z}^2 \to (I \times E)^*$ denote a hash table data structure which maps pairs of integer points in the plane to a list of pairs consisting of some information type and a (blue) edge in the WSPD. Let u denote the size of the universe $\mathbb{Z}^2 \times \mathbb{Z}^2$ and note that $u = O(n)$. Using universal hashing [CW77] this data structure has constant expected access time. The idea of universal hashing is to choose (at run time) the hash function randomly from a given family of hash functions in a way that is independent of the keys which are going to be used. A collection of hash functions $\mathcal{M}$ is said to be *universal* if for every two different keys x and y we have

$$\Pr_{h \in \mathcal{M}}[h(x) = h(y)] = \frac{1}{m}$$

where m denote the size of the table $(I \times E)^*$. If we pick h at random from $\mathcal{M}$, then the average number of keys y_i such that $h(x) = h(y_i)$ is at most u/m. So, if we choose $m = \Theta(u)$, each operation has $O(1)$ average running time.

Preprocessing

- For every blue edge connecting clusters (A,B) in the split tree
  - $c_A \leftarrow \text{center}(R(A))$, $c_B \leftarrow \text{center}(R(B))$
  - $w \leftarrow |c_A c_B|/s$
  - $\widetilde{w} \leftarrow 2^{\lceil \log w \rceil}$
  - $\widetilde{c_A} \leftarrow \text{snap}(c_A, \widetilde{w})$

- $\widetilde{c_B} \leftarrow \text{snap}(c_B, \widetilde{w})$
- Append $((I_{(A,B)}, (A,B)))$ to $\mathcal{H}\,[(\widetilde{c_A}, \widetilde{c_B})]$

Look at Figure 3.7 for a sketch of the preprocessing routine for one cluster pair (A,B). Clearly this preprocessing step takes linear time in the size of the WSPD. So given a query pair (s,t), how to retrieve the information $I_{(A,B)}$ stored with the unique cluster pair (A,B) with $s \in A$ and $t \in B$?

Query(p,q)

- $w' \leftarrow |pq|/s$
- $\widetilde{w_1} \leftarrow 2^{\lceil \log w' \rceil - 1}$
- $\widetilde{w_2} \leftarrow 2^{\lceil \log w' \rceil}$
- $\widetilde{w_3} \leftarrow 2^{\lceil \log w' \rceil + 1}$
- for grid-widths w_i, $i = 1,2,3$ and adjacent grid-points $\widetilde{c_p}, \widetilde{c_q}$ of p and q respectively
  - Inspect all items $(I_{(A,B)}, (A,B))$ in $\mathcal{H}\,[(\widetilde{c_p}, \widetilde{c_q})]$
    - if $p \in A$ and $q \in B$ return $I_{(A,B)}$

In this description we call a grid-point $\widetilde{g}$ *adjacent* to a point p if $|\widetilde{g}p|_x, |\widetilde{g}p|_y < \frac{3}{2}\widetilde{w}$, where $|\cdot|_{x/y}$ denotes the horizontal/vertical distance. Clearly there are at most 9 *adjacent* points for any point p in a grid of width $\widetilde{w}$. In the remainder of this section we will show that this query procedure outputs the correct result (the unique $I_{(A,B)}$ with $p \in A$ and $q \in B$ such that (A,B) is blue edge in the WSPD) and requires only constant time. In the following we stick to the notation that $\widetilde{w} = 2^{\lceil \log |c_A c_B|/s \rceil}$, where c_A, c_B are the cluster centers of the cluster pair (A,B) we are looking for.

Lemma 3.10 *For $s > 4$, we have $\widetilde{w_i} = \widetilde{w}$ for some $i \in \{1,2,3\}$.*

Proof: As A and B are well-separated and $p \in A$ and $q \in B$, we know that $|c_A p|, |c_B q| < |c_A c_B|/s$. Therefore $|c_A c_B|(1-2/s) < |pq| < |c_A c_B|(1+2/s)$ and hence

$$\lceil \log \frac{|c_A c_B|(1-2/s)}{s} \rceil < \lceil \log \frac{|pq|}{s} \rceil < \lceil \log \frac{|c_A c_B|(1+2/s)}{s} \rceil$$

Using the fact that $s > 4$ we obtain

$$\lceil \log \frac{|c_A c_B|}{s} - 1 \rceil < \lceil \log \frac{|pq|}{s} \rceil < \lceil \log \frac{|c_A c_B|}{s} + 1 \rceil$$

and therefore

$$\lceil \log \frac{|c_A c_B|}{s} \rceil - 1 < \lceil \log \frac{|pq|}{s} \rceil < \lceil \log \frac{|c_A c_B|}{s} \rceil + 1$$

which proves the Lemma. □

This Lemma says that at some point the query procedure uses the correct grid-width as determined by c_A and c_B. Furthermore for any given grid-width and a pair of query points p and q, there are at most $9 \cdot 9 = 81$ pairs of adjacent grid points to inspect. We still need to argue that given the correct grid-width $\widetilde{w}$, the correct pair of grid points $(\widetilde{c_A}, \widetilde{c_B})$ is amongst these ≤ 81 possible pairs of grid points that are inspected.

Lemma 3.11 *For $\widetilde{w_i} = \widetilde{w}$, $\widetilde{c_A}$ and $\widetilde{c_B}$ are amongst the inspected grid-points.*

Proof: In the following we will restrict on the part to show that c_A is adjacent to p. Clearly we have $|c_A p| < \frac{|c_A c_B|}{s} \leq \widetilde{w}$, so in particular $|c_A p|_x, |c_A p|_y < \widetilde{w}$. Furthermore $|c_A \widetilde{c_A}|_x, |c_A \widetilde{c_A}|_y, \leq \widetilde{w}/2$ and hence $|\widetilde{c_A} p|_{x/y} < \frac{3}{2}\widetilde{w}$, i.e. c_A is *adjacent* to p. □

The last thing to show is that only a constant number of cluster pairs (A,B) can be rounded during the preprocessing phase to a specific pair of grid positions $(\widetilde{g_1}, \widetilde{g_2})$ and therefore we only have to scan a list of constant size that is associated with $(\widetilde{g_1}, \widetilde{g_2})$. Before we can prove this, we have to cite a Lemma from the original work of Callahan and Kosaraju on the WSPD [CK92].

Lemma 3.12 (CK92) *Let C be a d-cube and let $S = \{A_1, \ldots, A_l\}$ be a set of nodes in the split tree such that $A_i \cap A_j = \emptyset$ and $l_{\max}(p(A_i)) \geq l(C)/c$ and $R(A_i)$ overlaps C for all i. Then we have $l \leq (3c+2)^d$.*

Here $p(A)$ denotes the parent of a node A in the split tree, $l_{\max}(A)$ the longest side of the minimum enclosing box of $R(A)$.

Lemma 3.13 *Consider a WSPD of a point set P with separation constant $s > 4$, grid width $\widetilde{w}$ and a pair of grid points $(\widetilde{g_1}, \widetilde{g_2})$. The number of cluster pairs (A,B) such that c_A and c_B are rounded to $(\widetilde{g_1}, \widetilde{g_2})$ is $O(1)$.*

Proof: In the following we will establish a lower bound on $r(p(A))$ such that we can apply Lemma 3.12, since $r(p(A)) \leq l_{\max}(p(A))$. As $(p(A),B)$ is not a blue edge in the WSPD we have $r(p(A)) > \frac{|c_{p(A)} c_B|}{s}$. Furthermore $|c_{p(A)} c_B| \geq |c_A c_B| - r(p(A))$ and hence using the fact that $\frac{s\widetilde{w}}{2} < |c_A c_B| \leq s\widetilde{w}$ we obtain the following lower bound

$$r(p(A)) > \frac{s\widetilde{w}}{2+2s} = \frac{\widetilde{w}}{2+2/s}$$

So for $s > 4$ we obtain $l_{\max}(p(A)) > \frac{2}{5}\widetilde{w}$ Before we can apply Lemma 3.12, observe that if the clusters around $\widetilde{g_1}$ which are paired with some clusters around $\widetilde{g_2}$ are not disjoint (i.e. one is the parent of another), we can just refine those clusters until no cluster is the parent of another, this will only increase the number of links to 'the other side'. But applying Lemma 3.12 for the cube centered at $\widetilde{g_1}$ with side-length $\widetilde{w}$ and

$c = 5/2$ bounds even this larger number of clusters by $(3 \cdot 5/2 + 2)^2 < 91$. As the same bound holds for the clusters around $\widetilde{g_2}$, the number of cluster pairs that might be assigned to $(\widetilde{g_1}, \widetilde{g_2})$ is less than $91 \cdot 91 < 9000$ and therefore O(1). □

Putting everything together we get the main theorem of this section:

Theorem 3.4 *Given a well-separated pair decomposition of a point set P with separation constant $s > 4$. Then we can construct a data structure in space $O(n \cdot s^2)$ and construction time $O(n \cdot s^2)$ such that for any pair of points (p,q) in P we can determine the unique pair of clusters (A,B) that is part the well-separated pair decomposition with $p \in A$, $q \in B$ in constant time.*

Together with the results of the previous Section we obtain the following main result of our work:

Theorem 3.5 *Given a set of points $P \subset \mathbb{Z}^2$, a distance function $\omega : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}^+$ of the form $\omega(p,q) = |pq|^\delta$, where $\delta \geq 1$ and $k \geq 2$ are constants, we can construct a data structure of size $O(\frac{1}{\varepsilon^2} \cdot n)$ in preprocessing time $O(\frac{1}{\varepsilon^4} \cdot n \log n + \frac{1}{\varepsilon^6} \cdot n)$ such that for any query (p,q) from P, a $(1+\varepsilon)$-approximate lightest k-hop path from p to q can be obtained in constant $O(1)$ time which does not depend on δ, ε, k.*

3.4 Implementation and Experiments

All our algorithms introduced in Sections 3.1, 3.2 and 3.3 are computing either optimal or approximated solution to the k-hop path problem. One interesting thing to see is how do simple heuristic solutions without any theoretic guarantee behave and to compare those results to our algorithm solutions. For that purposes, we will use the following heuristic:

The Milestone Heuristic For very dense point sets, there is a very simple heuristic, which uses the observation that in the "ideal" case, the segment pq is divided into k subsegments of equal length. Clearly, if such a k-hop path can be obtained, it is the optimum path. So the *Milestone Heuristic* tries to approximate this "ideal" path by virtually placing the $k-1$ "ideal" points $v_1, \ldots v_{k-1}$ on pq segment. As these v_i are typically not in P, we perform for each of them a nearest neighbor query on the point set P and use the outcome as the replacement for v_i (each of these queries can be performed in $O(\log n)$ time). Nearest neighbor query data structures from computational geometry are by now standard in many software libraries and very space- and time-efficient implementations are available, e.g. in [MN95]. So the algorithm looks as follows:

1. determine "ideal" hop positions $v_1, \ldots, v_{k-1}$
2. for each v_i perform a nearest neighbor query on P to obtain $v'_i \in P$
3. output $pv'_1 \ldots v'_{k-1}q$ as the k-hop path

Unfortunately this approach can be fooled quite badly if the point set P is not equally distributed and there are large areas without any points in the area between p and q.

Heuristics for Optimal k-hop Path Computation

In Section 3.1 we introduced a naive $O(km)$ algorithm for computing an optimal k-hop path. In Lemma 3.3, we used a layered graph construction to describe how the method works. Note that we don't really have to make k copies of our starting instance but instead we could use a dynamic programming approach that fills a table of dimension $n \times k$ as follows:

- $\forall v \in P$: $\pi(p,v)^{(1)}_{\text{opt}} \leftarrow pv$
- for $i = 2$ to k do
  - $\forall v \in P$:
    - compute $\pi(p,v)^{(i)}_{\text{opt}}$ by looking at all possible w, the concatenations $\pi(p,w)^{(i-1)}_{\text{opt}}v$ and their weights $\omega(\pi(p,w)^{(i-1)}_{\text{opt}}) + \omega(w,v)$

Clearly this algorithm has running time $O(k \cdot n^2)$ as we have to fill in a table of size $k \cdot n$ and determining the value of one cell costs $O(n)$ since we look at all possible $w \in P$. It is not hard to figure out that this approach only works for extremely small problem instances and even for those, it is rather slow as we get a quadratic behavior in n per query. One obvious improvement to the above algorithm is due to the observation that if we are interested in the optimal k-hop path from p to q, points which are "far" away from the segment pq cannot be of any use for the solution (see Definition 3.1 of the frame and Lemma 3.2).

Neighborhood Pruning However, for the purposes of our experiments, we will use a slightly different definition of the neighborhood around segment pq. Namely, let D denote the distance between the query points, i.e. $D = |pq|$. If we restrict our dynamic programming approach to all points in P which have distance at most $\lambda \cdot D$ to the segment $|pq|$ – we call this the λ-*neighborhood of* pq –, what is the smallest value of λ such that we can still compute the optimal solution? See Figure 3.8 for an example of λ-neighborhoods. It is not hard to see that if the optimal path π_{opt} leaves the region which has distance at most $\lambda \cdot D$ to pq, the sum of the Euclidean lengths of the segments of this path must be at least $2 \cdot D \cdot \sqrt{\lambda^2 + 1/4}$. And as the "optimal" strategy to chop a path of any given length into k pieces such that the overall energy is minimized, is to chop it into pieces of equal length, we get the following inequality

$$\frac{(2 \cdot D \cdot \sqrt{\lambda^2 + 1/4})^\delta}{k^{\delta-1}} \leq D^\delta \tag{3.5}$$

which bounds λ in terms of the cost D^δ that are incurred when taking just one direct hop from p to q. So we get

$$\lambda_{\max} = \frac{\sqrt{k^{\frac{2\delta-2}{\delta}} - 1}}{2}$$

Therefore, if there are only few points in the neighborhood of the query points p and q (more precisely if there are only few points within distance $\lambda_{\max}|pq|$), we first use a standard range query data structure from computational geometry to report all those points and run the naive approach only for those and can expect a reasonably fast query time, which is now only quadratic in the number of points in the neighborhood of p and q.

