Approximation Algorithms for Geometric Networks

Andersson, Mattias

2007

Approximation Algorithms for Geometric Networks

Mattias Andersson
Department of Computer Science
Lund University

Department of Computer Science
Lund University
Ole Römers väg 3
S–223 63 Lund
Sweden

E-mail: Mattias.Andersson@cs.lth.se

Printed in Sweden

ISBN 978-91-628-7250-2
ISSN 1650-1268
Dissertation 23, 2007

Abstract

The main contribution of this thesis is approximation algorithms for several computational geometry problems. The underlying structure for most of the problems studied is a geometric network. A geometric network is, in its abstract form, a set of vertices, pairwise connected with an edge, such that the weight of this connecting edge is the Euclidean distance between the pair of points connected. Such a network may be used to represent a multitude of real-life structures, such as, for example, a set of cities connected with roads.

Considering the case that a specific network is given, we study three separate problems. In the first problem we consider the case of interconnected 'islands' of well-connected networks, in which shortest paths are computed. In the second problem the input network is a triangulation. We efficiently simplify this triangulation using edge contractions. Finally, we consider individual movement trajectories representing, for example, wild animals where we compute leadership individuals.

Next, we consider the case that only a set of vertices is given, and the aim is to actually construct a network. We consider two such problems. In the first one we compute a partition of the vertices into several subsets where, considering the minimum spanning tree (MST) for each subset, we aim to minimize the largest MST. The other problem is to construct a t-spanner of low weight fast and simple. We do this by first extending the so-called gap theorem.

In addition to the above geometric network problems we also study a problem where we aim to place a set of different sized rectangles, such that the area of their corresponding bounding box is minimized, and such that a grid may be placed over the rectangles. The grid should not intersect any rectangle, and each cell of the grid should contain at most one rectangle.

All studied problems are such that they do not easily allow computation of optimal solutions in a feasible time. Instead we consider approximation algorithms, where near-optimal solutions are produced in polynomial time.

Contents

Preface

This thesis would never have been completed without the generous help of a number of people.

First and foremost I must thank my supervisors, Christos Levcopoulos and Joachim Gudmundsson, for all the help and guidance during these years. They both provided a friendly, productive atmosphere which made work most pleasurable, educational and inspirational. I consider myself truly fortunate receiving their supervision and I deeply thank them both!

Next, all people at the Computer Science Department in Lund must be thanked! It's been a pleasure getting to know everyone. In particular I must thank fellow Ph.D. students Eva-Marta, Mats-Petter, Mia, Magdalene, Andreas, Per and Martin for the great company. Also, special thanks go to Sonja, Eivor, Peter, Lars and Tomas for all the kind help in various practical matters!

For my co-authors (other than Christos and Joachim), Joachim Giesen, Giri Narasimhan, Mark Pauly, Bettina Speckmann, Patrick Laube and Thomas Wolle, a big thank you to all for the nice collaboration!

I would like to thank the following people and organizations for helping providing necessary travel funding: Andrzej Lingas, Nordic Network on Algorithms (NoNa), the Solander Program and Emo Welzl (Pre-doc Program, ETH, Zürich). Special thanks go to Joachim Gudmundsson for inviting me to Utrecht, the Netherlands, and for all the great help and collaboration during my research visit in Sydney!

My nearest family, thank you for all the love and support you constantly provide: Gunilla, Kent, Gullan, Ola, Inbar, Kailand, Anders, Helene, Adina, Linnea, Anna, Rickard, Jonathan, Martin, Keo, Lem, Alexander, Willie, William, Mats and Marianne! (And now, whenever you have a question, all you have to do is read this book!)

Finally, with all my love I thank my wonderful wife Emily. Her care and understanding, as always, supports me in everything!

Chapter 1

Introduction

Networks are fundamental structures, which we regularly encounter in our everyday lives. In its most abstract form, a network consists of elements that are somehow connected. A well-known example is, of course, the internet, with millions of computers connected all over the world. Other examples include neural networks, which aim to represent how neurons are connected in the brain, and social networks which represent how people are connected through, for example, family relations.

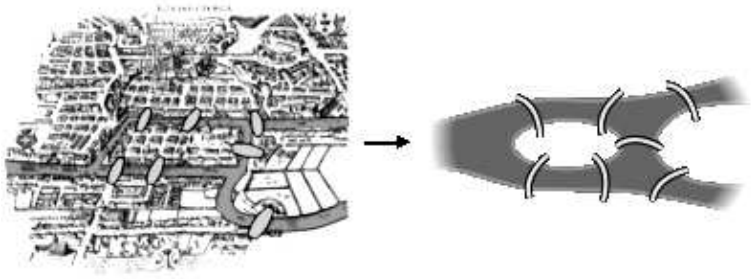

Figure 1.1: Illustration of the problem known as the *seven bridges of Königsberg*. (Picture published under the GNU Free Documentation License.)

Because of their fundamental nature, it is not surprising that scientific exploration of networks began early. One of the more famous examples is a mathematical problem known as the *seven bridges of Königsberg* or the *Euler path problem*, named after the famous mathematician Leonhard Euler (1707-1783) who finally solved the problem. At that time the river Pregel ran through Königsberg in such a way that two small islands in the center

of town were formed, see Fig. 1.1. The two islands were connected with each other and the town mainland through seven bridges, thus forming a land-bridge network. It was said that the town's people, at their Sunday walks, tried to find a path such that each bridge was crossed exactly once, and such that they ended up at the same spot where they started. No one had succeeded and it was not known whether or not it was at all possible.

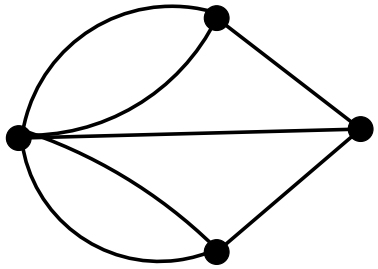

Figure 1.2: Euler represented the bridge-land network using only edges and points. Such a representation is known as a graph. (Picture published under the GNU Free Documentation License.)

Euler indeed managed to show that no such path existed. To do this he represented the land-bridge network using points and edges, as shown in Fig. 1.2, where the points were used to represent land, and the edges were used to represent bridges. Such a representation of a network is denoted a *graph*.

1.1 Geometric networks and graphs

In this thesis we consider a special kind of networks, called *geometric networks*, and their representation, so-called *geometric graphs*. In a geometric network the actual physical and geometric properties of the network, such as placement, distance and angle between elements, are of primary interest. As an example, a railroad network would be classified as geometric, while a social network would not.

In Fig. 1.3 the so-called *Pågatågen* railway network in the south of Sweden is represented using a geometric graph. The points represent towns, and an edge represents railway between two towns. Further, an edge has the same weight as the distance between the two towns that it connects. In a similar manner, geometric graphs can be used to represent a multitude

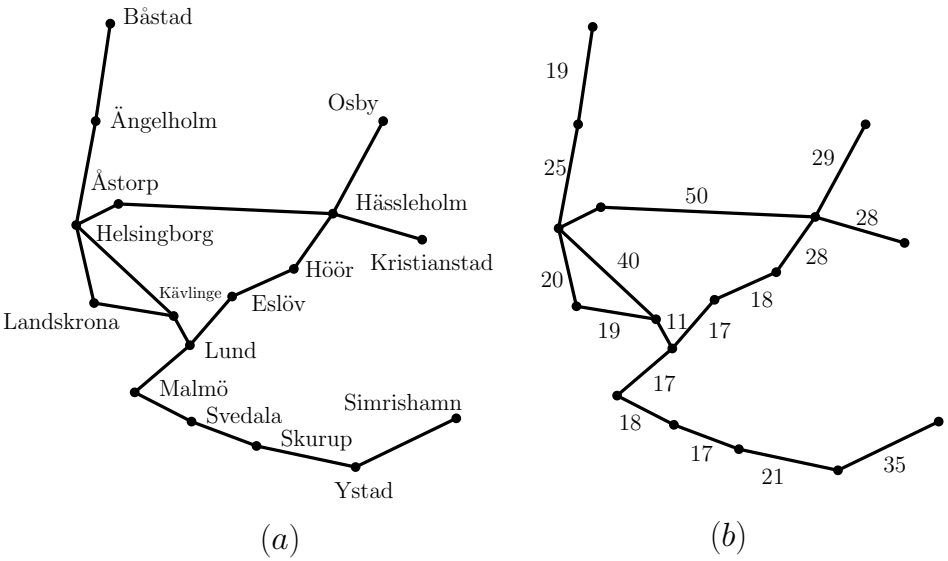

Figure 1.3: (a) A simplified representation of the so-called Pågatågen railway network in the south of Sweden. (b) Distances between cities, in kilometers.

of real-life geometric networks, such as molecular structures or electronic circuits.

1.2 Algorithms

For most people, daily use of computers include the use of, for example, email, web browsing and word processing programs. Such programs are nowadays so fast and simple that they, in effect, hide what really goes on inside the computer on which they run. What computers mainly do, as the name implies, is to *compute*. As an example, when a user sends an email, a transfer path through the computer network must be computed in order to allow efficient transmission.

During computation, millions or even billions of very simple operations, such as adding two numbers or changing the value of a memory cell, are performed. Thus, whenever something needs to be computed, we must create an *algorithm* (see Fig. 1.4 for an example), that is, a step-wise description of which basic operations to perform. One may compare an algorithm to a cooking recipe, which is, similarly, a step-by-step instruction on how to prepare a meal.

Algorithm GRIDPACK(S, ϵ')
1. $bestVal \leftarrow \infty$, $n \leftarrow |S|$, $\alpha = \beta = \sqrt{1+\epsilon'}$, $\gamma = \frac{1}{\sqrt{1+\epsilon'}-1}$
2. **for** each $1 \leq i, j \leq \log_\alpha n^c$ **do**
3. $S_{i,j} \leftarrow \emptyset$
4. **for** each $r \in S$ **do**
5. $i \leftarrow \lceil \log_\alpha \text{ width}(r) \rceil$
6. $j \leftarrow \lceil \log_\alpha \text{ height}(r) \rceil$
7. $S_{i,j} \leftarrow S_{i,j} \cup \{r\}$
8. **end**
9. **for** $k \leftarrow 1$ **to** $n^{2c \cdot f(\alpha,\beta,\gamma)}$ **do**
10. $G \leftarrow$ GENERATEGRID$(\alpha, \beta, \gamma, k, n, c)$
11. $val \leftarrow$ TESTGRID$(G, \{S\}_{1 \leq i,j \leq \log_\alpha n^c}, \alpha, \beta, \gamma)$
12. **if** $val < bestVal$ **then**
13. $bestVal \leftarrow val$ and $bestGrid \leftarrow G$
14. **end**
15. **Output** PACKINTOGRID$(S, bestGrid)$

Figure 1.4: An example algorithm. Each row contains a small number of basic operations.