Cascaded Neighborhood Pruning In the neighborhood pruning approach we have used the one-hop cost as an upper bound to limit the size of the neighborhood that still needs to be explored (see Eq. 3.5). Clearly, if we had a better upper bound (i.e. tentative solution) for the cost of getting from p to q within k hops, we could restrict the size of the neighborhood even further. How could such a better tentative solution be obtained? Well, we could start with a very small value for λ, even $\lambda = 0$ is viable, it just restricts the neighborhood to all points which lie on the segment pq. We run our dynamic programming approach on that set of points and use the outcome to bound the maximal value of λ that we have to consider to guarantee the optimal solution is found. So this cascaded strategy could be implemented as follows:

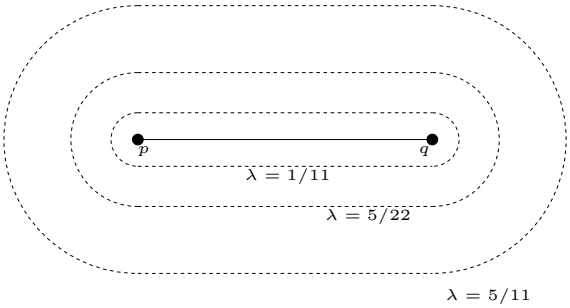

Figure 3.8: λ-*neighborhoods of a segment pq*

1. $\lambda \leftarrow 0.1$
2. UPPER $= |pq|^{\delta}$
3. while $\frac{(2 \cdot D \cdot \sqrt{\lambda^2+1/4})^{\delta}}{k^{\delta-1}} \leq$ UPPER
   - compute using dynamic programming the optimal k-hop path with respect to the λ-neighborhood of pq, update UPPER if necessary
   - $\lambda \leftarrow \lambda \cdot 2$

The procedure terminates as soon as it can prove that no larger neighborhood has to be inspected, which of course happens no later than after $O(\log \lambda_{\max})$ rounds. For dense point sets, this will turn out to be a lot more effective than the naive or simple neighborhood pruning strategy without cascading.

Experimental Settings

All the algorithms were implemented with the help of the LEDA library of data structures and algorithms ([MN95]). We used the floating-point geometry kernel which represents points in the plane by two `double` coordinates. As range query structure we employed the LEDA datatype `point_dictionary` which allows range queries in time $O(\log^2 n)$ and nearest neighbor queries in $O(n)$ worst-case time. But as these subroutines never dominated the running time in the respective algorithms where they were used, we did not put more effort into $O(\log n)$ worst-case query time implementations.

A very critical issue was the use of an appropriate hashing datastructure for accessing the precomputed template paths. We are using the LEDA type `h_array` which hashes 32-bit integer values to some information domain. But our implementation requires to hash 4-tuples of 32-bit integer values. So we had to reduce the number of bits by a factor of 4. In our experiments the best choice for a hash function was to choose the 3rd to 10th least significant bits of each of these 4 integers and concatenate them to obtain the hash value for the 4-tuple. For other hash functions we tried, the number of

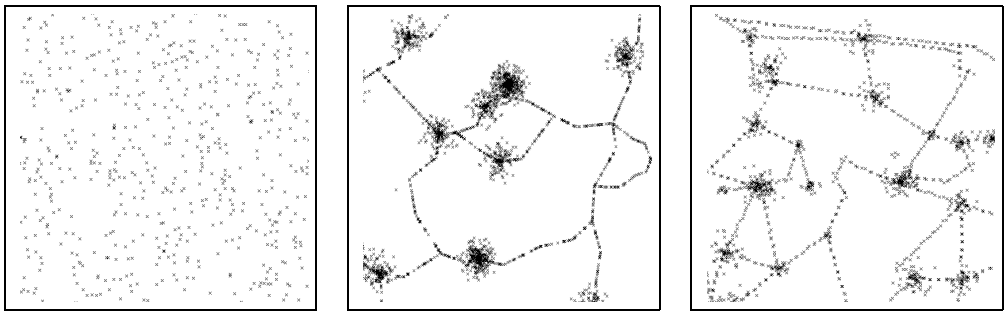

Figure 3.9: *Examples for test data: random (left), MST-based (middle) and Delaunay-based (right)*

collisions increased considerably and therefore accesses to the hashing table required going through a long list.

All algorithms were tested within an embedded simulation environment where data can be either read in or generated and then processed by our algorithms. Using a graphical user-interface, the different parameters and alternative algorithms can be selected and evaluated for running-time and quality of their produced solution.

Benchmarks

We conducted experiments on different test data and using different parameters for our algorithms. All running times were measured on a low-end 700 MHz Pentium III with 256 MB of RAM. We used g++ 2.95.4 with the -O option under a Linux 2.4.19 system.

Different test data sets were used to evaluate the quality of our algorithms. See Figure 3.9 for examples of the generated data.

As we had no real-world data available that could be put into a freely-available publication, we simulated the placement of radio stations along a road-network between cities to simulate a real world scenarios in Radio Networks. We had two simple algorithms to generate such data:

MST-based generation We first generated a random set of points (the cities) and computed a Euclidean minimum spanning tree. For all the leaves of the tree inside the Convex Hull of the random set we added a new edge. Furthermore, we generated a cluster of points around every city and also put randomly some points along every edge (roads). At the end, we pruned sharp angles.

Delaunay-based generation We first generated a random set of points (the cities) and computed the Delaunay triangulation. As we did not want to keep this ”triangular” road network, we removed some of the edges under the constraint that the remaining graph is still strongly connected. Then we assigned random weights to the cities and

Table 3.1: *1000 points randomly generated;* $k = 5$, $\delta = 2$, $S = 5$, $\varepsilon = 5$*; Query time and quality*

	WSPD	BF	BFp	Grid	Milestone
Av. Time	$8.0 \cdot 10^{-4}$	0.91	0.24	0.038	0.002
Max Time	$2.0 \cdot 10^{-3}$	1.45	1.24	0.080	0.01
Av. Rel. Err	15%	0	0	2.7 %	2.7 %
Max Rel. Err	49%	0	0	6.5 %	20 %
$\sigma_{\text{rel.err}}$	0.12	0	0	0.018	0.039

Table 3.2: *4000 points randomly generated;* $k = 5$, $\delta = 2$, $S = 5$, $\varepsilon = 5$*; Query time and quality*

	WSPD	BF	BFp	Grid	Milest.
Av. Time	$5.66 \cdot 10^{-4}$	14.59	4.75	0.07	0.01
Max Time	0.003	24.63	14.46	0.099	0.01
Av. Rel. Err	16 %	0 %	0 %	2.6 %	0.5 %
Max Rel. Err	32.6 %	0 %	0 %	4.8 %	2.5 %
$\sigma_{\text{rel.err}}$	0.088	0	0	0.016	0.007

generated radio stations accordingly. Finally we generated some random stations along the remaining edges.

Timings and Quality

In the following we are going to report timings and quality of the computed solutions for our different test data and varying problem sizes. For the precomputation of the template paths we chose a separation constant of $S = 5$ for the WSPD and $\varepsilon = 5$ for the grid pruning subroutine. Even though in theory, this guarantees only a solution within a factor of 216 (!) of the optimal solution, in practice, the returned solutions were rather close to the optimum. For the used parameters of $k = 5$, $\delta = 2$, $S = 5$, $\varepsilon = 5$, in fact the returned solutions were not more than 20 % off the optimum on the average. See tables 3.1, 3.2, 3.3, 3.4, 3.5 for the timing and quality results. Note that the 'WSPD' column in our tables stands for the "path template" approach from Section 3.3. Furthermore, 'Brute Force' and 'Brute Force pruned' (BF and BFp) columns stand for naive algorithm from Section 3.1 with neighborhood pruning and cascaded neighborhood pruning, respectively. 'Grid' column reports results for $O(\log n)$ approximation algorithm from Section 3.2 while 'Milestone' column reports results for Milestone heuristic.

From the results you can see that the query time using the WSPD approach remains basically constant, independent of the problem size, which is not true for all other

Table 3.3: *1000 points from the MST model;* $k = 5$, $\delta = 2$, $S = 5$, $\varepsilon = 5$*; Query time and quality*

	WSPD	BF	BFp	Grid	Milestone
Av. Time	$1 \cdot 10^{-4}$	1.193	0.937	0.009	0.0006
Max Time	$1 \cdot 10^{-3}$	1.63	4.03	0.01	0.01
Av. Rel. Err	14 %	0 %	0 %	3.6%	10.2 %
Max Rel. Err	38.7 %	0 %	0 %	14.4 %	35.9 %
$\sigma_{\text{rel.err}}$	0.123	0	0	0.047	0.114

Table 3.4: *4000 points from the MST model;* $k = 5$, $\delta = 2$, $S = 5$, $\varepsilon = 5$*; Query time and quality*

	WSPD	BF	BFp	Grid	Milestone
Av. Time	$1 \cdot 10^{-4}$	18.6	10.1	0.024	0.011
Max Time	0.001	27.19	21.09	0.039	0.02
Av. Rel. Err	10.1 %	0 %	0 %	3.3 %	14.3 %
Max Rel. Err	20.5 %	0 %	0 %	8.1 %	33.7 %
$\sigma_{\text{rel.err}}$	0.048	0	0	0.026	0.109

Table 3.5: *1000 points from the Delaunay model;* $k = 5$, $\delta = 2$, $S = 5$, $\varepsilon = 5$*; Query time and quality*

	WSPD	BF	BFp	Grid	Milestone
Av. Time	$4 \cdot 10^{-4}$	0.772	0.303	0.014	$5 \cdot 10^{-4}$
Max Time	0.002	1.13	1.28	0.03	0.01
Av. Rel. Err	17.2 %	0 %	0 %	5.7 %	10.7 %
Max Rel. Err	35 %	0 %	0 %	56 %	57 %
$\sigma_{\text{rel.err}}$	0.101	0	0	0.12	0.134

Table 3.6: *Time/Space for preprocessing on 1000 random points,* $k = 5$, $\sigma = 2$ *and varying* ε *and* S

	# templ.	time (s)	avg.err	max err
$\varepsilon = 10, S = 4$	7813	57.7	23%	58 %
$\varepsilon = 10, S = 5$	12042	98.0	22%	47 %
$\varepsilon = 10, S = 7$	21200	187.3	14%	36 %
$\varepsilon = 10, S = 11$	46148	458.8	12%	29 %
$\varepsilon = 5, S = 4$	8287	145.7	17%	37 %
$\varepsilon = 5, S = 5$	12004	230.6	15%	27 %
$\varepsilon = 5, S = 7$	21924	559.8	12%	34 %
$\varepsilon = 5, S = 11$	46236	1446.5	6 %	13 %
$\varepsilon = 2, S = 4$	7925	433.91	22 %	47 %
$\varepsilon = 2, S = 5$	11724	712.31	9 %	28 %
$\varepsilon = 2, S = 7$	22126	1606	9 %	31 %
$\varepsilon = 2, S = 11$	43347	3875	5 %	24 %

algorithms. In particular the brute-force variants suffer severely when increasing the problem size, but also the Milestone approach gets slower. The Grid approach also deteriorates a bit, but will saturate at some point (at least in theory). With regards to the quality, the brute force approaches are clearly the best since optimal, but also the Milestone Approach is not too bad. The results produced by the WSPD approach are mostly comparable to the Milestone and Grid approach but can be tuned by choosing different parameters as we will see later.

Of course, these very fast query times have their cost, both in terms of time for the precomputation as well as in terms of the space required to store the template paths. For this purpose we look again at the example of 1000 random points but now vary both ε (the parameter used for the grid approach when computing the template paths) as well as S (the separation constant for the WSPD). See table 3.6 for the results. Apart from the size of the precomputed structure and the preprocessing time we show the average and maximal relative error that was incurred by the precomputed paths for 30 random k-hop queries. Clearly, the more time and space one is willing to invest into computing good path templates, the better results one gets for the queries.

The choice of k – the number of allowed hops – also affects the running time of the grid pruning approach and therefore of the preprocessing step. See table 3.7.

As larger values for k require a finer grid, the running time of the precomputation grows rapidly. To keep the quality of the solution, we would have had to increase the value for S as well to accommodate for the finer granularity of the solution.