Algorithms implemented on a computer are known as *computer programs*. What an algorithm, or computer program, basically does is that it takes an *input*, such as a map, and computes an *output*, such as a shortest path between two cities, using a finite number of basic operations.

In this thesis we suggest mainly algorithms for geometric network problems. See Section 5.3.3 for an example, where Steps 1 to 4 describe the algorithm.

1.3 Algorithm analysis

When suggesting an algorithm two things need to be shown. The first thing is that the suggested algorithms are correct, and the second that they are efficient.

Let us first consider the efficiency. Efficiency can be measured in many ways, such as how fast an algorithm is or how much computer memory it requires. One way to determine whether or not an algorithm is sufficiently efficient, is to actually create a computer program and test it. As an example, in Section 4.6 the results of such tests are described.

Another way is to perform a theoretical analysis, where we, regarding

speed, simply count the number of basic operations that will be performed. The fewer operations performed, the faster the algorithm.

Next, let us consider correctness, that is, we want the algorithms to do what they are intended to do. As an example, if we propose an algorithm and state that it computes the shortest path between two cities, we must mathematically prove this statement.

For an example, see Section 5.3.3, Theorem 5.8, where both efficiency and correctness are shown.

1.4 Approximation algorithms

Even though a modern computer can perform millions of basic operations per second, some problems are still so hard that they are not computable within reasonable time. As an example, there is a famous problem known as the *traveling salesman problem (TSP)*, where a salesman wants to visit all towns within his sales area exactly once. The problem is then to compute the shortest path for him to travel.

All known algorithms for this problem are such that if we increase the number of towns that the salesman wants to visit by one, then the number of performed basic operations may multipliy by a constant factor. Thus, for sufficiently large instances this problem, in the worst case, requires months instead of seconds in order to compute the shortest path.

Clearly this is not a reasonable approach, and instead of computing the absolute shortest path, we consider algorithms that compute a path that is *almost* a short as the shortest path. We say that such an algorithm is an *approximation algorithm*. Once we do this simple relaxation of the problem, algorithms are possible that will solve the traveling salesman problem within seconds or less.

In this thesis we do not study the traveling salesman problem directly, but we consider problems that are similarly hard to solve efficiently, and thus, require approximation algorithms. See Section 7.1 for an example.

1.5 Studied problems

Next we consider in greater detail the geometric graph problems studied, and also describe the approximation algorithms constructed for these problems. When constructing algorithms it is often helpful to know some of the basic properties of the geometric graph. As an example, a graph may have few or many edges, or be more or less well-connected. Here we mainly consider four

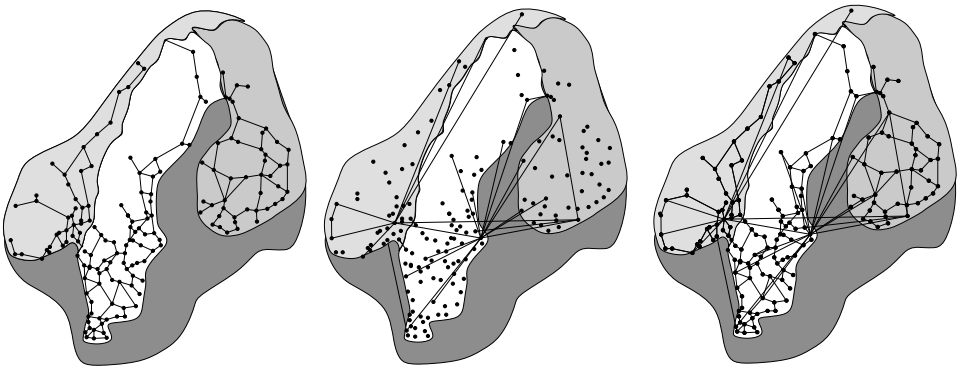

Figure 1.5: The three figures models parts of the transportation network in Norway, Sweden and Finland. (a) The domestic railway network in the three countries (not complete). (b) The railway connections between the countries together with the main air and sea connections within, and between, Norway, Sweden and Finland. (c) The two networks combined into one graph.

different kinds of geometric graphs, *t-spanners*, *triangulations*, *trajectories* and *minimum spanning trees*, each with its own specific basic properties.

Finally, we also study a geometric packing problem from the VLSI industry, where the underlying structure is not a geometric graph.

1.5.1 t-spanners

Fig. 1.5 illustrates part of the Nordic transportation network. One observation one can make is that there are better travel connections within a country than between countries. The network within one country is an example of a t-spanner, that is, a network that is rather well connected.

More specifically, when we build, for example, a railway network, it is usually not possible to have railway between every pair of cities. As an example, in Fig. 1.3 we see that if we want to go from Svedala to Ystad we have to travel via Skurup. Fortunately, the detour will not be that long compared to the distance, as the crow flies, between Svedala and Ystad. However, if we go from Båstad to Osby then we will have to take quite a long detour via Hässleholm, Åstorp, Helsingborg and Ängelholm. A t-spanner is a network such that no matter which two cities we travel between the detour will not be that long.

In Chapter 2 we consider the case exemplified by Fig. 1.5, where we construct an algorithm which pre-processes the network, and which then al-

lows fast computation of approximate shortest paths. A possible application for such an algorithm would be in automated travel planning, where short and cheap travel routes are desired. Further, in Chapter 6 we show how to construct a t-spanner fast and simple. The constructed t-spanner has the added property that the total weight of all its edges is low.

1.5.2 Triangulations

A triangulation is a geometric graph where all faces (except the outer face) are triangles, see Fig. 1.6. When storing a complex physical structure on a computer, it is often not possible to represent and store that structure exactly. As an example, the human form is simply too detailed for exact representation. Instead an approximate structure is needed, and for this purpose triangulations are commonly used. Triangulations are used in various fields, such as computer graphics and geographical information systems (GIS).

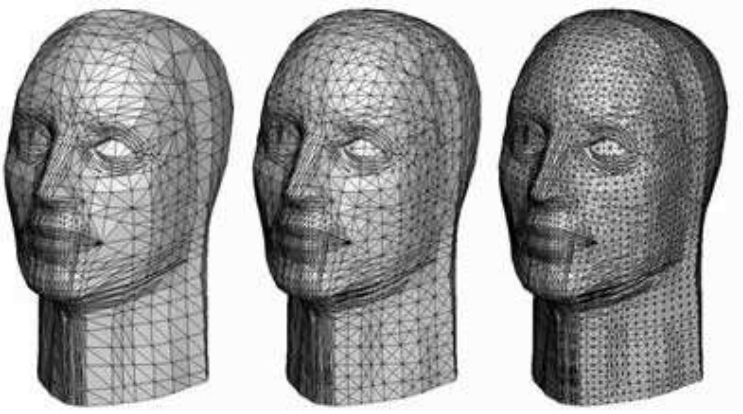

Figure 1.6: A human head represented using a triangulation, here in three different levels of detail.(Picture from the GNU Triangulated Surface Library [1].)

In Chapter 3 we give an algorithm that simplifies a triangulation in several steps, resulting in a simplification hierarchy. Such a hierarchy can, for example, be used in computer graphics, where objects are commonly viewed at various distances, thus requiring different levels of detail. Again, see Fig. 1.6, where a human head is represented using three different levels of detail.

1.5.3 Trajectories

Next we consider trajectories. In recent years location aware technology, such as GPS, has become increasingly accessible. This technology is used, for example, for real-time maps to be used in cars. The technology also enables collection of movement data, which is used, for example, by biologists to analyze animal movement. Such data is typically represented in a geometric graph as shown in Fig. 1.7, where each trajectory corresponds to a single individual. Each point in the graph corresponds to a location of an animal, and an edge represents movement between two locations.

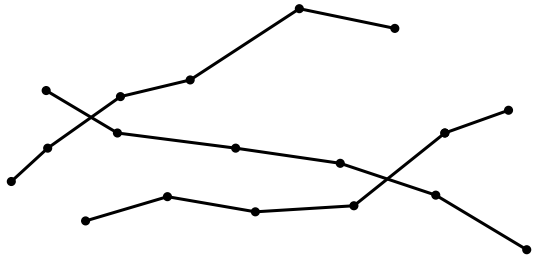

Figure 1.7: Trajectories representing the movement of three separate animals.

In Chapter 4 an algorithm is constructed which computes leaders in a group of moving animals. Computing such leaderships is of value, among other things, to biologists in order to analyze group behavior among animals.

1.5.4 Minimum Spanning Trees

In Chapter 5 we consider a problem from the ship building industry. The aim is to cut ship pieces from a sheet of metal using several robots moving over the sheet, see Fig. 1.8. Time is of importance, and thus we want to minimize the distance traveled by the robots. This problem is closely related to the traveling salesman problem described in Section 1.4, only we have several salesmen or robots. We give an approximation algorithm for this problem, which computes a cutting path for the robots such that the total cutting time is minimized.

More specifically, the algorithm uses a geometric graph called a minimum spanning tree (MST). Assume that you have, for example, a number of offices that are connected using as little phone cable as possible, the corresponding geometric graph is then a minimum spanning tree. The algorithm computes several MST's, one for each robot, so that the largest MST is minimized.

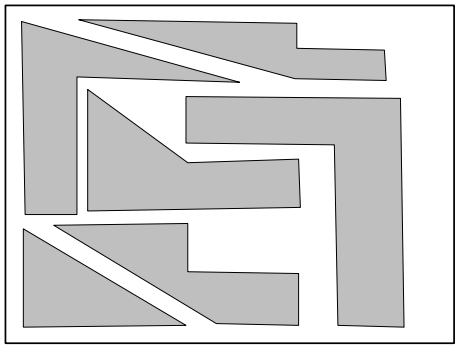

Figure 1.8: Ship pieces on a sheet of metal to be cut out by several robots.

Each such MST can then easily be transformed into an approximate traveling salesman path for each robot.

1.5.5 Chips on Wafers

One of the main physical components when building a computer is the so-called *computer chip*, which has a rectangular shape, see Fig. 1.9a. Chips are in turn constructed on so-called wafers, on which many chips of different sizes are often simultaneously constructed. One problem is that the wafer is expensive, and thus, it is desirable to pack the chips such that they use a minimum area.

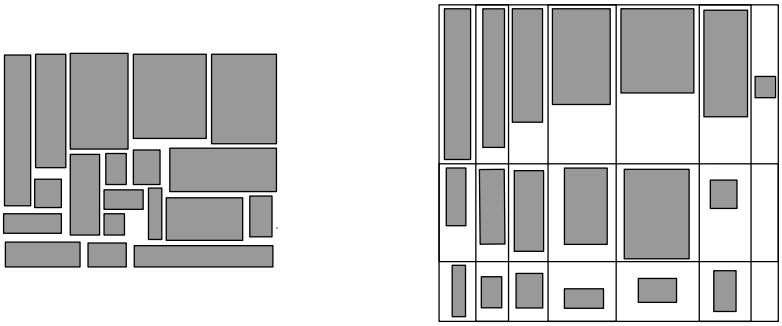

Figure 1.9: (a) A set of chips. (b) Chips grid packed on a wafer.

After construction the chips are cut out, and for technical reasons these

cuts must be made all across the wafer. Thus, the chips must be packed using a grid, such that no chip crosses the grid, and such that each cell of the grid contains at most one chip, see Fig. 1.9b.

In Chapter 7, we construct an algorithm that computes a grid packing that is arbitrarily close to the optimal packing.

1.6 Definitions

1.6.1 Preliminaries

A *vertex* $v = \{v_1, v_2, \ldots, v_d\}$ is an element in d-dimensional real space $\mathbb{R}^d$. An *edge* e between two vertices u and v, denoted (u, v), is the set

$$(u, v) = \{\alpha u + (1 - \alpha v) : \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1\}.$$

We denote the distance between two vertices u and v as $|uv|$, where

$$|uv| = \left(\sum_{i=1}^{d} |u_i - v_i|^t \right)^{1/t},$$

for some $1 \leq t \leq \infty$. That is, distances are measured according to some metric L_t, where the L_2 metric is assumed, unless otherwise stated.

A *polygonal chain* is specified by a sequence of vertices $p_1, p_2, \ldots p_n$, and edges (p_i, p_{i+1}), $1 \leq i < n$. A *simple* polygonal chain is such that only two consecutive edges (p_i, p_{i+1}) and (p_{i+1}, p_{i+2}) intersect, and only at their common vertex p_{i+1}, $1 \leq i \leq n-2$. A polygonal chain is called *closed* if $p_1 = p_n$. A *simple polygon* is an area in the plane bounded by a simple closed polygonal chain.

For a more thorough introduction to computational geometry, see for example the book by de Berg, van Kreveld, Overmars and Schwarzkopf [46].

1.6.2 Euclidean graphs

A Euclidean graph $G = (V, E)$ is a set of vertices V and a set of edges E, where the weight $|e|$ or $|(u, v)|$ of an edge $e = (u, v) \in E$ is such that $|e| = |(u, v)| = |uv|$. An edge is *directed* if we decide an order on its end vertices, and it is *undirected* otherwise. A directed edge $e = (u, v)$ is said to be directed from u to v. A Euclidean graph $G = (V, E)$ is a:

- t-spanner, if there exists a path between every pair of vertices $u, v \in V$ of total length at most $t \cdot |uv|$.

- Triangulation, for $d = 2$, if G is planar and consists of a simple polygon P, such that all points and edges of G are in the interior of P and such that no more interior edges can be added to E while maintaining planarity.
- Minimum Spanning Tree, abbreviated MST, if it consists of a tree that spans all the vertices of V, such that the total weight of E is minimized.
- Trajectory Graph, if the graph consists of a number of polygonal chains, such that each vertex in V is included in exactly one polygonal chain.

1.6.3 Complexity

Consider the real RAM [134] model of computation, an algorithm A, and a possible input i for A. Let $T_A(i)$ denote the number of primitive operations performed by A on i before terminating. Furthermore, let I denote the set of all possible inputs for A, and let $|i|$ denote the size of i. We perform worst-case analysis, and thus define a time complexity function

$$T_A(n) = \max_{i \in I, |i|=n} \{T_A(i)\}.$$

A space complexity function $S_A(n)$ is defined in a similar manner. Further, we disregard multiplicative factors in the space and time complexity, and use complexity classes as described by Knuth [103].

$O(g(n))$ denotes all functions $f(n)$, where there exist positive constants k and n_0 such that $f(n) \leq k \cdot g(n)$ for all $n \geq n_0$.

$\Omega(g(n))$ denotes all functions $f(n)$, where there exist positive constants k and n_0 such that $f(n) \geq k \cdot g(n)$ for all $n \geq n_0$.

$\Theta(g(n))$ denotes all functions $f(n)$, where there exist positive constants k_1, k_2 and n_0 such that $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$ for all $n \geq n_0$.

An algorithm A is called a *polynomial time* algorithm if $T_A(n) = O(n^j)$, for some constant j.

1.6.4 Approximation algorithms

An *optimization problem* consists of a set of instances I. Each instance $i \in I$ is associated with a pair (F, c), where F is the set of feasible solutions, and where c is a mapping $c : F \to \mathcal{R}$. That is c defines a *cost* $c(f)$ for each solution $f \in F$. An optimization problem can be either a *maximization* or

a *minimization* problem, and below we consider the minimization version. The maximization version can be defined in a similar manner. The problem, then, is to find a solution $f \in F$, such that

$$c(f) \leq c(y), \forall y \in F.$$

Let OPT denote a solution f for which this holds. We say that an algorithm A is a k-approximation algorithm, if A, for any instance $i \in I$ given as input, finds a solution $f' \in F$ such that

$$c(f') \leq k \cdot OPT.$$

In this thesis we consider approximation algorithms that find such a solution in polynomial time. Further, a $(1+\varepsilon)$-algorithm which takes an instance $i \in I$ and an arbitrarily small real constant $\varepsilon > 0$ as input is called a polynomial time approximation scheme, or a $PTAS$ for short.

Not all optimization problems have known polynomial time algorithms, where NP-complete problems is a well-known example. Thus, instead approximation algorithms are considered. For further description and definition of concepts such as optimization and NP-complete problems, see for example the book on combinatorial optimization by Papadimitriou and Steiglitz [131].

1.7 Thesis outline

This thesis consists of three separate parts. In the first part we consider problems where a specific network is given as input, in the second we consider problems where a network is computed, and in the third part other geometric optimization problems are considered.

Part I: Processing Networks

Chapter 2. We consider "islands" of t-spanners, where we construct a data structure which can be used to answer approximate shortest path queries. More formally, let $\mathcal{H}_1 = (\mathcal{V}, \mathcal{E}_1)$ be a collection of N pairwise vertex disjoint $\mathcal{O}(1)$-spanners where the weight of an edge is equal to the Euclidean distance between its endpoints. Let $\mathcal{H}_2 = (\mathcal{V}, \mathcal{E}_2)$ be the graph on $\mathcal{V}$ with M edges of non-negative weight. The union of the two graphs is denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1 \cup \mathcal{E}_2)$. We present a data structure of size $\mathcal{O}(M^2 + n \log n)$ that answers $(1+\varepsilon)$-approximate shortest path queries in $\mathcal{G}$ in constant time, where $\varepsilon > 0$ is constant.

Chapter 3. We consider the problem of simplifying a planar triangle mesh using edge contractions, under the restriction that the resulting vertices must be a subset of the input set. That is, contraction of an edge must be made onto one of its adjacent vertices, which results in removing the other adjacent vertex. We show that if the perimeter of the mesh consists of at most five vertices, then we can always find a vertex not on the perimeter which can be removed in this way. In order to maintain a higher number of removable vertices under the above restriction, we study edge flips which can be performed in a visually smooth way. A removal of a vertex which is preceded by one such smooth operation is called a 2-step removal. Moreover, we introduce the possibility that the user defines "important" vertices (or edges) which have to remain intact. Given m such important vertices or edges we show that a simplification hierarchy of size $O(n)$ and depth $O(\log(n/m))$ can be constructed by 2-step removals in $O(n)$ time, such that the simplified graph contains the m important vertices and edges, and at most $O(m)$ other vertices from the input graph. In some triangulations, many vertices may not even be 2-step removable. In order to provide the option to remove such vertices, we also define and examine k-step removals. This increases the lower bound on the number of removable vertices.

Chapter 4. Widespread availability of location aware devices (such as GPS receivers) promotes capture of detailed movement trajectories of people, animals, vehicles and other moving objects, opening new options for a better understanding of the processes involved. In this chapter we investigate spatio-temporal movement patterns in large tracking data sets. We present a natural definition of the pattern 'one object is leading others', which is based on behavioural patterns discussed in the behavioural ecology literature. Such leadership patterns can be characterised by a minimum time length for which they have to exist and by a minimum number of entities involved in the pattern. Furthermore, we distinguish two models (discrete and continuous) of the time axis for which patterns can start and end. For all variants of these leadership patterns, we describe algorithms for their detection, given the trajectories of a group of moving entities. A theoretical analysis as well as experiments show that these algorithms efficiently report leadership patterns.

Part II: Computing Networks

Chapter 5. To better handle situations where additional resources are available to carry out a task, many problems from the manufacturing industry involve dividing a task into a number of smaller tasks, while optimizing

a specific objective function. We consider the problem of partitioning a given set $\mathcal{S}$ of n points in the plane into k subsets, $\mathcal{S}_1, \ldots, \mathcal{S}_k$, such that $\max_{1 \leq i \leq k} |MST(\mathcal{S}_i)|$ is minimized. Variants of this problem arise in applications from the shipbuilding industry. We show that this problem is NP-hard, and we also present an approximation algorithm for the problem, in the case when k is a fixed constant. The approximation algorithm runs in time $O(n \log n)$ and produces a partition that is within a factor $(4/3 + \varepsilon)$ of the optimal if $k = 2$, and a factor $(2 + \varepsilon)$ otherwise.