As mentioned in the introduction, even though setting $\delta = 2$ models the exact, free-space energy consumption, in practice people use larger values $\delta \in [2,4]$ to account for absorption effects etc. As δ also affects the running time of the grid pruning approach,

Table 3.7: *Time for preprocessing on 1000 random points,* $\sigma = 2$, $S = 5$, $\varepsilon = 5$, *varying k*

	WSPD		
	pre. time(s)	avg. err	max err
$k = 2$	14.6	6.1 %	13.8 %
$k = 4$	74.8	15.9 %	41.6 %
$k = 8$	530	25 %	41.1 %
$k = 16$	3471	29.2 %	55.8 %

Table 3.8: *Time for preprocessing on 1000 random points,* $k = 5$, $S = 5$, $\varepsilon = 5$, *varying* δ

	pre. time (s)	avg. err	max err
$\delta = 2$	232	14 %	30 %
$\delta = 3$	524	30 %	60 %
$\delta = 4$	817	41 %	75 %

we give experimental data for varying δ in Table 3.8.

It turns out that higher values for δ induce a considerably higher precomputation since the grid size chosen by the grid pruning algorithm is smaller. But still, the quality deteriorates, as with the larger exponent in the cost function, even small perturbations might increase the cost considerably. So to keep the same error bounds, a smaller value for ε and/or a larger value for S would have to be used.

3.5 Discussion and Conclusion

We have developed a data structure for constant approximate shortest path queries in a simple model for geometric graphs. We have also demonstrated through our experiments that near optimal k-hop paths can be queried very efficiently.

Although this model is motivated by communication in radio networks, it is sufficiently simple to be of independent theoretical interest and possibly for other applications. For example, Chan, Efrat, and Har-Peled [EHP98, CE01] use similar concepts to model the fuel consumption of airplanes routed between a set P of airports. For more details about real-world application scenarios of k-hop shortest path problem see Introduction chapter of the thesis.

However, looking through the eyes of real-world application, there are few weaknesses of our model. Let's discuss some of them in more details and show how possibly one could cope with such problems.

Fairness In Radio Networks scenario, minimal total energy consumption does not guarantee fairness, i.e., it might happen that one station is used so often that its batteries are quickly drained. This effect can be mitigated in several ways. For example, rather than storing fixed routes, we can simple store areas (e.g. squares) where relay stations should be located. Any combination of points in these relay areas will yield an energy efficient path. At least in densely populated areas one can then balance energy consumption by picking random stations in each relay area. One can even explicitly take energy reserves or other priorizations into account.

Lazy Precomputation In the path template approach as presented before, the idea was first to identify a collection of $O(n)$ source–target pairs (namely the centers of the clusters that are connected by a blue edge in the WSPD) and then precompute a good k-hop path for each of these pairs. In practice, it will turn out that identifying the blue edges can be done very quickly, and the really time-dominating step is the computation of the template paths (even when done using our $O(\log n)$ grid pruning approach). But our data structure can easily be modified into a "lazy precomputation" scheme. So at precomputation time, only the blue edges are determind. At query time for a pair (p,q), we first identify the corresponding blue edge. If a template path has been stored for that edge already, we use it (and have spent $O(1)$ time only to answer the query). Only if no template path has been stored already, we compute one using the grid approach (making this query expensive, i.e. $O(\log n)$). Observe that if a similar query, i.e. a query (p',q') with p' near p and q' near q, arrives later, it will find the precomputed template path and can therefore be answered in $O(1)$.

Fault-tolerance In many real-world applications reliability and quality of service (QoS) plays an important role. In particular, availability of the system has a very high priority. For our application this means that connections between two sites p and q

should not be prohibited or become very expensive if some stations inbetween collapse. Therefore, it is very reasonable to provide for backup paths between the sites, i.e. if one or more stations of an energy efficient path between p and q become unavailable, there are other equally efficient paths already precomputed at hand. But this is easy to incorporate into our approach. For each blue edge of the WSPD we precompute instead of *one* template path *several* template paths which are all node-disjoint. These node-disjoint paths can be obtained as follows: First use the grid approach to compute the first energy-efficient k-hop path. Then remove all the used representative nodes from the grid cells that have been used in this path. If there are still other nodes left in the respective grid cells, use them to get another path, which looks very similar to the first one, but is node-disjoint from the latter. If some of the used grid-cells are empty, just run the brute-force algorithm on the remaining grid cell representatives. In this way, one can easily compute several path templates for each blue edge all of which are node-disjoint.

Dynamization The good news, however, is that all the data-structures that we have used in our algorithms are also – at least in theory – available in a dynamic version, where updates can be performed in $O(\log n)$ time. Hence our whole construction could also be applied for moving and/or changing points. Whether these dynamic versions of the algorithms are also of practical value, has to be shown or proven wrong by an experimental study. For more information on dynamic versions of the required data structures, we refer to [AMN$^+$98], [AM00], [CK95b].

Acknowledgments

We would like to thank Sören Laue for a helpful conversations related to proof of Lemma 3.6 and Alexandra Zhilyakova for helping with the pictures in this chapter.

Chapter 4

(Approximate) Conic Nearest Neighbors and the induced Voronoi Diagram

Answering nearest neighbor queries for a given point set S in a (low-dimensional) Euclidean space is a classical problem in computational geometry and many algorithms have been proposed to solve that problem. This problem was mentioned in Knuth [Knu73] as "The post-office problem": Given a set S of n points in $\mathbb{R}^d$, store it in a data structure such that for any query point $q \in \mathbb{R}^d$, we can efficiently find point $s \in S$ that is closest to q, e.g. $d(q,s) = \min\{d(q,s') : s' \in S\}$ where $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ denotes the standard Euclidean metric. The natural datastructure to solve the post-office problem is the so-called *Voronoi diagram* $VD(S)$. $VD(S)$ is a decomposition of $\mathbb{R}^d$ into *Voronoi cells* such that for the cell $V(s,S)$ of point $s \in S$ we have $V(s,S) := \{p \in \mathbb{R}^d | d(p,s) \leq d(p,s') \forall s' \in S\}$. Unfortunately it is easy to come up with point sets for which the Voronoi diagram has size $\Omega(n^{\lceil d/2 \rceil})$, so its use to answer nearest neighbor queries in dimensions $d > 2$ is rather unattractive. Moreover, in the higher-dimensional case it seems impossible to give a data structure of size $O(n \text{ polylog } n)$ that answers queries in poly-logarithmic times. The lack of other methods that guarantee efficient query times has led to the introduction of the notion of *approximate nearest neighbors*. For a query point q and an arbitrary small constant $\varepsilon > 0$, a point $s \in S$ is a $(1+\varepsilon)$ approximate nearest neighbor for q if $d(q,s) \leq d(q,s') \cdot (1+\varepsilon) \forall s' \in S$. Not insisting on the *exact* nearest neighbor has allowed for datastructures of near-linear size and logarithmic query time, e.g. [AMN+98], [DGK99]. Similarly, the notion of an *approximate Voronoi diagram* (AVD) has allowed for space decompositions of near-linear size, e.g. by Har-Peled [Har01] and Arya and Malamatos [AM02]. Here each cell C of this decomposition has a point $N(C) \in S$ assigned such that $\forall q \in C, N(C)$ is an approximate nearest neighbor for q. In all these papers as well as in our work the dimension d is treated as a constant.

In this chapter we consider a variant of the nearest neighbor search problem that has

not been studied as extensively before. Namely, given a set of points S (and a cone C) we want to preprocess S such that for any query point $q \in \mathbb{R}^d$ we can determine the (approximate) nearest neighbor $s_q \in S$ that is contained in the cone C with apex at q. We will refer to this problem as *conic nearest neighbor* (cNN) problem. Note that the set S can be preprocessed into a data structure independent or dependent of the cone. Preprocessing S independently of the cone makes the problem more interesting and we are not aware of any previous results in this direction.

Throughout this chapter, we will restrict cone the C to the simplicial cone. A simplicial cone is the intersection of d halfspaces in $\mathbb{R}^d$. The hyperplanes that bound these halfspaces are assumed to be in general position, in the sense that their intersection is a point, called the *apex* of the cone (e.g. in the plane, a cone having its apex at the point q is a wedge bounded by two rays emanating from q that make an angle of at most π, see Figure 4.1).

Related Work

The original motivation for the cNN problem comes from several important applications. Probably the most famous application comes from the early 1980's, and the work on constructing the minimum spanning trees (MST) for a set S of n points in $\mathbb{R}^d$ of Yao [Yao82]. A Euclidean minimum spanning tree of the set S is a spanning tree of S and whose edges have a minimum total weight among all spanning trees. Yao [Yao82] who showed that the a minimum spanning tree (MST) connecting n points from S in a d-dimensional space can be computed in $o(n^2)$[1]. In his work, Yao showed that an $O(n)$-size supergraph of the minimum spanning tree can be found partitioning the space using a fan of constant number of cones around every point $p \in S$ and answering a conic nearest neighbors query for each cone. Using a simple geometric argument, he showed that if the angle of the cones is not 'too big', connecting every p to its cNN's will form a MST supergraph. Lets illustrate this idea with an informal description in $\mathbb{R}^2$. For every point $p \in S$, we divide the plane into eight regions (cones) $C_1, \ldots, C_8$ relative to p as shown in right-hand side of the Figure 4.1. The cones are formed by four lines passing through p and having angles of $0^o, 45^o, 90^o, 135^o$ with the x-axis. In each cone, compute the conic nearest neighbor of p, and add the corresponding edge to the set of edges E. After repeating the procedure for every point in S, we will compute the graph G with at most $8n$ edges for which Yao asserts that it contains an MST of S. To prove this, he uses a very simple and elementary observation: if q and r are any two points that belong to the first cone C_1 (like in the Figure 4.1), then if the angle of the cone is not more than 45^o, the following property holds

$$d(q,r) < \max\{d(p,q), d(p,r)\}. \tag{4.1}$$

It is not difficult to see that connecting point p with the rest of the points in $S \setminus \{p\}$ incurring the smallest cost can only happen via 8 possible conic nearest neighbors of p.

[1] An old open problem is whether the Euclidean MST of n points in $\mathbb{R}^d$ can be computed in time close to the lower bound of $\Omega(n \log n)$.

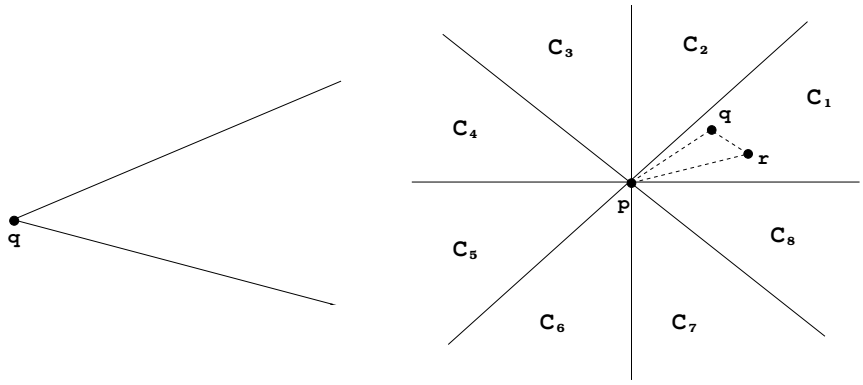

Figure 4.1: *(left) Example of cone in the plane with the apex in q. (right) Fan of cones partitioning the space around the point p.*

This construction is known today in the wireless literature as Yao-graphs or sometimes also as Θ-graphs (for the overview of the application of Yao-graphs in the wireless connectivity algorithms see the chapter "Applications of the Computational Geometry in Wireless Ad-Hoc Networks" of the book [CHD04]). Note that answering the cNN queries plays the crucial rule in the above approach. A natural question that arises in this regard is whether it is possible to compute the cNN of point p faster than just inspecting all the points in S? In [Yao82], Yao proposes a data structure such that cNN queries can be answered in $O(n^{1-a(d)+\psi})$, where $a(d) = 2^{-(d+1)}$ and for any fixed constant $\psi > 0$. We want to note, though, that his construction is dependent on the cone. However, this does not affect asymptotic running time and space since Yao is interested in only a constant number of different cones.

A similar idea of eliminating the unnecessary edges with the fan of cones and constructing the linear size Yao-graph has been widely used in geometric spanners (see [KG92], [RS91], [AMS94], [AMS99]). A *geometric spanner* for a set of points S is a graph in which each pair of vertices is connected by a "short" path. More precisely, a graph is called a t-spanner for S if the shortest path distance of any two points $p, q \in S$ is not more than t times the Euclidean distance between p and q. The intuition why answering cNN queries helps with construction of geometric spanner can be found in the following: Let q and r belong to cone C_1 (like in Figure 4.1) and assume q is a cNN of p with respect to the cone C_1. Suppose you want to travel from p to r. But instead of doing so, you move to the cNN point q first. Note, that the distance to r will be reduced by an amount proportional to the distance between p and q. If for the rest of the path we can maintain the same condition, the overall length of the path from p to r will be a small factor of the distance between p and q. In the presence of obstacles, one would have to be a little bit more careful, but essentially the same intuition works.

Yao graphs were also used for kinetically maintaining nearest neighbors of moving points (see [BGH97]) and motion planning (see [Che86], [Cla87]).