Chapter 6. The gap property was introduced by Chandra et al. [30], who also proved the Gap Theorem. The theorem states that if a directed edge set of a point set V in $\mathcal{R}^d$ fulfills certain properties then the total weight of the edges in the set is bounded by $O(\log n \cdot wt(MST(V)))$. One of the constraints is that every vertex only can have one outgoing edge. In this chapter we extend the theorem and prove that a similar bound can be proven even though many edges have the same vertex as source as long as their sinks are not too close to each other.

We believe that the extended theorem can be applied to a wider range of geometric graphs and as an example we show how it can be used to develop a very simple algorithm to construct a t-spanner of low weight.

Part III: Assorted Problems

Chapter 7. A set of rectangles $\mathcal{S}$ is said to be *grid packed* if there exists a rectangular grid (not necessarily regular) such that every rectangle lies in the grid and there is at most one rectangle of $\mathcal{S}$ in each cell. The area of a grid packing is the area of a minimal bounding box that contains all the rectangles in the grid packing. We present an approximation algorithm that given a set $\mathcal{S}$ of rectangles and a real constant $\varepsilon > 0$ produces a grid packing of $\mathcal{S}$ whose area is at most $(1 + \varepsilon)$ times larger than an optimal grid packing in polynomial time. If ε is chosen large enough the running time of the algorithm will be linear. We also study several interesting variants, for example the smallest area grid packing containing at least $k \leq n$ rectangles, and given a region $\mathcal{A}$ grid pack as many rectangles as possible within $\mathcal{A}$. Apart from the approximation algorithms we present several hardness results.

1.8 Publications

Chapters 2 to 7 each correspond to a published paper. Below each chapter and corresponding publication is given.

Chapter 2: *Approximate Distance Oracles for Graphs with Dense Clusters.* Authors: Mattias Andersson, Joachim Gudmundsson and Christos Levcopoulos. In special issue of the CGTA (Computational Geometry, Theory and Applications) devoted to selected papers from EWCG'04 (European Workshop on Computational Geometry), August 2007, Volume 37, Issue 3, pp 142-154.

Chapter 3: *Restricted Mesh Simplification using Edge Contractions.* Authors: Mattias Andersson, Joachim Gudmundsson and Christos Levcopoulos. In Proc. of COCOON'06 (Computing and Combinatorics Conference), Taipei, Taiwan. Invited to special issue of IJCGA (International Journal of Computational Geometry and Applications) devoted to selected papers from COCOON'06.

Chapter 4: *Reporting Leadership Patterns among Trajectories.* Authors: Mattias Andersson, Joachim Gudmundsson, Patrick Laube and Thomas Wolle. In Proc. of ACM SAC ASIIS'07 (ACM Symposium on Applied Computing, Track: Advances in Spatial and Image-based Information Systems), Seoul, Korea. To appear in GeoInformatica.

Chapter 5: *Balanced Partition of Minimum Spanning Trees.* Authors: Mattias Andersson, Joachim Gudmundsson, Christos Levcopoulos and Giri Narasimhan. In special issue of IJCGA (Internation Journal of Computational Geometry and Applications), devoted to selected papers from CGA'02 (Computational Geometry and Applications), Volume 13, Nr. 4, pp 303-316.

Chapter 6: *Generalizing the Gap Theorem for Constructing a t-spanner of Low Weight Fast and Simple.* Authors: Mattias Andersson, Joachim Gudmundsson and Christos Levcopoulos. Technical Report, ISSN 1650-1276 Report 159, LU-CS-TR:2007-242, Department of Computer Science, Lund University, August 2007.

Chapter 7: *Chips on Wafers, or Packing Rectangles into Grids.* Authors: Mattias Andersson, Joachim Gudmundsson, Christos Levcopoulos. In special issue of CGTA (Computational Geometry, Theory and Applications) devoted to selected papers from EWCG'03 (European Workshop on Computational Geometry), Volume 30, Issue 2, February 2005, pp 95-111.

Part I

Processing Networks

Chapter 2

Approximate Distance Oracles for Graphs with Dense Clusters

The *shortest-path* (SP) problem for weighted graphs with n vertices and m edges is a fundamental problem for which efficient solutions can now be found in any standard algorithms text, see also [47, 68, 136, 156, 155]. Lately the approximation version of this problem has also been studied extensively [6, 37, 48]. In numerous algorithms, the query version of the SP-problem frequently appears as a subroutine. In such a query, we are given two vertices and have to compute or approximate the shortest path between them. Thorup and Zwick [157] presented an algorithm for undirected weighted graphs that computes $(2k-1)$-approximate solutions to the query version of the SP problem in $\mathcal{O}(k)$ time, using a data structure that takes expected time $\mathcal{O}(kmn^{1/k})$ to construct and utilizes $\mathcal{O}(kn^{1+1/k})$ space. It is not an approximation scheme in the true sense because the value k needs to be a positive integer. Since the query time is essentially bounded by a constant, Thorup and Zwick refer to their queries as approximate *distance oracles*. The time of pre-processing was recently improved by Baswana and Sen in [19].

We focus on the geometric version of this problem. A geometric graph has vertices corresponding to points in $\mathbb{R}^d$ and edge weights from a Euclidean metric. Throughout this chapter we will assume that d is a constant. A geometric graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is said to be a t-spanner for $\mathcal{V}$, if for any two points p and q in $\mathcal{V}$, there exists a path of length at most t times the Euclidean distance between p and q. For geometric graphs, also, considerable previous work exists on the shortest path and related problems. A good

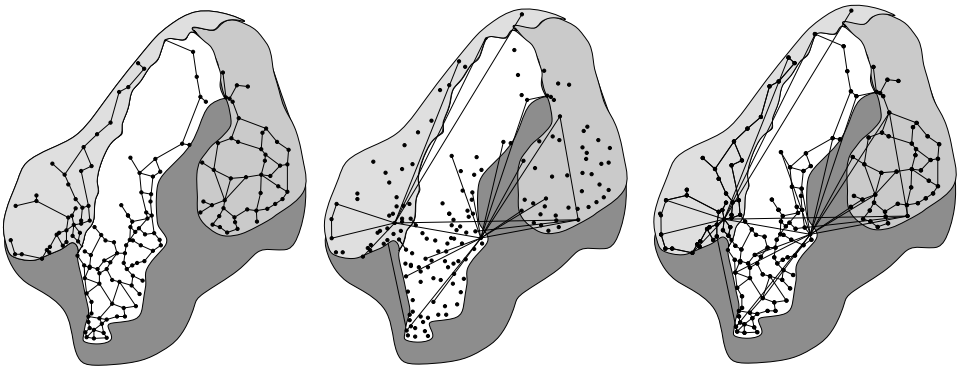

Figure 2.1: The three figures models parts of the transportation network in Norway, Sweden and Finland. (a) The domestic railway network in the three countries (not complete). (b) The railway connections between the countries together with the main air and sea connections within, and between, Norway, Sweden and Finland. (c) The two networks combined into one graph $\mathcal{G}$.

survey can be found in [126], see also [14, 32, 33, 35, 60, 61, 153]. The geometric query version was recently studied by Gudmundsson et al. [79, 80] and they presented the first data structure that answers approximate shortest-path queries in constant time, provided that the input graph is a t-spanner for some known constant $t > 1$. Their data structure uses $\mathcal{O}(n \log n)$ space and can be constructed in time $\mathcal{O}(m + n \log n)$.

In this chapter we extend the results in [79, 80] to hold also for "islands" of t-spanners, i.e., a set of N vertex disjoint t-spanners inter-connected through "airports" i.e. M edges of arbitrary non-negative weight. We construct a data structure that can answer $(1 + \varepsilon)$-approximate shortest path queries in constant time. The data structure uses $\mathcal{O}(M^2 + n \log n)$ space and can be constructed in time $\mathcal{O}(m + (M^2 + n) \log n)$, where m is the total number of edges. Hence, for $M = \mathcal{O}(\sqrt{n})$ the bound is essentially the same as in [79, 80].

We claim that the generalization studied is natural in many applications. Consider for example the freight costs within Norway, Sweden and Finland, see Fig. 2.1. The railway network and the road network within a country are usually t-spanners for some small value t, and the weight (transport cost) of an edge is linearly dependent on the Euclidean distance. In Fig. 2.1a the railway networks of Norway, Sweden and Finland (although not complete) is shown. The weight of an edge is dependent on the Euclidean distance between its endpoints. Hence, each country's railway network and road

network can most often be modeled as a Euclidean t-spanner for some small constant t. (Places that are not reachable by train are treated as single t- spanners containing only one point, for example Haugesund on the west coast of Norway is only reachable by boat.). Apart from these edges there are also edges that models, for example, air freight, sea freight, or inter-connecting railway transports. An example of this is shown Fig. 2.1b, where the main air and sea routes together with the inter-connecting railway tracks are shown. The weight of these edges can be completely independent of the Euclidean distance, as is usually the case when it comes to air fares. The reason why inter-connecting railway transports is included in the latter set of edges is because the railway networks of different countries are usually sparsely connected. For example, there is one connection between Sweden and Finland, two between Norway and Sweden and, zero between Finland and Norway. The same holds for many other adjacent countries, for example, there are three connections between the Netherlands and Germany, two between the Netherlands and Belgium, and three between France and Spain. Finally, note that in most cases M is very small compared to n.

In [79] it was shown that an approximate shortest-path distance oracle can be applied to a large number of problems, for example, finding shortest obstacle-avoiding path between two vertices in a planar polygonal domain with obstacles and interesting query versions of closest pair problems. The extension presented in this chapter also generalizes the results for the above mentioned problems.

The main idea for obtaining our results is to develop a method to efficiently combine existing methods for $\mathcal{O}(1)$-spanners with methods for general graphs. One problem, for example, may be given a starting point p and a destination q, which should be the first airport to travel to, since (in theory) there might be a non-constant number of airports on p's island? In order to achieve this, we determine a small number, $\mathcal{O}(M)$, of representative "junction" points, so that every point p in the graph is represented by exactly one such junction point $r(p)$, located on the same island as p. All airports are also treated as such junction points. For all pairs of junction points we precompute approximate distances, using space $\mathcal{O}(M^2)$. For any two points p and q, an approximately shortest path between them is found either by only using edges of one of the $\mathcal{O}(1)$-spanners, or by following a path from p to its representative junction point $r(p)$, then from $r(p)$ to $r(q)$, and finally from $r(q)$ to q. In order to choose such a small set of suitable representative junction points we present, in Section 2.2.4, a partition of space which may be useful also in other applications. In Section 5.3 we show general correctness, and in Section 2.4 we mention some refinements

and extensions of the main results.

2.1 Preliminaries

Our model of computation is the traditional algebraic computation model with the added power of indirect addressing. We will use the following notation. For points p and q in $\mathbb{R}^d$, $|p,q|$ denotes the Euclidean distance between p and q. If $\mathcal{G}$ is a geometric graph, then $\delta_{\mathcal{G}}(p,q)$ denotes the Euclidean length of a shortest path in $\mathcal{G}$ between p and q. If P is a path in $\mathcal{G}$ between p and q having length Δ with $\delta_{\mathcal{G}}(p,q) \leq \Delta \leq (1+\varepsilon)\cdot\delta_{\mathcal{G}}(p,q)$, then P is a $(1+\varepsilon)$-approximate shortest path for p and q.

The main result of this chapter is stated in the following theorem:

Theorem 2.1 *Consider two graphs $\mathcal{H}_1 = (\mathcal{V}, \mathcal{F}_1)$ and $\mathcal{H}_2 = (\mathcal{V}, \mathcal{F}_2)$, where $\mathcal{H}_1$ is a collection of N vertex disjoint Euclidean t-spanners ($t > 1$ is a constant), and $\mathcal{H}_2$ is a graph with M edges of non-negative weight. The union of the two graphs is denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E} = \{\mathcal{F}_1 \cup \mathcal{F}_2\})$.*

One can construct a data structure in time $\mathcal{O}((|\mathcal{E}| + M^2)\log n)$ using $\mathcal{O}(M^2 + n\log n)$ space that can answer $(1+\varepsilon)$-approximate shortest path queries in $\mathcal{G}$ in constant time, where $0 < \epsilon < 1$ is a given constant.

The set of pairwise vertex disjoint t-spanners of $\mathcal{H}_1$ is called the "islands" of $\mathcal{G}$ and will be denoted $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \ldots, \mathcal{G}_N = (\mathcal{V}_N, \mathcal{E}_N)$. An edge $(u,v) \in \mathcal{F}_2$ is said to be an *inter-connecting* edge (even though both its endpoints may belong to the same island). A vertex $v \in \mathcal{V}_i$ incident to an edge in $\mathcal{H}_2$ is called an *airport*, for simplicity (even though these vertices may represent any kind of junction point). The set of all airports of $\mathcal{V}_i$ is denoted $\mathcal{C}_i$. Note that the total number of airports is $\mathcal{O}(M)$ since the number of inter-connecting edges is M.

2.2 Tools

In the construction of the distance oracle we will need several tools, among them the well-separated pair decomposition by Callahan and Kosaraju [26], a graph pruning tool by Gudmundsson et al. [79, 81] and well-separated clusters by Krznaric and Levcopoulos [107]. In this section we briefly recollect these tools. In section 2.2.4 we also show a useful tool that clusters points with respect to a subset of representative points, as described in the introduction.

2.2.1 Well-separated pair decomposition

Definition 2.2 [26] Let $s > 0$ be a real number, and let $\mathcal{A}$ and $\mathcal{B}$ be two finite sets of points in $\mathbb{R}^d$. We say that $\mathcal{A}$ and $\mathcal{B}$ are well-separated with respect to s if there are two disjoint balls $C_\mathcal{A}$ and $C_\mathcal{B}$, having the same radius, such that $C_\mathcal{A}$ contains $\mathcal{A}$ and, $C_\mathcal{B}$ contains $\mathcal{B}$, and the distance between $C_\mathcal{A}$ and $C_\mathcal{B}$ is at least s times the radius of $C_\mathcal{A}$. We refer to s as the separation ratio.

Lemma 2.3 *[26] Let $\mathcal{A}$ and $\mathcal{B}$ be two sets of points that are well-separated with respect to s, let x and x' be two points of $\mathcal{A}$, and let y and y' be two points of $\mathcal{B}$. Then $|x, x'| \leq (2/s)|x', y'|$, and $|x', y'| \leq (1 + 4/s)|x, y|$.*

Definition 2.4 [26] Let S be a set of points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A well-separated pair decomposition (WSPD) for S with respect to s is a sequence $\{\mathcal{A}_i, \mathcal{B}_i\}, 1 \leq i \leq m$, of pairs of non-empty subsets of S, such that

1. $\mathcal{A}_i \cap \mathcal{B}_i = \emptyset$ for all $i = 1, \ldots, m$,
2. for each unordered pair $\{p, q\}$ of distinct points of S, there is exactly one pair $\{\mathcal{A}_i, \mathcal{B}_i\}$ in the sequence, such that (i) $p \in \mathcal{A}_i$ and $q \in \mathcal{B}_i$, or (ii) $q \in \mathcal{A}_i$ and $p \in \mathcal{B}_i$,
3. $\mathcal{A}_i$ and $\mathcal{B}_i$ are well-separated with respect to s for all $i = 1, \ldots, m$.

The integer m is called the size of the WSPD.

Callahan and Kosaraju show how such a WSPD can be computed. They start by constructing in $\mathcal{O}(n \log n)$ time, a split tree T having the points in S as leaves. Given this tree, they show how a WSPD of size $m = \mathcal{O}(s^d n)$ can be computed in time $\mathcal{O}(s^d n)$. In this WSPD, each pair $\{\mathcal{A}_i, \mathcal{B}_i\}$ is represented by two nodes u_i and v_i of T. That is, $\mathcal{A}_i$ and $\mathcal{B}_i$ are the sets of all points stored at the leaves of the subtrees rooted at u_i and v_i, respectively.

Theorem 2.5 *[26] Let S be a set of points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A WSPD for S with respect to s having size $\mathcal{O}(s^d n)$ can be computed in $\mathcal{O}(n \log n + s^d n)$ time.*

2.2.2 Pruning a t-spanner

In [79] it was shown that a simple way of pruning an existing t-spanner with m edges into a $(t(1 + \varepsilon))$-spanner with $\mathcal{O}(n)$ edges was to use the WSPD described in the previous section.

Assume that we are given a t-spanner $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Compute a WSPD $\{\mathcal{A}_i, \mathcal{B}_i\}$, $1 \leq i \leq \ell$, for $\mathcal{V}$, with separation constant $s = 4(1+(1+\varepsilon)t)/\varepsilon$ and $\ell = O(n)$. Let $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ be the graph that contains for each i, exactly one (arbitrary) edge (x_i, y_i) of E with $x_i \in \mathcal{A}_i$ and $y_i \in B_i$, provided such an edge exists. It holds that $\mathcal{G}'$ is a $(1+\varepsilon)$ spanner of $\mathcal{G}$, and hence:

Fact 2.6 *(Corollary 1 in [81]) Given a real constant $\epsilon > 0$ and a t-spanner $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for some real constant $t > 1$, with n vertices and m edges, one can compute a $(1+\epsilon)$-spanner $\mathcal{G}'$ of $\mathcal{G}$ with $\mathcal{O}(n)$ edges in time $\mathcal{O}(m+n\log n)$.*

2.2.3 Well-Separated Clusters

Let $\mathcal{S}$ be a set of points in the plane, and let $b \geq 1$ be a real constant. Let the *rectangular diameter* of $\mathcal{A} \in \mathcal{S}$, abbreviated $\mathrm{rd}(\mathcal{A})$, be the diameter of smallest axis-aligned rectangle containing $\mathcal{A}$. We may now consider the following cluster definitions from [107]:

Definition 2.7 A subset $\mathcal{A}$ of $\mathcal{S}$ is a *b-cluster* if $\mathcal{A}$ equals $\mathcal{S}$ or the distance between any point of $\mathcal{A}$ and any point of $\mathcal{S} - \mathcal{A}$ is greater than $b \cdot rd(\mathcal{A})$.

Definition 2.8 The *hierarchy of b-clusters* of $\mathcal{S}$ is a rooted tree whose nodes correspond to distinct b-clusters, such that the root corresponds to $\mathcal{S}$ and leaves to single points of $\mathcal{S}$. Let $\nu(\mathcal{A})$ be any internal node and let $\mathcal{A}$ be its corresponding b-cluster. The children of $\nu(\mathcal{A})$ correspond to every b-cluster $\mathcal{C}$ such $\mathcal{C} \subset \mathcal{A}$ and there is no b-cluster $\mathcal{B}$ such that $\mathcal{C} \subset \mathcal{B} \subset \mathcal{A}$.

The following observation is straightforward.

Observation 2.9 *Let $\mathcal{A}$ and $\mathcal{B}$ be two distinct b-clusters, and let x and x' be two points of $\mathcal{A}$, and let y and y' be two points of $\mathcal{B}$. Then $|x', y'| \leq (1 + 2/b)|x, y|$.*

Proof: Assume w.l.o.g. that $|x', y'| \geq |x, y|$ and that $rd(\mathcal{A}) \geq rd(\mathcal{B})$. This means $|x', y'| \leq |x, y| + 2rd(\mathcal{A}) \leq |x, y| + 2|x, y|/b = (1 + 2/b)|x, y|$. □

The cluster tree can also be computed efficiently.

Theorem 2.10 *Let $\mathcal{S}$ be a set of n points in $\mathbb{R}^d$ and a real constant $b \geq 1$, the hierarchy of b-clusters of $\mathcal{S}$ can be computed in $\mathcal{O}(n \log n)$ time and space.*

Proof: The hierarchy of b-clusters can easily be computed in $\mathcal{O}(n \log n)$ time for any constant number of dimensions d, e.g., by using a hierarchical cluster decomposition according to the complete-linkage criterion in the L_0-metric (see Krznaric and Levcopoulos [108]), since each such b-cluster is also a cluster in the complete-linkage hierarchy. □

2.2.4 Partitioning space into small cells

In this section, given a set $\mathcal{V}$ of n points in $\mathbb{R}^d$, and a subset $\mathcal{V}' \subseteq \mathcal{V}$, we show how to associate a representative point $r \in \mathcal{V}$ to each point $p \in \mathcal{V}$, such that the distance $|p, r| + |r, q|$, for any point $q \in \mathcal{V}'$, is a good approximation of the distance $|p, q|$. The total number of representative points is $\mathcal{O}(|\mathcal{V}'|)$. The idea is to partition space into cells, such that all points included in a cell may share a common representative point.