Czumaj et al. in [CEF+03] and [CEF+05] propose an algorithm that estimates with high probability the weight of a Euclidean MST to within $(1+\varepsilon)$ in sublinear time. Their algorithm assumes that the input is supported by approximate cone nearest

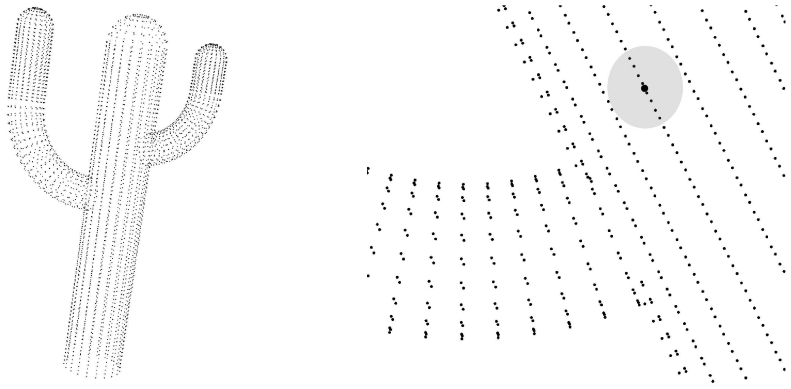

Figure 4.2: *(left) Point sample from the surface of a cactus. (right) Zoom-In on one of the branches of the cactus: Neighborhood of a sample point that does not allow for reliable normal estimation.*

neighbor queries data structure like the ones we present in Section 4.3.

Another important application for cNN queries stems from a problem in surface reconstruction/analysis of point cloud data. Here one is given a set S of sample points (point cloud) that are taken from an unknown surface Γ. The goal is now to reason about or in the best case even *reconstruct* the original surface Γ, e.g. find a faithful polyhedral approximation which has the sample points in S as vertices. See Figure 4.2 for a sample set taken from a cactus (model).

One important primitive when working with point cloud data is the estimation of surface normals for a given point on the surface (but only using the set S and not the original surface Γ). A common strategy for estimating the surface normal of a point $p \in \Gamma$ is to first collect the k-nearest neighbors of p (for some constant k) and then connect p to two of its k-nearest neighbors to form a 'nice' triangle using the triangle normal as an estimation for the surface normal. In fact one can prove that if the sample set S is a ε-sample (intuitively: it is sampled densely enough), then the triangle Δpqr formed by p and two of its k-nearest neighbors p,q yields a faithful approximation (up to an angle of $O(\varepsilon)$) to the surface normal at p *if* Δ is non-skinny, i.e. it is not formed by three almost collinear points, see [FR02]. Essentially, if p,q,r are almost collinear, a slight height change of the surface in one of the points changes the triangle normal by an arbitrarily large amount.

Now, the problem is that even if S is a ε-sample, there is no constant k for which we can guarantee that amongst the k-nearest neighbors of p are two points q,r which form a 'nice' triangle with p. That is due to the fact that the ε-sample condition is only a *lower* bound on the sampling density but allows for arbitrary oversampling in arbitrary patterns. For example, in Figure 4.2 right, as long as $k \leq 8$ collecting k-nearest neighbors from the query point will never yield sample points suitable for a faithful normal estimation in the query point. Observe, that this threshold $k \leq 8$ can be made arbitrarily high by just sampling further points along the already densely sampled lines. The sample set obviously remains a ε-sample, but normal estimation

via simple collection of k-nearest neighbors gets increasingly difficult[2]. The solution to this problem is to allow for nearest neighbor searches that are restricted to some direction, see [FR02]. Here, the authors divide the space into a *constant* number of *cones* with a common apex and then determine for a point p nearest neighbors within each cone. Using this approach it is easy to extract a 'nice' triangle with nearby points of p.

Attempts to instrument directly some of the known datastructures for approximate nearest neighbor queries ([AMN$^+$98], [DGK99]) to solve the case of cone queries failed so far, since a certain crucial packing property necessary for these approaches does not seem to hold in case of cones.

The broad applications of cNN were our main motivation to work on this problem.

Our contribution

In this chapter we consider the problem of computing (approximate) nearest neighbors in a point set S with the additional restriction that returned points have to lie in a cone with apex at the query point.

We start our study of cNN query problem by examining the structure of the exact linear size conic Voronoi diagram (cVD) induced by the notion of conic proximity in 2 and 3 dimensions (Section 4.1.1). We quickly review the sweep-line algorithm suggested by Clarkson [Cla87] for constructing the cVD in the plane and observe that the complexity of cVD becomes prohibitive in dimensions 3 and larger.

In Section 4.2 we develop a datastructure such that for an *arbitrary* simplicial cone *exact* cone queries can be answered in sublinear time. Armed with the *Partition theorem*[3], we are able to extend Yao's cNN result from [Yao82] constructing a datastructure independent of the cone. Intuitively, Partition theorem tells that we can always 'quickly' partition points of S into r simplices of approximately the same size such that every hyperplane cannot interset 'too many' simplices. The idea of approach is to preprocess simplices for nearest neighbor (NN) queries such that given any query point q and the cone C with apex in q:

(i) prune all the simplices outside of C;

(ii) for the simplices completely inside the cone, find the closest point using the NN queries;

(iii) recur on the simplices intersected by the cone's hyperplanes.

[2]If one assumes that S is a *tight* ε-, (ε,δ)-, or (locally) uniform sample, k can be determined easily. But if S is neither of those but only an ε-sample, making S tight itself seems to require reliable normal estimation.

[3]Note that Yao was not aware of Partition theorem, since it has been introduced in early 1990's (see [Mat92]) in context of simplex range-counting query in $\mathbb{R}^d$.

Since the number of simplices on which we have to recur is not too big by the Partition theorem, we show that cNN of q in the plane will be computed in $O(n^{1/2+\psi})$ and if $d > 2$ in $O(n^{(1-1/(d+1))+\psi})$, for any fixed $\psi > 0$.

Following the ideas of Arya, Mount and Smid in [AMS94] and [AMS99], for a fixed cone we propose near-linear size datastructures for answering cone queries approximately in $O((1/\varepsilon^d)\log^d n)$ time (see Section 4.3.1). Intuitively, given the query point $q = (q_1, \ldots, q_d)$ and the cone C with apex in q, the idea there is to tilt the axes of the coordinate system and obtain a new coordinates. In the new coordinates, the point $p^N = (p_1^N, \ldots, p_d^N)$ is inside the cone C with apex in $q^N = (q_1^N, \ldots, q_d^N)$ iff $p_i^N \geq q_i^N$, $1 \leq i \leq d$. Using the standard range queries data structure, we can report easily points inside the C. Furthermore, we also show how to extract approximately nearest point among them without increasing the space of the data structure.

As our main result, in Section 4.3 we show how to construct approximate conic Voronoi diagram of size $O((n/\varepsilon^d)\log(1/\varepsilon))$ which allows for query time $O(\log(n/\varepsilon))$. Following the ideas of [AM02], we partition the space into cells such that each cell $\mathcal{C}$ has a representative point associated $N(\mathcal{C})$. For any point $q \in \mathcal{C}$ and the cone C with apex in q, representative point $N(\mathcal{C})$ will have the following properties:

(i) $N(\mathcal{C})$ is slightly (by a factor of $(1+\varepsilon)$) further that the true cNN;

(ii) $N(\mathcal{C})$ lies either in the cone C or slightly outside of C (by an angle of $O(\varepsilon)$).

The construction is based on the well-separated pair decomposition (see Theorem 2.1). Namely, the idea is to construct the exponential grid of constant size around every WSPD pair. Since there are only linearly many WSPD pairs, the union of all exponential grids will give us a space decomposition of linear size. In the last and most difficult step we will associate appropriate $N(\mathcal{C})$ with each cell $\mathcal{C}$ satisfying the above two properties.

We conclude our work discussing some open problems in Section 4.4.

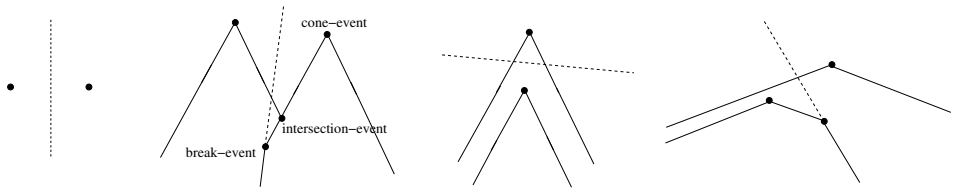

Figure 4.3: *Left most figure denote the classic VD cells whereas rest of the figures depict conic VD cells depending on the relative position of two sites and on the fixed angle* α*. Observe that the hashed lines denote the standard Voronoi separator between two sites.*

4.1 Preliminaries

We start the study of cNN problem by reviewing the notion of the exact conic Voronoi diagram introduced by Clarkson [Cla87]. We believe that understanding the structure of exact cVD will prove to be of use when dealing with cNN queries and in particular with understanding the approximate cVD in $\mathbb{R}^d$ introduced in Section 4.3. We also try to apprehend why cVD becomes rather unattractive in $\mathbb{R}^d$, for $d > 2$.

4.1.1 Exact Conic Voronoi diagrams

Let S be a set of n distinct points (sites) in the plane. In the following we fix a set of linearly independent vectors $V = \{v_1, v_2\}$ such that $C(q,V) := \{x \in \mathbb{R}^2 : x = q + (\lambda_1 v_1 + \lambda_2 v_2), \lambda_1, \lambda_2 \geq 0\}$ defines a simplicial cone with apex in q. The *reverse cone* of q is defined as $\overline{C}(q,V) := \{x \in \mathbb{R}^2 \mid x = q - (\lambda_1 v_1 + \lambda_2 v_2), \lambda_1, \lambda_2 \geq 0\}$. We assume that the cone angle $0 < \alpha(v_1, v_2) < \pi$ and since V is fixed, let $C(q) := C(q,V)$. We want to precompute the set of sites in S such that we can efficiently answer cNN queries for q with respect to $C(q)$. The precomputation step is based on the idea to compute a Voronoi-like diagram on the set of sites which we will call *Conic Voronoi diagram* (cVD). Answering the cNN for an arbitrary query point q will then reduce to the classical point location problem in the cVD.

For a fixed set S of sites, fixed angle and fixed direction we define a Conic Voronoi diagram of S as the subdivision of the plane into n cells, one for each site in S, with the property that a point q lies in the cell corresponding to a site $s_i \in S$ if and only if s_i is a conic nearest neighbor of q. That is, the conic Voronoi cell for $s_i \in S$ is defined to be:

$$\mathcal{V}_C(s_i) = \{p \in \mathbb{R}^2 \mid s_i \in C(p) \text{ and } d(p,s_i) \leq d(p,s_j) \text{ for all } s_j \in C(p) \cap S\}.$$

Note that the cVD is a bit more complicated than the normal Voronoi diagram. W.l.o.g. we assume from now on that the cones are facing downwards. In Figure 4.3 three different kinds of cVDs for two points are depicted (the cVD for one point $s = (s_x, s_y)$ is just the reversed cone $\overline{C}(s)$ mirrored at the line $y = s_x$). In the cVD for a set $S = \{s_1, \ldots, s_n\}$ of sites there are two different kind of edges: Edges that lie on the reversed

cones $\overline{C}(s_i)$ (*cone-edges*) and edges that lie on the standard Voronoi edges (*voronoi-edges*) between sites.

The cVD is planar and connected. A cell in the cVD belonging to site s_i has s_i as the unique point with highest y-coordinate. The latter allows us to compute the cVD with a sweep-line algorithm where we sweep the scene from top to bottom. We always maintain the invariant that above the sweep-line the cVD is correctly computed. For the current sweep-line we know in which order it is intersected by the edges (which may be cone- or voronoi-edges) and by the faces of the cVD. This is stored in an X-structure. The order changes only at three kind of event points: The sweep-line encounters a new site (*site-event*), a cone-edge becomes a voronoi-edge (*break-event*), or two neighboring edges along the sweep-line intersect (*intersection-event*). A Y-structure stores the event-points discovered so far. It is initialized with the site-events.

At each event-point the X- and Y-structure can be updated in $O(\log n)$ time. An event-point causes only local changes along the sweep-line. Also the Y-structure needs only constantly many updates. The concrete implementation is straightforward and left to the reader.

Lemma 4.1 ([Cla87]) *Let P be a set of n sites. The conic Voronoi diagram of P is of size $O(n)$ and can be constructed in $O(n \log n)$.*

Proof: If we imagine all the unbounded edges of cVD as going to a common point in infinity, we can prove the lemma with the help of Euler's formula, which states that for any connected planar graph with n_v nodes, n_e edges and n_f cells $n_v - n_e + n_f = 2$. Note that in our case $n_f = n + 1$ since every cell corresponds to a unique site. Furthermore, let $n_v = n'_v + n''_v + 1$ where n'_v denotes the number of site- and break-points and n''_v denotes the number of intersection-points of cVD. Since every site can produce at most two break-points, we have that $n'_v \leq 3n$. Also, observe that the rest of the points in the cVD have degree at least three (including node infinity) which implies that $3(n''_v + 1) \leq 2n_e$. Together with Euler formula this implies the statement of our lemma. □

Next we will show a lower bound for the Conic Voronoi Diagram in 3-dimensional space:

Lemma 4.2 *There exists a set of n points in 3-space which has a Conic Voronoi Diagram of size $\Omega(n^2)$.*

Proof: This proof and the pictures are based on 'true' cones, but translate directly to the case of simplicial cones. The main idea of construction is the same as for Voronoi diagrams. Suppose that the cones are facing downwards. We take two planes parallel to the (x,y)-plane. On the upper such plane we place $n/2$ points along a line l_1. We do the same on the lower plane but along a line l_2 which is perpendicular to l_1. For illustration have a look at Figure 4.4 a). The cVD of the lower $n/2$ points is depicted

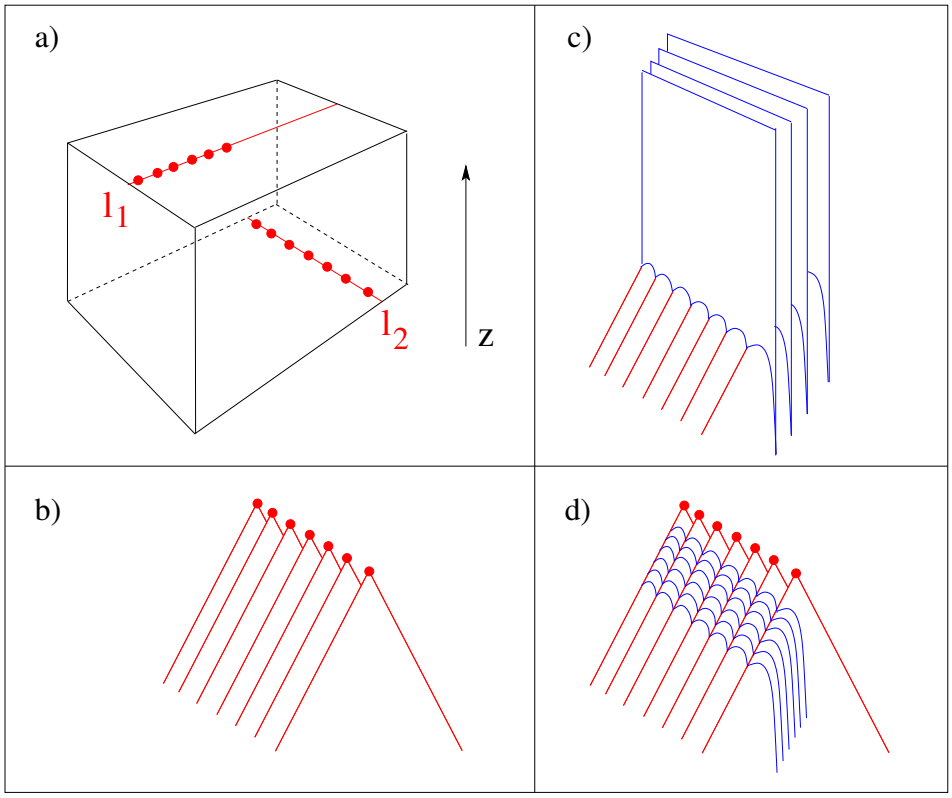

Figure 4.4: *The construction of a Conic Voronoi diagram of quadratic complexity in 3-space.*

in Figure 4.4 b). The separators of this cVD between neighboring cones are planar patches parallel to the z-axis bounded by parabolas. The cVD of all n points consists of two of these kind of cVD's just described: one for the lower and one for the upper $n/2$ points. The cells of the upper points are delimited from below by the cVD of the lower points. If we choose the upper points sufficiently high above the lower points the separators between neighboring cones are stopped by the lower cVD as shown in Figure 4.4 c). Each separator causes $n/2$ parabolic segments of the overall cVD, resulting in $\Omega(n^2)$ parabolic segments, Figure 4.4 d). □

Like for the regular Euclidean distance, the size of the respective cone Voronoi diagram in dimensions larger than 2 turns out to be prohibitive for its use in answering cone queries.

4.2 Exact cNN Queries for arbitrary Cones

Let S be a set of n distinct points (*sites*) in $\mathbb{R}^d$. Furthermore, let V be a set of d linearly independent vectors $v_1,\ldots,v_d \in \mathbb{R}^d$. We define the set $C(V) := \{v \in \mathbb{R}^d | v = \sum_i \lambda_i v_i,\ \lambda_i \geq 0\}$. For any point $q \in \mathbb{R}^d$ we define the *simplicial cone* of q as $C(q,V) := \{x \in \mathbb{R}^d \mid x = q+v,\ v \in C(V)\}$.

Suppose we want to construct a data structure such that for any cone $C(q,V)$ we can efficiently report the site $s_q \in S \cap C(q,V)$ such that $d(q,s_q) \leq d(q,s)$ for any other $s \in S \cap C(q,V)$. We say that s_q is a *conic nearest neighbor* (cNN) of q with respect to $C(q,V)$.

The solution to our problem mainly relies on the well known *Partition theorem* which has been used in the context of range searching ([Mat92]). We cite the theorem in the following:

Theorem 4.1 *Any set S of n points in $\mathbb{R}^d$ can be partitioned into $O(r)$ simplices, such that every simplex contains between n/r and $2n/r$ points and every hyperplane crosses at most $O(r^{1-1/d})$ simplices (crossing number). Moreover, for any $\psi > 0$ such a simplicial partition can be constructed in $O(n^{1+\psi})$ time.*

Using this theorem recursively one can construct a tree which is called a *partition tree* (e.g. the root of the tree, associated with S, has $O(r)$ children, each associated with a simplex from the first level, and so on). From now on we assume that r is a constant. Observe that if r is a constant, the partition tree is of $O(n)$ size and it can be constructed in time $O(n^{1+\psi})$ for any $\psi > 0$. With the help of such tree it is well-known that one can answer range counting queries in $O(n)$ space and $O(n^{(1-1/d)+\psi})$ time in $\mathbb{R}^d$, which is very close to the best possible.

The good news for us is that we can use almost the same data structure to answer cone queries. We first deal with the 2D case and then show that a similar approach works in higher dimension as well.

Lemma 4.3 *Let $S \subset \mathbb{R}^2$ be a set of points. For any $\psi > 0$, there is a data structure of $O(n \log n)$ size and $O(n^{1+\psi})$ construction time such that for any point q and an arbitrary cone $C(q,V)$ one can compute cNN of q in time $O(n^{1/2+\psi})$.*

Proof: Assume we are given a partition tree and suppose that we have some conic query to answer. Clearly by Theorem 4.1 we have the bound on the number of triangles that intersect the two semi-lines which is $O(\sqrt{r})$. On those triangles we recur, which leads to a total of $O(\sqrt{n})$ triangles intersected by the semi-lines. But still there might be $O(r)$ triangles lying completely inside the cone. To overcome this problem, one can precompute a normal Voronoi diagram (VD) for the points inside each triangle (the VD extends outside each triangle as well). This will increase the total space of the partition tree by a $O(\log n)$ factor since every level in the tree now will be of $O(n)$ size. However, by doing this we avoid recursing on the triangles that lie completely inside

the cone. Namely, for every triangle that lies inside the cone the point closest to q can be found by point-location in $O(\log n)$ time. □

In principle our ideas developed so far for computing cNN of a point q in the plane work for cNN in arbitrary dimension. The disadvantage is the high space complexity for storing the Voronoi diagrams associated with each triangle. The space complexity for a Voronoi diagram of n points is $\Omega(n^{\lceil d/2 \rceil})$. However, one can avoid storing the Voronoi diagrams at the cost of moving to the higher-dimensional space $\mathbb{R}^{d+1}$ instead of $\mathbb{R}^d$.

Lemma 4.4 *Let $S \subset \mathbb{R}^d$ be a set of points. For any $\psi > 0$, there is a data structure of $O(n)$ size and $O(n^{1+\psi})$ construction time such that for any point $q \in \mathbb{R}^d$ and an arbitrary cone $C(q,V)$ one can compute cNN of q in time $O(n^{(1-1/(d+1))+\psi})$.*

Proof: The main idea is to build a partition tree for S, additionally storing a representative point for every simplex. For the query cone $C(q,V)$ we recurse on all simplices that are intersected by the faces of the cone. What again remains to do is the handling of the inner simplices, i.e simplices which are completely inside the cone. Among these we determine the one with the representative point closest to q. Let s_i be this nearest representative point. We draw a sphere $B_{q,r}$ around q with radius $r = |q - s_i|$. It is easy to see that only the inner simplices intersecting B are candidates to contain the cNN of q. So we have to recurse also for these.

In order to be able to bound the number of inner simplices intersecting B we build the partition tree not in d- but in $(d+1)$-dimensional space. We make use of a well-known lifting step and project all points of S onto the unit-paraboloid P in $\mathbb{R}^{d+1}$. This gives us a new set $\tilde{S}$. We build the partition tree for $\tilde{S}$. This of course also leads to a partition of S into convex $\leq (d+2)$-gons. By construction and by the fact that a sphere in $\mathbb{R}^d$ is the projected intersection of a hyperplane and the unit paraboloid in $\mathbb{R}^{d+1}$ we know that every hyperplane and every sphere in $\mathbb{R}^d$ intersects $O(r^{1-1/(d+1)}) \leq (d+2)$-gons.

□

The techniques presented in this section only allow for sublinear but not polylogarithmic query times. Since in practice this is rather prohibitive, we will later present datastructures that guarantee polylogarithmic query times at the cost of only approximate answers.

4.3 Approximate cNN Queries

We have seen in previous two sections possible solutions for the exact cNN problem. However, in Section 4.2 we have constructed a linear size data structure which only allows cNN queries in sublinear time (note however, that this data structure does not depend on the cone). Moreover, our approach of constructing an exact conic voronoi diagram in Section 4.1.1 which would allow logarithmic cNN query time turns out to be impractical in higher dimensions.

In this section we will relax the problem in a sense that when querying with a point q we do not insist on receiving the *exact* nearest neighbor from the datastructure. More precisely, given a set S of points in $\mathbb{R}^d$ and a simplicial cone $C(q)$, preprocess S such that we can report a point that is slightly (by a factor of $(1+\varepsilon)$) further than the true conic nearest neighbor of q and – in the case of an approximate conic voronoi diagram – slightly outside (by an angle of $O(\varepsilon)$) of the cone $C(q)$. For such a relaxed version of the problem we will be able to construct the linear size data structure such that approximate conic nearest neighbor can be reported in logarithmic query time even in higher dimensions.

Let V be a set of d linear independent vectors $v_1,\ldots,v_d \in \mathbb{R}^d$. Let $s \in \mathbb{R}^d$ be some point and $b_i = v_i^T s$ for $i = 1,\ldots d$. We define the *cone* of s with respect to V as $\text{cone}(s) := \{p \in \mathbb{R}^d : \forall i : v_i^T p \leq b_i\}$. We also define the *reverse cone* of s with respect to V as $\overline{\text{cone}}(s) := \{p \in \mathbb{R}^d : \forall i : v_i^T p \geq b_i\}$. Note that we use here a different definition of cone – it is expressed with respect to the *normals* of the bounding hyperplanes. We will call the cone in which answers to a query q should lie the *reverse cone*.

In the following we fix a set of linear independent vectors V and aim to preprocess a set S of points in $\mathbb{R}^d$ such that for any given query point q we can find an *approximate conic nearest neighbor* (cANN) s_q with the following properties:

- if $d_{\min} = \min\{d(s,q) : s \in S \cap \overline{\text{cone}}(q)\}$, then $d(s_q,q) \leq (1+\varepsilon)d_{\min}$,
- either $s_q \in S \cap \overline{\text{cone}}(q)$ or the angle α between $\overline{s_q q}$ and $\overline{\text{cone}}(q)$ is $O(\varepsilon)$, where $\alpha := \min_{p \in \overline{\text{cone}}(q)} \angle s_q q p$.

Intuitively the former guarantees that the returned point is not too far away compared to the 'true' conic nearest neighbor, the latter guarantees that it 'almost' lies within the desired reverse cone of the query point.

The first approach we propose is based on a simple construction of nested range trees and returns points that lie exactly in the cone of query point q, and only approximates the distance. The second approach is based on a conic approximate Voronoi diagram (cAVD) – a decomposition of the underlying space – and allows for very fast query times and rather low space requirements.

4.3.1 Reduction to 'orthogonal' Range Queries in a skewed Coordinate System

Here we briefly sketch a method to solve approximate conic nearest neighbor queries using nested range trees. For convenience we assume that the respective cone which we want to query is reasonably small, i.e. $\forall v_i, v_j \in V : \angle v_i v_j \geq 3\pi/4$, which implies that the angular diameter[4] is less than $\pi/4$. If this is not the case we could always subdivide the desired cone into a constant number of smaller cones.

The idea is to derive new coordinates for all $s \in S$ based on the set V of normals of the hyperplanes forming the cone[5]. We set the new ith coordinate s_i^N of a point $s = (s_1, \ldots, s_d)$ to be $s_i^N := v_i^T s$. Now, if we have a query point q with respective new coordinates $(q_1^N, \ldots q_d^N)$ all points p within its reverse cone have new coordinates $p_i^N \geq q_i^N$. That is we can determine them using a nested range query for points of the form $[q_1^N, \infty], \ldots, [q_d^N, \infty]$. It remains to extract an (approximately) closest amongst them. The following lemma is due to Arya, Mount and Smid ([AMS94] and [AMS99]):

Lemma 4.5 ([AMS94], [AMS99]) *Let S be a set of n points in $\mathbb{R}^d$, let V be a set of linear independent vectors (normals) and let for every $q \in S$, l_q denote a fixed ray that emanates from q and is contained in $\overline{cone}(q)$. We can preprocess S in a data structure of size $O(n \log^{d-1} n)$ such that given any point $q \in S$, we can compute in $O(\log^d n)$ time a point p in $\overline{cone}(q) \cap S \setminus \{q\}$ for which the distance between q and the orthogonal projection of p onto l_q is minimal, or determine that such a point does not exist.*

Proof: We build a 1-dimensional range tree on the first coordinate. Each internal node stores the points in its respective subtree as a 1-dimensional range tree on the second coordinate and so on. The overall space requirement is $O(n \log^{d-1} n)$. The result (a set of points) for the query in the first tree is returned in $O(\log n)$ batches, we query each of these batches according to the 2nd coordinate and so on. Finally, in the last level (d) we obtain all points within the reverse cone of q in $O(\log^d n)$ batches. We order the sets associated with the internal nodes of the last level of our tree hierarchy according to the direction $\overrightarrow{l_q}$. Thus, point p with the desired property can be easily found by inspecting the *first* element in each batch. □

It's easy to see that for the cones of the angle less than $\pi/4$ and a point p in $\overline{\text{cone}}(q) \cap S \setminus \{q\}$ for which the distance between q and the orthogonal projection of p onto l_q is minimal, $d(q,p) \leq d(q,p') \cdot 3/2$ for all $p' \in \overline{\text{cone}}(q)$. Thus, point p is already a $3/2$ approximation of a true cNN of q.