We will use the following fact by Arya et al. [17]:

Fact 2.11 *(Theorem 1 in [17]) Consider a set $\mathcal{S}$ of n points in $\mathbb{R}^d$. There is a constant $c_{d,\varepsilon} \leq d\lceil 1 + 6d/\varepsilon \rceil^d$, such that in $\mathcal{O}(dn \log n)$ time it is possible to construct a data structure of size $\mathcal{O}(dn)$, such that for any Minkowski metric:*

(i) Given any $\varepsilon > 0$ and $q \in \mathbb{R}^d$, a $(1+\varepsilon)$-approximate nearest neighbor of q in S can be reported in $\mathcal{O}(c_{d,\varepsilon} \log n)$.

(ii) More generally, given $\varepsilon_N > 0$, $q \in \mathbb{R}^d$, and any k, $1 \leq k \leq n$, a sequence of k $(1+\varepsilon)$-approximate nearest neighbors can be computed in $\mathcal{O}((c_{d,\varepsilon} + kd) \log n)$ time.

Computing representative points

As a pre-processing step we compute the b-cluster tree $\mathcal{T}$ of $\mathcal{V}'$ with $b = 10/\varepsilon^2$, as described in Theorem 2.10.

For a level i in $\mathcal{T}$ let $\nu(\mathcal{D}_1), \ldots, \nu(\mathcal{D}_{\ell_i})$ be the nodes at that level, where $\mathcal{D}_1, \ldots, \mathcal{D}_{\ell_i}$ are the associated clusters. For each cluster $\mathcal{D}_j$ pick an arbitrary vertex d_j as the center point of $\mathcal{D}_j$. The set of the ℓ_i center points is denoted $\mathcal{D}(i)$. Perform the following four steps for each level i of $\mathcal{T}$.

1. Compute an approximate nearest neighbor structure with $\mathcal{D}(i)$ as input, as described in Fact 2.11.

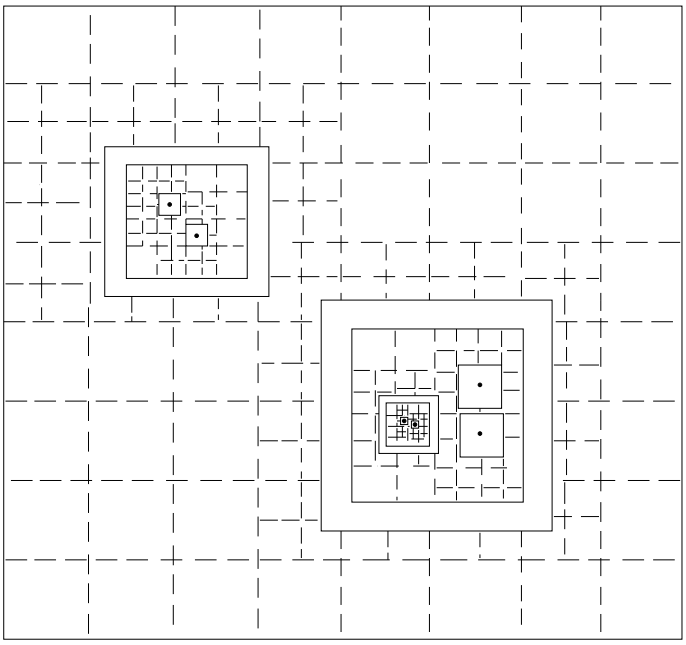

Figure 2.2: An example cell partition, with respect to $\mathcal{V}'$, made by the algorithm. Doughnuts are drawn with solid lines, while inner cells are drawn with dashed lines.

2. For each center point d_j in $\mathcal{D}(i)$ compute the $(1+\varepsilon)$-approximate nearest neighbor of d_j. The point returned by the structure is denoted v_j, where $v_j \neq d_j$.

3. For each cluster $\mathcal{D}_j$ construct two squares; $is(\mathcal{D}_j)$ and $os(\mathcal{D}_j)$ with centers at d_j and side length $2\alpha = 2(1+1/\varepsilon) \cdot rd(\mathcal{D}_j)$ and $2\beta = \frac{2\varepsilon|d_j,v_j|}{(1+\varepsilon)(1+2/b)}$ respectively, where $\alpha < \beta$. The two squares are called the inner and outer shells of $\mathcal{D}_j$, and the set theoretical difference between the inner and the outer shell is denoted the *doughnut* of $\mathcal{D}_j$.

4. The inner shell of $\mathcal{D}_j$ is recursively partitioned into four equally sized squares, until each square s either

   (a) is completely included in the union of the outer shells of the children of $\nu(\mathcal{D}_j)$. In this case the square is deleted and, hence, not further partitioned. Or,

   (b) has diameter at most $\frac{\varepsilon}{1+\varepsilon} \cdot K$, where K is the smallest distance be-

tween a point within s and a point in $\mathcal{D}_j$. A $(1+\varepsilon)$-approximation of K can be computed in time $\mathcal{O}(\log |\mathcal{D}_j|)$. This implies that the diameter of s is bounded by $\varepsilon \cdot K$.

The resulting cells are denoted *inner cells*. Note that, due to step 4a, every inner cell is empty of points from $\mathcal{D}_j$.

Finally, after all levels of $\mathcal{T}$ have been processed, we assign a representative point, $r(p)$, to each point p in $\mathcal{V}$. Preprocess all the produced cells and perform a point-location query for each point. If p belongs to a doughnut cell then the center point of the associated cluster (see step 1) is the representative point of p. Otherwise, if p belongs to an inner cell C and p is the first point within C processed in this step then $r(C)$ is set to p. If p is not the first point then $r(p) = r(C)$. Further, note that an inner cell may overlap with the union of the outer shells of the children of $\nu(\mathcal{D}_j)$. If a point is included in both an inner cell and an outer shell, we treat it as if it belonged to the inner cell, and assign a representative point as above.

The analysis

Below we prove the main result of this section.

Theorem 2.12 *Given a set $\mathcal{V}$ of n points in $\mathbb{R}^d$, a subset $\mathcal{V}' \subseteq \mathcal{V}$ and a positive real value $\tau_1 < 1$, the above algorithm associate for each point $p \in \mathcal{V}$ a representative point $r(p) \in \mathcal{V}$ such that for any point $q \in \mathcal{V}'$, it holds that*

$$min\{|p, r(p)|, |r(p), q|\} \leq \tau_1 |p, q|.$$

The number of representative points is $\mathcal{O}(|\mathcal{V}'|)$ and they can be computed in time $\mathcal{O}(n \log n)$.

The proof of Theorem 2.12 is partitioned into three steps: first we show that for each vertex $v \in \mathcal{V}$ the algorithm always choose a good representative point, then it will be shown that the total number of representative points is $\mathcal{O}(|\mathcal{V}'|)$, and finally we prove the time-complexity of the algorithm.

Lemma 2.13 *For each point $p \in \mathcal{V}$ and for every point $q \in \mathcal{V}'$ it holds that*

$$min\{|p, r(p)|, |r(p), q|\} \leq \tau_1 |p, q|.$$

Proof: Consider the above algorithm and select a positive constant $\varepsilon = (1 + \sqrt{2})\tau_1/\sqrt{8}$. For each point $p \in \mathcal{V}$ we distinguish between two cases depending on the cell C in which p lies, either in an inner cell or a doughnut.

inner cell: Let $p' \in \mathcal{V}'$ be the nearest neighbor of p in $\mathcal{V}'$. The distance between a point in C, and its nearest neighbor, can differ at most $\mathrm{rd}(C)$ from the distance between any other point in C and its nearest neighbor. Thus, from the way C was created it is straight-forward to see that $\mathrm{rd}(C) \leq \varepsilon \cdot |p, p'|$ and thus

$$min\{|p, r(p)|, |r(p), q|\} = |p, r(p)| \leq \mathrm{rd}(C) \leq \varepsilon |p, p'| \leq \varepsilon |p, q| \leq \tau_1 |p, q|.$$

doughnut: Let $cl(C)$ denote the cluster that was processed when C was created. We distinguish between two cases, either $q \in cl(C)$, or not.

- $\mathbf{q \in cl(C)}$: From the algorithm it holds that $|p, r(p)| \geq \alpha - rd(cl(C))$ and that $|r(p), q| \leq \mathrm{rd}(cl(C))$, which means that

$$\min\{|p, r(p)|, |r(p), q|\} = |r(p), q| \leq rd(cl(C)).$$

  It holds that $rd(cl(C))$ can be rewritten as $\varepsilon((1+1/\varepsilon) \cdot rd(cl(C)) - rd(cl(C)))$, hence since $\alpha = (1 + 1/\varepsilon) \cdot rd(cl(C))$ and since $rd(cl(C)) \leq \varepsilon |p, q|$ we get

$$\begin{aligned} \min\{|p, r(p)|, |r(p), q|\} &= |r(p), q| \\ &\leq rd(cl(C)) \\ &\leq \varepsilon(\alpha - rd(cl(C))) \\ &\leq \varepsilon |p, r(p)| \\ &\leq \varepsilon(|p, q| + rd(cl(C))) \\ &\leq 2\varepsilon |p, q| \leq \tau_1 |p, q| \end{aligned}$$

- $\mathbf{q \notin cl(C)}$: We know that from the assignment of representative points that $|p, r(p)| \leq 2\sqrt{2}\beta$. Let d_i be the center point of $cl(C)$ chosen by the algorithm, let $d_i' \in \mathcal{V}' \setminus c(C)$ be the point closest to d_i and let $p' \in \mathcal{V}' \setminus cl(C)$ be the point closest to p. From the definition of β it holds that $\beta \leq \varepsilon |d_i, d_i'| \leq \varepsilon |p, p'| - \sqrt{2}\beta \Rightarrow \beta \leq \varepsilon |p, p'|/(1 + \sqrt{2})$. As a result we get

$$\begin{aligned} min\{|p, r(p)|, |r(p), q|\} &\leq |p, r(p)| \\ &\leq 2\sqrt{2}\beta \\ &\leq 2\sqrt{2}\varepsilon |p, p'|/(1 + \sqrt{2}) \\ &\leq 2\sqrt{2}\varepsilon |p, q|/(1 + \sqrt{2}) \\ &= \tau_1 |p, q|. \end{aligned}$$

This concludes the proof of the lemma. □

Lemma 2.14 *The number of representative points is* $\mathcal{O}(|\mathcal{V}'|)$.