Lemma 4.6 *Given a set S of points in $\mathbb{R}^d$ and a cone defined by a set of vector normals V, we can build a datastructure of size $O((1/\varepsilon^{d-1}) n \log^{d-1} n)$ such that cANN (with no angle error) can be determined in time $O((1/\varepsilon^{d-1}) \log^d n)$.*

[4] The angular diameter of a cone(q) is defined as $\max_{p,r \in \text{cone}(q)} \angle \overrightarrow{qp}\overrightarrow{qr}$.

[5] The idea of using a skewed coordinate system defined by the cone has been already widely used, see e.g. [Yao82], [AMS94], [AMS99].

Proof: Let W denote the set of fixed rays that emanates from q with the property that any arbitrary cone of angular diameter $O(\varepsilon)$ with apex in q and contained in $\overline{\text{cone}}(q)$ has to contain exactly one ray from W. Intuitively, set W defines ε-dense set of rays. Furthermore, let $\overrightarrow{W}$ denote the set of corresponding supporting direction vectors. It is not difficult to verify that the size of W is $O(1/\varepsilon^{d-1})$. Furthermore, with each of the $O(\log^d n)$ batches save $O(1/\varepsilon^{d-1})$ points such that each point is first with respect to the order defined by some direction vector from $\overrightarrow{W}$. Thus, inspecting at most $O(1/\varepsilon^{d-1})$ elements in each of the batches and outputting the one among these that minimizes the distance to q will return an cANN. □

Note that the Lemma 4.6 already provides a cANN solution but it uses a rather naive idea, though, which increases space requirements of our data structure by the factor $O(1/\varepsilon^{d-1})$. However, in the following Theorem we will show that cANN can be computed even without such an increase in space.

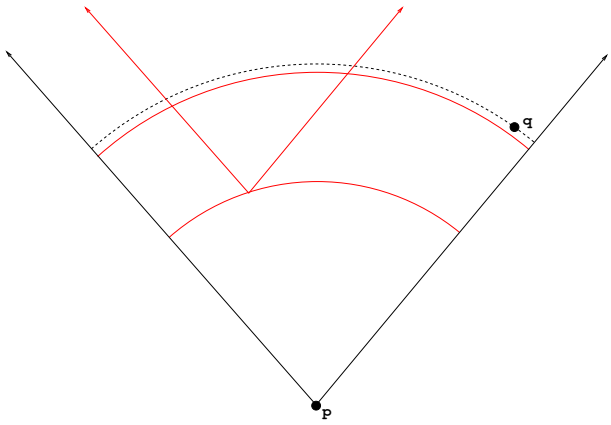

Figure 4.5: *Illustration of the proof of Theorem 4.2.*

Theorem 4.2 *Given a set S of points in $\mathbb{R}^d$ and a cone defined by a set of normal vectors V, we can preprocess them in a datastructure of size $O(n\log^{d-1} n)$ such that we can determine an cANN (with no angle error) in time $O((1/\varepsilon^d)\log^d n)$.*

Proof: Let l_q, as before, denote a fixed ray that emanates from q and is contained in $\overline{\text{cone}}(q)$ and let $\overrightarrow{l_q}$ be its supporting vector. We order the sets associated with the internal nodes of the last level of our tree hierarchy according to the direction $\overrightarrow{l_q}$ such that the point p amongst the $O(\log^d n)$ batches that minimizes $\overrightarrow{l_q}^T p$ can be determined easily. As discussed before, the point p is already $3/2$ approximation of a true cNN. That is we have an upper bound of $d_{\text{up}} = d(q,p)$ and a lower bound of $d_{\text{low}} = 2/3 \cdot d(q,p)$ for the distance of the conic nearest neighbor of q. Now consider the following part of the reverse cone of q: $G := \overline{\text{cone}}(q) \cap B(q, d_{\text{up}}/(1+\varepsilon)) - B(q, d_{\text{low}})$. Using a standard packing argument it is easy to see that we can certify using $O(1/\varepsilon^{d-1})$ many cone queries whether G does not contain any point (in which case the point determining

the current upper bound is a $(1+\varepsilon)$ cANN) or whether G contains a point (in which case we improve the upper bound by a factor of at least $(1+\varepsilon)$) (see Figure 4.5). Hence after at most $O(\log_{1+\varepsilon}(3/2)) = O(1/\varepsilon)$ iterations we arrive at a $(1+\varepsilon)$ cANN. Thus, the overall query time is $O((1/\varepsilon^d)\log^d n)$. □

We note that using the standard technique of *fractional cascading* one can improve the query time in Lemma 4.6 and Theorem 4.2 by an additional $\log n$ factor.

4.3.2 An approximate conic Voronoi diagram of near-linear size

In this part, we will discuss the construction of near-linear size data structure called approximate conic Voronoi diagram (cAVD) which would allow approximate cNN queries (with respect to distance and angle) in logarithmic time.

Constructing an cAVD can be interpreted as a meshing problem. Namely, for a set S of n points in $\mathbb{R}^d$, the main idea is captured in following: discretize the cAVD problem dividing some sufficiently large neighborhood around the point set S into cells such that all points in some particular cell have the same approximate conic nearest neighbor. Moreover, the neighborhood around S should be chosen such that for points outside the neighborhood any point from S is a good cANN, if such exists. We will borrow some ideas first presented in [AM02] for construction of ('normal') approximate Voronoi diagrams but following more the presentation in [HP]. In contrast to the constructions in [AM02] or [HP] we do not rely on a separate query datastructure to determine points associated with single cells, but rather determine them directly during the construction via a WSPD of the point set. This allow us having smaller smaller dependence on the dimension (ε^{-d} instead of ε^{-2d}) in the construction time.

Preliminaries

Assume that the point set S is contained in a cube of dimensions $[0.5-\varepsilon, 0.5+\varepsilon]^d$, and this cube is a minimum axis-aligned cube for S. For the rest of the chapter we only consider a decomposition or conic nearest neighbor relationships for points $q \in [0,1]^d$ since for points outside this unit cube, we can determine easily whether they're contained in an enclosing cone of the point set S, and then any point in S is a conic approximate nearest neighbor. Our algorithm constructs cells that arise in the quadtree (or its higher dimensional equivalent) when decomposing the unit cube $[0,1]^d$ recursively – we call this the *canonical grid* and the cells the *canonical cells*. The canonical grid G_{α_i} consists of cubes of width/side length α_i (for $\alpha_i = 2^{-i}, i \in \mathbb{N}$).

Hence the constructed cells in the algorithm are either disjoint, identical or one is contained in the other.

We now briefly review the notion of *box-decomposition* trees. Intuitively, a box-decomposition tree is an enhanced form of the well known quadtree structure and its higher dimensional generalizations. A (multidimensional) *quadtree* is a hierar-

chical decomposition of space into hypercubes in $\mathbb{R}^d$. Starting with the unit hypercube $U = [0,1]^d$, a *quadtree box* is any d-cube that can be obtained by a recursive splitting process that starts with U and generally splits an existing quadtree box by d axis-orthogonal hyperplanes passing through its center into 2^d identical subcubes. Such a decomposition naturally defines a 2^d-ary tree, such that each node is associated with a cube, called its *cell*. The *size* of a quadtree box is its side length. A nice property of quadtree boxes is that any two quadtree boxes either have disjoint interiors or one is contained inside the other. However, quadtrees suffer from two shortcomings, which make them inappropriate for worst-case analysis. First, if points are densely clustered in some very small region of space, it is not possible to bound the number of quadtree splits needed to decompose the cluster by a function of n alone. The box-decomposition (BD) tree [AMN+94] overcomes this by introducing an additional decomposition operation called shrinking. Second, if the point distribution is not uniform, the tree may not have logarithmic depth. The balanced box-decomposition (BBD) tree [AMN+98] extends the BD tree and remedies this problem by employing a balanced shrinking operation. More formally, a *box-decomposition* (*BD*) *tree* of a set S of n points is a 2^d-ary tree that compactly represents a hierarchical decomposition of space [AMN+94]. A BD-tree cell is either a quadtree box or the set theoretic difference of two quadtree boxes, an *outer box* and an *inner box*. Cells of the former type are called *box cells* and cells of the latter type are called *doughnut cells*. As with the quadtree, the root node is associated with U. When a cell contains at most one point of S it is declared a leaf. There are two ways that an internal node can be decomposed. A *splitting operation* decomposes space into 2^d subcubes in exactly the same way as the quadtree. A *shrinking operation* decomposes a quadtree box u into two children. The inner child cell is the smallest quadtree box u' that contains $S \cap u$, and outer child cell is the doughnut cell $u - u'$, which contains no points and hence is a leaf. The relevant properties of the BD tree are given below. Property (i) is proved in [AMN+94], and property (ii) is a simple generalization of (i).

(i) The BD tree has $O(n)$ nodes and can be constructed in time $O(n \log n)$.

(ii) A collection $\mathfrak{C}$ of n quadtree boxes can be stored in a BD tree with $O(n)$ nodes such that the subdivision induced by its leaves is a refinement of the subdivision induced by the quadtree boxes in $\mathfrak{C}$. This can be constructed in time $O(n \log n)$.

The *balanced box-decomposition* (*BBD*) *tree* introduced in [AMN+98] has many of the properties of a BD tree and it has $O(\log n)$ depth. As in the BD tree, a cell of a BBD tree is a quadtree box or the difference of two quadtree boxes. Let us state the properties of the BBD tree relevant to our problem:

(i) BBD tree T has $O(n)$ nodes and $O(\log n)$ depth and can be constructed in time $O(n \log n)$.

(ii) We can transform any BD tree T of size $O(n)$ into a BBD tree T' in time $O(n \log n)$ without the asymptotically increase in space.

(iii) For a given query point q, BBD tree T can be used to find smallest box containing q in $O(\log n)$ time.

A central component of our construction is the so-called *well-separated pair decomposition* (WSPD) of a point set. For a point set S in $\mathbb{R}^d$ and a *separation constant* s, the WSPD is a collection of $O(ns^d)$ cluster pairs (A_i, B_i), with $A_i, B_i \subset S$ and *centers* $a_i \in A_i, b_i \in B_i$ such that $\forall p \in A_i : d(p,a_i) \leq |ab|/s$ (and likewise for points in B_i). Furthermore, for any two points $s,t \in S$, there exists a unique pair (A_i, B_i) with $s \in A_i$ and $t \in B_i$. Intuitively a WSPD approximates the $\Omega(n^2)$ distance relationships using only $O(n)$ pairs of clusters (for constant s). For more details about the WSPD, see section 2.1.

We use the notion of an *exponential grid* $G_E(p,r,R,\varepsilon)$ around p introduced in [HP]. Let $b_i = b(p,r_i)$, $i = 0,\ldots\lceil \log R/r \rceil$ be the ball of radius $r_i = r2^i$. Define G'_i to be the set of cells of the canonical grid G_{α_i} that intersect b_i with $\alpha_i = 2^{\lfloor \log(\varepsilon r_i/(16d)) \rfloor}$. Obviously $|G'_i| = O(1/\varepsilon^d)$. We remove from G'_i all cells completely covered by cells of G'_{i-1}. Cells that are partially covered by cells in G'_{i-1} are replaced by the cells covering them in G'_{i-1}. Let G_i be the resulting set of canonical cells. And let $G_E(p,r,R,\varepsilon) = \cup_i G_i$. We have $|G_E(p,r,R,\varepsilon| = O(\varepsilon^{-d}\log(R/r))$. It can be computed in linear time in its size.

Data Structure and Query Processing

The overall picture of our construction is as follows: based on the well-separated pair decomposition of the point set S we generate a set $\mathfrak{C}$ of $O((n/\varepsilon^d)\log(1/\varepsilon))$ many grid cells. Some of the grid cells are marked *critical*, and some of the cells have a point from S associated, such that if the smallest of the generated cells containing a query point q is non-critical and has a point associated, this point is an approximate conic nearest neighbor for q (where the approximation is both with respect to the angle as well as the distance). Then in a second step, we treat the critical cells individually and partition them using a constant number of hyperplanes. The set of all generated (and possibly split) cells can then be transformed into a space decomposition and/or stored in a BBD tree of size $O(|\mathfrak{C}|) = O((n/\varepsilon^d)\log(1/\varepsilon))$ and it can be constructed in time $O(|\mathfrak{C}|\log|\mathfrak{C}|) = O((n/\varepsilon^d)\log(n/\varepsilon)\log(1/\varepsilon))$.