Proof: For each cluster there is (at most) one doughnut cell, thus $|\mathcal{V}'|$ in total, hence we only need to bound the number of inner cells.

Intuitively this is done as follows. Given a b-cluster $\mathcal{D}$ let $\nu(\mathcal{D})$ be the node in $\mathcal{T}$ associated with $\mathcal{D}$ and let $\mathcal{D}_1, \ldots, \mathcal{D}_\ell$ be the cluster associated with the children of $\nu(\mathcal{D})$. It will be shown that the number of inner cells of $\mathcal{D}$ is $\mathcal{O}(\ell)$, which means that we have $\mathcal{O}(|\mathcal{V}'|)$ inner cells in total. For each cluster $\mathcal{D}_i$ let d_i be the center point of $\mathcal{D}_i$, the set of these points is denoted $\mathcal{K}$ and $|\mathcal{K}| = \ell$. In the analysis we will consider the WSPD of $\mathcal{K}$ with a constant s as separation constant. For each well-separated pair $\{\mathcal{A}_i, \mathcal{B}_i\}$ we consider a disc of radius $\Theta(dist(\mathcal{A}_i, \mathcal{B}_i))$ surrounding the pair. We let each such disc "pay" for all cells of approximate size $dist(\mathcal{A}_i, \mathcal{B}_i)$ included in the circle, which, according to standard packing arguments, is a constant number of cells. We then show that each cell is paid for by at least one disc. Since the size of the WSPD according to Theorem 2.5 is $\mathcal{O}(\ell)$, it holds that the number of cells is $\mathcal{O}(\ell)$.

We are now ready to give a more detailed analysis. Using Observation 2.9 and Lemma 2.3 we obtain the following observation, used throughout this proof:

Observation 2.15 *Consider a well-separated pair* $\{\mathcal{A}_i, \mathcal{B}_i\}$ *and two center points* $d_j \in \mathcal{A}_i$ *and* $d_k \in \mathcal{B}_i$*, with corresponding b-clusters* $\mathcal{D}_j$ *and* D_k*. Given points* $x \in \mathcal{A}_i$*,* $y \in \mathcal{B}_i$*,* $x' \in \mathcal{D}_j$ *and* $y' \in \mathcal{D}_k$*, then* $|x', y'| \leq (1 + 4/s)(1 + 2/b)|x, y|$.

For simplicity of writing we will set $\tau = (1 + 4/s)(1 + 2/b)$. Next, for each well-separated pair $\{\mathcal{A}_i, \mathcal{B}_i\}$ choose two arbitrary points $a_i \in \mathcal{A}_i$ and $b_i \in \mathcal{B}_i$. Let $\mathcal{C}_i$ be the disc with center at a_i and of radius $17\tau(1 + 1/\varepsilon) \cdot |a_i, b_i|$. The aim is to show that each cell is paid for, that is, given a cell c we need to show that there exists a pair $\{\mathcal{A}_i, \mathcal{B}_i\}$ such that

(i) c intersects the disc $\mathcal{C}_i$, and

(ii) $\frac{1}{\gamma} \cdot rd(c) \leq |a_i, b_i| \leq \delta \cdot rd(c)$, where $\delta = \frac{68\tau(1+\varepsilon)(1+2/b)}{\varepsilon^2}$ and $\gamma = 4\tau$.

That is, the cell must overlap the disc surrounding $\{\mathcal{A}_i, \mathcal{B}_i\}$ and its diameter must be comparable to the distance between the points in $\mathcal{A}_i$ and the points in $\mathcal{B}_i$.

Consider an arbitrary inner cell c, let p be an arbitrary point of $\mathcal{V}$ within c and let q be the nearest neighbor in $\mathcal{V}'$ of p. Assume w.l.o.g. that $q \in \mathcal{D}_1$, and recall that d_1 is the center point of $\mathcal{D}_1$. Finally, let r be the point in $\{\mathcal{V}' \setminus \mathcal{D}_1\}$ closest to d_1. We will first show that there must exist well-separated pairs such that (i) holds. This will be shown by contradiction, where we distinguish between three cases:

Case 1: $|a_i, b_i| > \delta \cdot rd(c)$ for all pairs $\{\mathcal{A}_i, \mathcal{B}_i\}$ that contain d_1.

Since the bound holds for every well-separated pair containing d_1 it especially holds that $|d_1, r| > \frac{\delta}{\tau} \cdot rd(c) = \frac{68 \cdot rd(c)}{\varepsilon^2}$. Considering the side length β of $os(\mathcal{D}_1)$ it holds from the algorithm, and especially from the way that c was constructed, that

$$\beta > \frac{\varepsilon\delta}{\tau(1+\varepsilon)(1+2/b)} \cdot rd(c) = \frac{68}{\varepsilon} \cdot rd(c)$$

and

$$rd(c) > \frac{\varepsilon}{2}\Big(\frac{|p,q|}{1+\varepsilon} - \sqrt{2} \cdot rd(c)\Big) > \frac{\varepsilon}{2}\Big(\frac{1}{2}|p,q| - \sqrt{2} \cdot rd(c)\Big).$$

The second inequality can be rewritten and simplified: $|p,q| + rd(c) < \frac{8}{\varepsilon} \cdot rd(c)$. Now, since $|p,q| + rd(c) < \frac{8}{\varepsilon} \cdot rd(c) < \frac{\beta}{8}$ it holds that c is completely included in the outer shell of $\mathcal{D}_1$, which is a contradiction since c then would have been removed by the algorithm (step 4a).

Case 2: $|a_i, b_i| < rd(c)/\gamma$ for all pairs $\{\mathcal{A}_i, \mathcal{B}_i\}$ that contain d_1.

This means that $rd(\mathcal{V}') \leq \frac{\tau \cdot rd(c)}{\gamma}$, which can be rewritten as $rd(c) \geq \frac{\gamma \cdot rd(\mathcal{V}')}{\tau}$. From the construction of the inner cells it holds that $|p,q| \geq rd(c)/\varepsilon$. Combining these two inequalities we obtain that

$$|p,q| \geq \frac{rd(c)}{\varepsilon} \geq \frac{\gamma \cdot rd(\mathcal{V}')}{\varepsilon \cdot \tau} = \frac{4 \cdot rd(\mathcal{V}')}{\varepsilon} > 2\alpha.$$

Thus, c must lie partly outside the inner shell, which is a contradiction since c was created by partitioning the inner shell.

Case 3: Neither Case 1 or 2 holds.

This means we have at least one well-separated pair such that $|a_i, b_i| < rd(c)/\gamma$, and at least one pair such that $|a_i, b_i| > \delta \cdot rd(c)$ for all pairs $\{\mathcal{A}_i, \mathcal{B}_i\}$ in which d_1 is included.

First consider the union of all well-separated pairs where $|a_i, b_i| < rd(c)/\gamma$. Each point in this union is included in one cluster $\mathcal{D}_i$. Consider the union of all such clusters, denoted $\mathcal{U}$. We have, for any point $u \in \mathcal{U}$ that $|d_1, u| < \tau \cdot rd(c)/\gamma \Rightarrow rd(\mathcal{U}) \leq 2\sqrt{2}\tau \cdot rd(c)/\gamma$.

Next consider all well-separated pairs $\{\mathcal{A}_i, \mathcal{B}_i\}$ such that $|a_i, b_i| > \delta \cdot rd(c)$, and assume w.l.o.g. that $d_1 \in \mathcal{A}_i$. Each point in the union of all $\mathcal{B}_i$'s is included in one cluster $\mathcal{D}_i$, and we let $\mathcal{U}'$ denote the union of all these clusters. Let u and u' be two points in $\mathcal{U}$ and $\mathcal{U}'$, respectively. We have that

$$|u, u'| \geq |d_1, u'| - rd(\mathcal{U}) \geq \frac{\delta \cdot rd(c)}{\tau} - rd(\mathcal{U}) \geq \frac{67}{\varepsilon^2} \cdot rd(\mathcal{U}) \geq b \cdot rd(\mathcal{U}),$$

since $b = 10/\varepsilon^2$. Thus $\mathcal{U}$ must be a b-cluster, which means there cannot exist well-separated pairs such that $rd(c)/\gamma > |a_i, b_i|$, which is a contradiction.

We have shown that for every cell c there exists a well-separated pair $\{\mathcal{A}_i, \mathcal{B}_i\}$ such that *(i)* holds, and such that $d_1 \in \{\mathcal{A}_i \cup \mathcal{B}_i\}$. In order to show (ii), we consider a well-separated pair $\{\mathcal{A}_i, \mathcal{B}_i\}$ that contains d_1 and such that $|a_i, b_i| \geq rd(c)/\gamma$. Recall that the radius of $\mathcal{C}_i$ is $17\tau(1 + 1/\varepsilon) \cdot |a_i, b_i|$. Let x denote the center of $\mathcal{C}_i$, it holds that

$$|p, x| < |p, q| + \tau \cdot |a_i, b_i| \leq \left(\frac{2}{\varepsilon} + 1\right) \cdot rd(c) + \tau \cdot |a_i, b_i| < 8\tau\left(1 + \frac{1}{\varepsilon}\right) \cdot |a_i, b_i|.$$

Since the distance from the center $\mathcal{C}_i$ to p is less than the radius of $\mathcal{C}_i$ it follows that c must overlap $\mathcal{C}_i$ and, hence, *(ii)* holds. In conclusion, we have $\mathcal{O}(|\mathcal{V}'|)$ cells and, since every cell has at most one representative point, the number of representative points is also $\mathcal{O}(|\mathcal{V}'|)$. □

Lemma 2.16 *The representative points can be computed in time* $\mathcal{O}(n \log n)$.

Proof: The preprocessing, building the b-cluster tree and selecting the center points of each cluster takes $\mathcal{O}(n \log n)$ time in total. An approximate nearest neighbor data structure is constructed on each level, but the total number of elements involved, summing over all levels, is at most $2|\mathcal{V}'|$. This follows since the number of leaves in $\mathcal{T}$ is $|\mathcal{V}'|$, and hence the total number of nodes in $\mathcal{T}$ is $2|\mathcal{V}'|$. It immediately follows that step 2 takes $2|\mathcal{V}'| \cdot \mathcal{O}(\log |\mathcal{V}'|)$ time in total and step 3 takes $\mathcal{O}(|\mathcal{V}'|)$ time.