STAGE I OF THE CONSTRUCTION:

We first construct a WSPD for the point set with separation constant 32 and then consider all pairs of the WSPD. A pair (A,B) with representatives $(a,b) \in P \times P$ in the WSPD has the property that $\forall p \in A$ we have $d(p,a) \leq |ab|/32$ (and likewise for B and b). For each pair (A,B) with representatives a and b, construct the exponential grid $G_E(a, |ab|/8, 64|ab|/\varepsilon, \varepsilon) \cup G_E(b, |ab|/8, 64|ab|/\varepsilon, \varepsilon)$. The idea is now to associate either a or b with some of the cells but maintaining the invariant that if a cell c has a (b respectively) associated, then any point $q \in c$ can see a (b respectively) within its reverse cone or the line between q and a makes an angle of at most $O(\varepsilon)$ with the reverse cone of q. Furthermore some cells which might or might not have a or b associated with it, will be marked as 'critical'. Only cells that are marked as critical or have a point associated with them are remembered for further processing.

The construction proceeds as follows: Partition the set of grid cells into C_a (cells

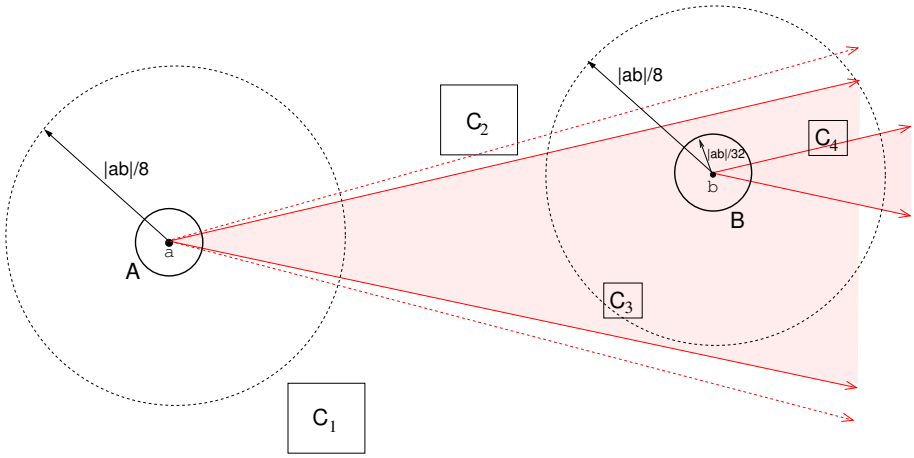

Figure 4.6: *Only cells C_2, C_3, C_4 are 'close' to the cone with apex in a. Note that the cell C_4 will be declared as 'critical'.*

that intersect the ball of radius $|ab|/8$ around a), C_b (cells that intersect the ball of radius $|ab|/8$ around b) and C_x (the remaining cells). Now determine which cells are 'close' to the cone for a (likewise to the cone for b) as follows: A cell $\mathcal{C}$ is said to be 'close' to the cone of a if $\mathcal{C} \cap \text{econe}(a) \neq \varnothing$ where enlarged cone is defined as $\text{econe}(a) := \{p | \exists p' \in \text{cone}(a)$ such that $\angle pap' \leq \varepsilon\}$. Now for all cells $\mathcal{C} \in C_x$:

- if $\mathcal{C}$ is only close to the cone of a, store a with $\mathcal{C}$,
- if $\mathcal{C}$ is only close to the cone of b, store b with $\mathcal{C}$,
- if $\mathcal{C}$ is not close to either ... don't store anything,
- if $\mathcal{C}$ is close to both, store either a or b (whichever is closer to the center of $\mathcal{C}$).

For all cells $\mathcal{C} \in C_b$ (that is, cells near b)

- if $\mathcal{C}$ is close to the cone of a, store a with it,
- furthermore, if $\mathcal{C}$ is close to the cone of b, mark $\mathcal{C}$ as 'critical' and remember the WSPD pair (A, B) with it.

Do the symmetric thing for all $\mathcal{C} \in C_a$ (see Figure 4.6).

We collect all the cells created (i.e. that have a point associated or have been marked critical) in the above step (some cells might be created several times) and aggregate the conic neighbor for some cell $\mathcal{C}$ as follows: cell $\mathcal{C}$ keeps the closest (to its center) point stored with $\mathcal{C}$ or one of its ancestors (i.e. cells that contain $\mathcal{C}$). This step completely ignores the fact whether cells are marked 'critical' or not. 'Criticality' is also *not* inherited, i.e. a cell $\mathcal{C}$ is called critical iff *all* of its instances created were critical (no

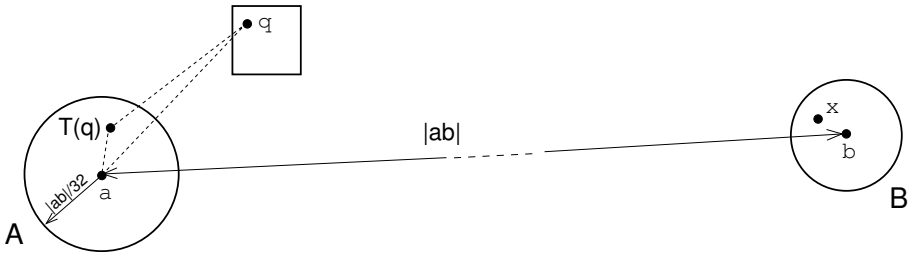

Figure 4.7: *Illustration of the proof of Lemma 4.8.*

dependence on ancestors or children). By doing so, the distance of the point associated with a cell can only decrease. This finishes the first stage of the construction.

The following lemma should now be easy to see.

Lemma 4.7 *Let C be a cell created during the processing of WSPD pair (A,B), $N(C)$ the point stored with it. Then for all points $q \in C$, $N(C)$ either lies in the reverse cone of q or is at most an angle of 2ε away from it.*

Proof: Assume w.l.o.g. $N(C) = a$. By construction, there exists a point $q' \in C$ and a point $p \in \text{cone}(a)$ (not necessarily from S) such that $\angle q'ap \leq \varepsilon$. Furthermore we also have $|qq'| \leq \varepsilon |aq'|$ (remember that C is far away from a otherwise we would not have associated a with it). But then it follows immediately that $\angle qap \leq \angle q'ap + \angle q'aq \leq \varepsilon + \arctan\varepsilon \leq 2\varepsilon$. So we get that the angle between the reverse cone of q and the ray $\overrightarrow{qa}$ is at most 2ε. □

Lemma 4.8 *Let $q \in [0,1]^d$ be a query point. If q has an (exact) conic nearest neighbor $T(q)$, then a cell containing q was created during our construction.*

Proof: Let x denote the point closest to $T(q)$ with $|T(q)x| \geq \varepsilon |T(q)q|$. Such a point x certainly exists (in the 'worst' case just take the furthest point from $T(q)$). Thus, the cell C containing q will be considered while constructing exponential grid around the WSPD pair (A,B) with representatives (a,b), which separates $T(q)$ from x (with $T(q) \in A$, $x \in B$, see Figure 4.7). However, we still need to show that the cell C had either a point associated or was marked critical (and hence created). Let us denote by $l = |T(q)q|$ the distance between q and its true cNN. If we have $|aT(q)| \leq \varepsilon l$, C was 'close' to the cone of a (as q has a point p' in the cone of a at distance of at most $|aT(q)|$ and then $\angle qap' \leq \arctan\varepsilon \leq \varepsilon$) and therefore was created. So assume $|aT(q)| > \varepsilon l$. But since on the other hand $|aT(q)| \leq |ab|/32$, $x = a$ would have been a better (i.e. closer) choice with $|T(q)x| \geq \varepsilon |T(q)q|$ which contradicts our initial choice of x. □

Let q be a query point, and C the *smallest* cell generated which contains q. We claim that if C is not marked 'critical', the point $N(C)$ stored with C is a $(1+O(\varepsilon))$ cANN with angle error of $O(\varepsilon)$.

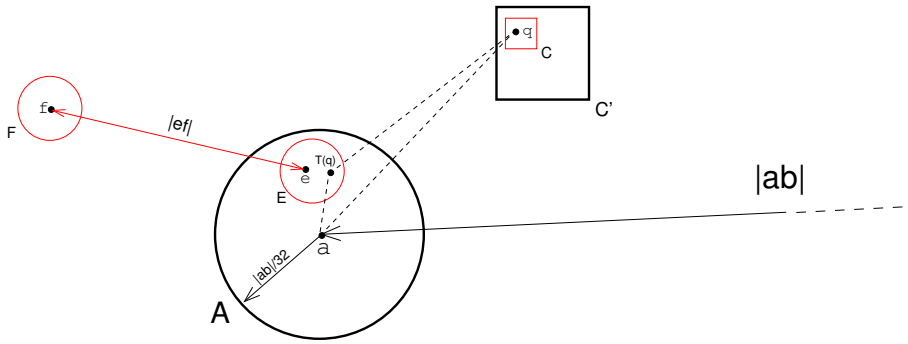

Figure 4.8: *Illustration of the proof of Lemma 4.9.*

Lemma 4.9 *Let $q \in [0,1]^d$ be a query point, $\mathcal{C}$ be the smallest cell created which contains q. If $\mathcal{C}$ is not critical, the point $N(\mathcal{C})$ stored with $\mathcal{C}$ has distance at most $(1+\varepsilon)d^{\mathcal{C}}_{\min}(q)$ where $d^{\mathcal{C}}_{\min}(q)$ denotes the distance of the exact conic nearest neighbor of q.*

Proof: We use the similar construction as in proof of Lemma 4.8; again we consider point x that is closest to $T(q)$ with $|T(q)x| \geq \varepsilon|T(q)q|$. We have argued that when considering the WSPD pair (A,B) separating $T(q)$ from x (with $T(q) \in A$, $x \in B$), we will create a cell $\mathcal{C}'$ containing q. We also showed that $|aT(q)| \leq \varepsilon l$ where $l = |T(q)q|$. Hence if the cell $\mathcal{C}'$ was not 'too close' to a, i.e. $d(\mathcal{C}',a)) > |ab|/8$, a was stored with $\mathcal{C}'$ and $|qa| \leq |T(q)q| + |aT(q)| \leq (1+\varepsilon)|T(q)q|$, that is a is a $(1+\varepsilon)$ cANN. For smaller cells contained in $\mathcal{C}'$ containing q, only better points could have been found, so the point associated with the smallest cell $\mathcal{C}$ containing q is certainly a cANN.

It remains to treat the case that $d(\mathcal{C}',a)) \leq |ab|/8$, that is, when considering (A,B), $\mathcal{C}'$ was marked critical (and a was not associated with it). Now assume the smallest non-critical cell $\mathcal{C} \neq \mathcal{C}'$ containing q was created by WSPD pair (E,F) with representatives e,f. Without loss of generality, assume $\mathcal{C}$ is closer to e than to f (see Figure 4.8). Then, if $|eT(q)| \leq \varepsilon|T(q)q|$ we're done since e is an cANN for q.

Thus, suppose that $|eT(q)| > \varepsilon|T(q)q|$. By the choice of x and the fact that $\mathcal{C}'$ was marked critical, we know that $\nexists s \in S : \varepsilon|qT(q)| < |sT(q)| < 15|ab|/16$ (otherwise $x = s$ would have been chosen). Let w and w' denote the width of cells $\mathcal{C}$ and $\mathcal{C}'$, respectively. We have $w' \leq \varepsilon|ab|/(8 \cdot 16 \cdot d)$ since $\mathcal{C}'$ was critical. Furthermore, $|eT(q)| > \varepsilon|T(q)q|$ but also by construction of the grid for (E,F) $|eq| \leq 32dw/\varepsilon \leq 32dw'/\varepsilon \leq |ab|/4$. Furthermore we have $|T(q)q| \leq |ab|/2$ otherwise $\mathcal{C}'$ would not have been critical and hence $|T(q)e| \leq |T(q)q| + |eq| \leq |ab|/2 + |ab|/4 \leq 3|ab|/4$ which contradicts the fact that $\nexists s \in S : \varepsilon|qT(q)| < |sT(q)| < |ab|15/16$. □

The previous two Lemmas imply that if the smallest generated cell $\mathcal{C}$ containing a query point q is not critical, the point stored with that cell is a valid cANN, i.e. it has distance at most $(1+\varepsilon)$ times the distance of the exact cNN, and the returned point lies at most an angle of 2ε off the reverse cone with apex at q.

STAGE II OF THE CONSTRUCTION:

In the following we will refine the construction to cope with the case that the query point ends up in a critical cell.

Let $\mathcal{C}$ be a smallest critical cell and suppose $\mathcal{C}$ was generated and declared critical while processing WSPD pair (A,B) with representatives (a,b) and $d(\mathcal{C},a) < |ab|/8$, where $d(\mathcal{C},a) = \min_{p\in\mathcal{C}} d(p,a)$. We simply split the relevant region of $\mathcal{C}$ by the planes of the econe(a) and assign a to one part, the 'old' representative (if existant) to the other part of $\mathcal{C}$. Note, however, that this procedure affects all possible smaller cells in $\mathcal{C}$, if any. Thus, the complexity of the intersection of relevant parts of $\mathcal{C}$ with the enlarged cone can be quite considerable (in particular, if $\mathcal{C}$ has many direct descendants). But since every cell has only one direct parent and the complexity of the intersection between a cube and d hyperplanes is constant for constant d, the overall complexity of the resulting decomposition remains linear in the number of original cubic cells. This finishes the second stage of the construction.

Lemma 4.10 *Let $\mathcal{C}$ be a critical cell that is also the smallest cell containing some query point q. If the true conic nearest neighbor of q is not in the set A but exists, then $\mathcal{C}$ has already an approximate conic nearest neighbor for q associated.*

Proof: Consider the WSPD pair (E,F) with representatives (e,f) separating $T(q)$ and a ($T(q) \in E$, $a \in F$), let $\mathcal{C}'$ be the respective cell containing q. If $d(\mathcal{C}',e) \geq |ef|/8$, the cell containing q got a point associated that is a cANN (proof along the lines of the above: if $|T(q)e| < \varepsilon|T(q)q|$ we don't have anything to show, otherwise consider the WSPD pair separating $T(q)$ and e ...). But if $d(\mathcal{C}',e) < |ef|/8$ the cell $\mathcal{C}'$ has to be smaller than the cell $\mathcal{C}$ (as it is closer to e than to a). Furthermore either $\mathcal{C}'$ or one of its children containing q has been marked or associated with a point (again the argument is that if $|T(q)e| < \varepsilon|T(q)q|$, $\mathcal{C}'$ would have been marked, otherwise consider the WSPD pair separating $T(q)$ and e ...). This contradicts the assumption that $\mathcal{C}$ was the smallest cell created containing q. □

So we know that for points q for which a critical cell $\mathcal{C}$ is the smallest containing cell, the conic nearest neighbor is in A (the cluster whose representative a marked $\mathcal{C}$ as critical) or it is already associated with $\mathcal{C}$.

Lemma 4.11 *Let $\mathcal{C}$ be a smallest critical cell and let δ be the minimum distance of a part of $\mathcal{C}$ to a, that is not covered by smaller cells. Then the diameter of point set A must be less than $\delta\varepsilon$.*

Proof: Assume $|A| > 1$ since otherwise claim holds trivially. If $\delta = 0$ (i.e. a is not covered by a smaller cell), another point in A together with a would have induced a finer grid cell at a. Hence assume $\delta > 0$ and suppose the diameter of point set A is greater than $\delta\varepsilon$. Consider the WSPD pair (E,F) separating a and the point $x \in A$ furthest from a. Since the distance $|ef|$ is much smaller than ab (e.g. $|ef| < |ab|/32$),

either the cell $\mathcal{C}$ is non-critical for (E,F) (contradiction to the initial assumption) or it is covered by smaller cells induced by (E,F). □

Lemma 4.12 *Any point q in a smallest critical cell $\mathcal{C}$ uncovered by smaller cells but contained in the enlarged cone of a has a as a cANN, distance wise and angle-wise.*

Proof: The returned point lies at most an angle of 2ε off the reverse cone with apex at q since the point q lies in the enlarged cone $\text{econe}(a)$. Furthermore, note that Lemma 4.10 and Lemma 4.11 imply that the point stored with that cell C has distance at most $(1+\varepsilon)$ times the distance of the exact cNN. □

The main result of this section can now be summarized in the following Theorem:

Theorem 4.3 *For a set of points $S \subset \mathbb{R}^d$ and a cone defined by d halfspaces, one can compute a decomposition of $\mathbb{R}^d$ into $O(\frac{n}{\varepsilon^d}\log\frac{1}{\varepsilon})$ regions with one associated point $\in S$ each, such that for any point $q \in \mathbb{R}^d$, the point associated with the cell $\mathcal{C}$ containing q is a cANN. If $\mathcal{C}$ has no point associated, q has no cNN in S. The space decomposition can be queried in time $O(\log(n/\varepsilon))$ and constructed in $O((n/\varepsilon^d)\log(n/\varepsilon)\log(1/\varepsilon))$ time.*

Proof: The size follows from the fact that the WSPD created has $O(n)$ pairs and each pair induces $O((1/\varepsilon^d)\log(1/\varepsilon))$ cells. These are then stored in a BD tree T. By BD-tree property (ii), the number of nodes in T is $O((n/\varepsilon^d)\log(1/\varepsilon))$ and can be constructed in time $O((n/\varepsilon^d)\log(n/\varepsilon)\log(1/\varepsilon))$. Finally we can transform the BD tree T into a BBD tree T'. This can be performed in time $O(m\log m)$, where m denotes the number of nodes in. This does not increase the space asymptotically, but has the desirable effect of reducing the time for tree descent to logarithmic in the total number of nodes, that is $O(\log(n/\varepsilon))$.

□

4.4 Discussion and Open Problems

In this work we tried to cope with a conic nearest neighbor problem, a variant of one of the most fundamental and basic problems in computational geometry, the nearest neighbor problem. The conic nearest neighbor problem has an applications in spanner constructions and wireless connectivity algorithms as already discussed in more detail at the beginning of this work.

However, our (approximate) datastructures that allow for polylogarithmic query time all require a *fixed* cone during their construction (e.g. the computed data structure depends on the cone). Even though in before mentioned applications, one is typically interested only in a constant number of cones – hence building our datastructures a constant number of times does not change the asymptotic space and running time – practicability would be greatly increased if we could design a data structure that could answer queries for *variable* cones (like the partition-tree based approach, but the latter has almost-linear query time).

Bibliography

[ABE98] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical models and image processing: GMIP*, 60(2):125–135, 1998.

[AM00] Sunil Arya and David M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17(3-4):135–152, 2000.

[AM02] S. Arya and T. Malamatos. Linear-size approximate voronoi diagrams. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 147–155, 2002.

[AMN+94] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Wu. An optimal algorithm for approximate nearest neighbor searching. In *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 573–582, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[AMS94] Sunil Arya, David M. Mount, and Michiel H. M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *IEEE Symposium on Foundations of Computer Science*, pages 703–712, 1994.

[AMS99] Sunil Arya, David M. Mount, and Michiel Smid. Dynamic algorithms for geometric spanners of small diameter: randomized solutions. *Comput. Geom. Theory Appl.*, 13(2):91–107, 1999.

[AMS00] Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximation algorithms for geometric shortest path problems. In *Proc. 32nd Annu. ACM Sympos. Theory Comput.*, pages 286–295, 2000.

[BCKK05] Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Geometric clustering to minimize the sum of cluster

sizes. In Gerth Stølting Brodal and Stefano Leonardi, editors, *ESA*, volume 3669 of *Lecture Notes in Computer Science*, pages 460–471. Springer, 2005.

[BGH97] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 747–756, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[BMKM05] Boaz Ben-Moshe, Matthew J. Katz, and Joseph S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 515–524, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[BSS02] Rene Beier, Peter Sanders, and Naveen Sivadasan. Energy optimal routing in radio networks using geometric data structures. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 366–376, London, UK, 2002. Springer-Verlag.

[CCRCU98] Felipe Contreras, Jurek Czyzowicz, Eduardo Rivera-Campo, and Jorge Urrutia. Optimal floodlight illumination of stages. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 409–410, New York, NY, USA, 1998. ACM Press.

[CE01] Timothy M. Chan and Alon Efrat. Fly cheaply: On the minimum fuel consumption problem. *J. Algorithms*, 41(2):330–337, 2001.

[CEF+03] Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Sublinear-time approximation of euclidean minimum spanning tree. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 813–822, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[CEF+05] Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM J. Comput.*, 35(1):91–109, 2005.

[CEHP04] Otfried Cheong, Alon Efrat, and Sariel Har-Peled. On finding a guard that sees most and a shop that sells most. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1098–1107, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[CHD04] Xiuzhen Cheng, Xiao Huang, and Ding-Zhu Du, editors. *Ad Hoc Wireless Networking*. Kluwer Academic Publishers, 2004.

[Che86] P Chew. There is a planar graph almost as good as the complete graph. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 169–177, New York, NY, USA, 1986. ACM Press.

[CK92] Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point-sets with applications to k-nearest-neighbors and n-body potential fields (preliminary version). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 546–556, New York, NY, USA, 1992. ACM Press.

[CK95a] Paul B. Callahan and S. Rao Kosaraju. Algorithms for dynamic closest pair and n-body potential fields. In *Proc. 6th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 263–272, 1995.

[CK95b] 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, 1995.

[Cla87] K. Clarkson. Approximation algorithms for shortest path motion planning. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 56–65, New York, NY, USA, 1987. ACM Press.

[CT98] Győrgy Csizmadia and Géza Tóth. Note on an art gallery problem. *Computational Geometry. Theory and Applications*, 10(1):47–55, 1998.

[CW77] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, New York, NY, USA, 1977. ACM Press.

[dBvKOS97] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry: algorithms and applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[DGK99] Christian A. Duncan, Michael T. Goodrich, and Stephen Kobourov. Balanced aspect ratio trees: combining the advantages of k-d trees and octrees. In *Proc. 10th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 300–309, 1999.

[Dij59] E. W. Dijkstra. A note on two problems in conexion with graphs. *In Numerische Mathematik*, 1:269–271, 1959.

[DO06] Erick D. Demaine and Joseph O'Rourke. Open problems (open problems from cccg 2005). In *Proceedings of the 18th Canadian Conference on Computational Geometry (CCCG'06)*, pages 75–80, 2006.

[EFKM05] Friedrich Eisenbrand, Stefan Funke, Andreas Karrenbauer, and Domagoj Matijevic. Energy-aware stage illumination. In *Proceedings of the 21st Annual Symposium on Computational Geometry : (SCG05)*,

pages 336–346, Pisa, Italy, 2005. Association of Computing Machinery (ACM), ACM.

[EHP98] Alon Efrat and Sariel Har-Peled. Fly cheaply: on the minimum fuel-consumption problem. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 143–145, New York, NY, USA, 1998. ACM Press.

[EHP02] Alon Efrat and Sariel Har-Peled. Locating guards in art galleries. In *Proceedings of 2nd IFIP International Conference on Theoretical Computer Science*, pages 181–192, 2002.

[Eid02] Stephan Eidenbenz. Approximation algorithms for terrain guarding. *Inf. Process. Lett.*, 82(2):99–105, 2002.

[FMMW06] Stefan Funke, Theocharis Malamatos, Domagoj Matijevic, and Nicola Wolpert. (approximate) conic nearest neighbors and the induced voronoi diagram. In *18th Canadian Conference on Computational Geometry*, pages 23–26, Kingston, Canada, 2006. School of Computing, Queen's University.

[FMS03] Stefan Funke, Domagoj Matijevic, and Peter Sanders. Approximating energy efficient paths in wireless multi-hop networks. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003: 11th Annual European Symposium*, volume 2832 of *Lecture Notes in Computer Science*, pages 230–241, Budapest, Hungary, September 2003. Springer.

[FMS04] Stefan Funke, Domagoj Matijevic, and Peter Sanders. Constant time queries for energy efficient paths in multi-hop wireless networks. In *First International Workshop on Algorithms for Wireless and Mobile Networks*, pages 97–111, Boston, USA, 2004. University of Bologna.

[FR02] Stefan Funke and Edgar A. Ramos. Smooth-surface reconstruction in near-linear time. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 781–790, 2002.

[FT77] L. Fejes-Tóth. Illumination of convex disks. In *Acta Math. Acad. Sci. Hungar. 29*, pages 355–360, 1977.

[GB01] H. González-Banos. A randomized art-gallery algorithm for sensor placement. In *SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry*, pages 232–240, New York, NY, USA, 2001. ACM Press.

[GLSv88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. 1988.

[Har01] S. Har-Peled. A replacement for voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

[HP] Sariel Har-Peled. Geometric approximation algorithms. Lecture Notes for CS598, UIUC.

[KG92] J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete Comput. Geom.*, 7(1):13–28, 1992.

[KKK83] J. Kahn, M. M. Klawe, and D. Kleitman. Traditional galleries require fewer watchmen. *SIAM J. Algebraic Discrete Methods*, 4:194–206, 1983.

[Knu73] Donald E. Knuth. *Fundamental Algorithms*, volume 3 of *The Art of Computer Programming*, chapter Sorting and Searching. Addison-Wesley, Reading, Massachusetts, 1973.

[LL90] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. pages 303–309, 1990.

[Mat92] Jirí Matousek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[MN90] Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990.

[MN95] Kurt Mehlhorn and Stefan Näher. Leda: a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102, 1995.

[Pap85] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inform. Process. Lett.*, 20:259–263, 1985.

[Pat00] D. Patel. Energy in ad-hoc networking for the picoradio. *Master's thesis*, 2000.

[Rap95] Theodore S. Rappaport, editor. *Wireless Communications: Principles and Practice*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[RS91] J. Ruppert and R. Seidel. Approximating the d-dimensional complete euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, pages 207–210, 1991.

[ST03] Bettina Speckmann and Csaba D. Tóth. Allocating vertex π-guards in simple polygons via pseudo-triangulations. In *Proc. 14th ann. ACM-SIAM symp. on Discr. Alg.*, pages 109–118. SIAM, 2003.

[Tot00] Csaba D. Toth. Art gallery problems with guards whose range of vision is 180^o. 17:121–134, 2000.

[TZ01] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *STOC*, pages 183–192, 2001.

[Urr00] Jorge Urrutia. Art gallery and illumination problems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027. North-Holland, 2000.

[vG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[Yao82] A. C. Yao. On constructing minimum spanning trees in k-dimensional space and related problems. *SIAM Journal on Computing*, 11:721–736, 1982.