The final step of the algorithm is done by performing n point-location queries, each taking $\mathcal{O}(\log n)$ time.

Hence, it remains to bound step 4, which is equivalent to bound the total number of squares considered during the partition. From Lemma 2.14, we know that the number of cells in the partition is $\mathcal{O}(|\mathcal{V}'|)$. However, the running time of the algorithm depends on the total number of squares considered during the construction of the partitioning. We will below show that the total number of squares also is bounded by $\mathcal{O}(|\mathcal{V}'|)$.

Consider an inner shell $\mathcal{I}$ and its corresponding b-cluster $\mathcal{D}$. Let $v(\mathcal{D})$ be the node in $\mathcal{T}$ associated with $\mathcal{D}$ and let $\mathcal{D}_1, \ldots, \mathcal{D}_\ell$ be the clusters associated with the children of $\nu(\mathcal{D})$. Further, consider the partition tree $\mathcal{Y}$ of $\mathcal{I}$, where the nodes correspond to squares in the natural way, and the leaf nodes correspond to either:

(i) inner cells included in the final partitioning, or

(ii) squares which were removed because they were completely included in an outer shell of $\mathcal{D}_1, \ldots \mathcal{D}_\ell$, and thus not partitioned further.

Let f be a node in $\mathcal{Y}$ and let c be its corresponding cell, such that the corresponding cell of one of its sibling nodes was removed due to inclusion in the outer shell $os(\mathcal{D}_i)$ of a cluster $\mathcal{D}_i$.

Since the cell of the sibling node was removed, and sibling cells are of equal size, it is clear that c must be smaller than $os(\mathcal{D}_i)$. Next, consider a leaf node $g \in \mathcal{Y}$ of type *(i)* and its corresponding cell c' resulting from the continued partitioning of c. We know that $os(\mathcal{D}_i)$ is greater than $is(\mathcal{D}_i)$, which, in turn is larger than $rd(\mathcal{D}_i)$ multiplied by some large constant factor based on ε. Further, since a part of g lies outside $os(\mathcal{D}_i)$ and is not partitioned further once it is at a sufficient distance from the points in $\mathcal{D}_i$, it is straight-forward to see that the size of c' is at least $rd(\mathcal{D}_i)$ multiplied by some constant factor including ε. Thus c is only a constant factor larger than c', which means that c' was constructed by partitioning c a constant number of times.

Our goal is to show that we have $O(\ell)$ nodes in $\mathcal{Y}$, since this immediately implies that the total number of considered squares are $\mathcal{O}(|\mathcal{V}'|)$. From Lemma 2.13 we know that we have $\mathcal{O}(\ell)$ leaf nodes of type *(i)*. Consider such a type *(i)* leaf node $g \in \mathcal{Y}$, and all ancestors $\mathcal{A}(g)$ up to the first node such that none of its children is a type *(ii)* node. We let g pay for all children of all nodes in $\mathcal{A}(g)$, which are of type *(ii)*. From the above reasoning it is straight-forward to see that each type *(i)* leaf node pays for a constant number of type *(ii)* leaf nodes. Further each type *(ii)* leaf node $h \in \mathcal{Y}$ must be paid for by a type *(i)* leaf node. This holds, since at least one of its siblings h' is not a type *(ii)* leaf node (if all siblings were of type *(ii)* then

their parents would by type *(ii)*, which is a contradiction), and thus h must be paid for by the leaf node of type *(i)* resulting from the partitioning of h'.

This means that the total number of leaf nodes (type either *(i)* or *(ii)*) are $\mathcal{O}(\ell)$. Further, since the number of internal nodes in $\mathcal{Y}$ is at most a constant factor larger than the total number of leaf nodes it follows that the total number of nodes in $\mathcal{Y}$ is $\mathcal{O}(\ell)$, and thus, the total number of considered squares is $\mathcal{O}(|\mathcal{V}'|)$. □

2.3 Constructing the Oracle

This section is divided into three subsections: first we present the construction of the structure, then how queries are answered and, finally the analysis is presented.

Consider two graphs $\mathcal{H}_1 = (\mathcal{V}, \mathcal{F}_1)$ and $\mathcal{H}_2 = (\mathcal{V}, \mathcal{F}_2)$ with the same vertex set, where $\mathcal{H}_1$ is a collection of N vertex disjoint Euclidean t-spanners $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, $1 \leq i \leq N$, with m edges where $t > 1$ is a constant, and $\mathcal{H}_2$ is a graph with M edges of non-negative weight. The union of the two graphs is denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E} = \{\mathcal{F}_1 \cup \mathcal{F}_2\})$.

2.3.1 Constructing the basic structures

In this section we show how to pre-process $\mathcal{G}$ in time $\mathcal{O}(m + (M^2 + n) \log n)$ such that we obtain three structures that will help us answer $(1 + \varepsilon)$-approximate distance queries in constant time. We will assume that the number of edges in each subgraph is linear with respect to the number of vertices in $\mathcal{V}_i$, if not the subgraph is pruned using Fact 2.6. Hence, we can from now on assume that $|\mathcal{E}_i| = \mathcal{O}(\mathcal{V}_i)$.

Let $\mathcal{V}'$ be the set of vertices in $\mathcal{V}$ incident on an inter-connecting edge. Now we can apply Theorem 2.12 with parameters $\mathcal{V}$, $\mathcal{V}' = \Gamma'$ and τ_1 to obtain a representative point for each point in $\mathcal{V}$.

Now we are ready to present the three structures:

Oracle A: An oracle that given points p and q returns a 3-tuple $[SI, r(p), r(q)]$, where SI is a boolean with value 'true' if p and q belongs to the same island, otherwise it is 'false', and $r(p)$ and $r(q)$ is the representative points for p and q respectively.

Oracle B: An $(1 + \varepsilon)$-approximate distance oracle for any pair of points belonging to the same island.

Matrix D: An $\mathcal{O}(M) \times \mathcal{O}(M)$ matrix. For each pair of representative points, p and q, D contains the $(1+\varepsilon)$-approximate shortest distance between p and q.

The representative point of a point p is denoted $r(p)$, and the set of all representative points of $\mathcal{V}_i$ and $\mathcal{V}$ is denoted Γ_i and Γ, respectively. Note that $\mathcal{C}_i \subseteq \Gamma_i$. Now we turn our attention to the construction of the oracles and the matrix.

Oracle A:

The oracle is a 4-level tree, denoted $\mathcal{T}$, with the points of $\mathcal{V}$ corresponding to the leaves of $\mathcal{T}$. The parents of the leaves correspond to the representative points of $\mathcal{V}$ and their parents correspond to the islands $\mathcal{G}_1, \ldots, \mathcal{G}_N$ of $\mathcal{G}$. Finally, the root of $\mathcal{T}$ corresponds to $\mathcal{G}$. Since the representative points already are computed, the tree $\mathcal{T}$ can be constructed in linear time. The root is at level 0 and the leaves are at level 3 in $\mathcal{T}$.

Assume that one is given two points p and q. Follow the paths from p and q respectively to the root of $\mathcal{T}$. If p and q have the same ancestor at level 1 then they lie on the same island and hence SI is set to 'true', otherwise to 'false'. Finally, the ancestor of p and the ancestor of q at level 2 corresponds to the representative points of p and q. Obviously a query can be answered in constant time since the number of levels in $\mathcal{T}$ is four.

Oracle B:

This oracle is the structure that is easiest to build since we can apply the following result to each of the islands. Combining Theorem 1 in [80] with Corollary 1 in [81] we get the following fact (see also [79]):

Fact 2.17 *Let $\mathcal{V}$ be a set of n points in R^d, let τ_2 be a positive real constant and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a t-spanner for $\mathcal{V}$, for some real constant $t > 1$, having m edges. In $\mathcal{O}(n \log n)$ time we can preprocess $\mathcal{G}$ into a data structure of size $\mathcal{O}(n \log n)$, such that for any two points p and q in $\mathcal{V}$, we can in constant time compute a $(1+\tau_2)$-approximation to the shortest-path distance in $\mathcal{G}$ between p and q.*

Hence, oracle B will actually be a collection of oracles, one for each island. Given two points p and q the appropriate oracle can easily be found in constant time using a similar construction as for oracle A. Thus, after $\mathcal{O}(n \log n)$ pre-processing using $\mathcal{O}(n \log n)$ space, $(1+\tau_2)$-approximate

shortest path queries between points on the same island can be answered in constant time.

Matrix D:

For each i, $1 \leq i \leq N$, compute the WSPD of Γ_i with separation constant $s = (\frac{1+\tau_2+\tau_3}{\tau_3-\tau_2})$. As output we obtain a set of well-separated pairs $\{\mathcal{A}_i, \mathcal{B}_i\}_{1 \leq i \leq w_i}$, such that $w_i = \mathcal{O}(|\mathcal{C}_i|)$. Next, construct the non-Euclidean graph $\mathcal{F} = (\Gamma, \mathcal{E}')$ as follows. For each Γ_i and each well-separated pair $\{\mathcal{A}_j, \mathcal{B}_j\}$ of the WSPD of Γ_i select two (arbitrary) representative points $a_j \in \mathcal{A}_j$ and $b_j \in \mathcal{B}_j$. Add the edge (a_j, b_j) to $\mathcal{E}'$ with weight $B_i(a_j, b_j)$, where $B_i(p, q)$ denotes a call to oracle B_i for $\mathcal{G}_i$ with parameters p and q. Note that the graph $\mathcal{F}$ will have $\mathcal{O}(M)$ vertices and edges.

Let D be an $\mathcal{O}(M) \times \mathcal{O}(M)$ matrix. For each representative point $p \in \Gamma$ compute the single-source shortest path in $\mathcal{F}$ to every point q in Γ and store the distance of each path in $D[p, q]$. The total time for this step is $\mathcal{O}(M^2 \log M)$, and it can be obtained by running Dijkstra's algorithm M times.

Lemma 2.18 *The oracles A and B, and the matrix D can be built in time $\mathcal{O}(m + (M^2 + n) \log n)$ and the total complexity of A, B and M is $\mathcal{O}(M^2 + n \log n)$.*

Proof: The lemma is obtained by adding up the complexity for the pre-processing together with the cost of building each structure. Recall that as pre-processing steps we first pruned the subgraphs and then we computed the representative point for each point in $\mathcal{V}$. This was done in $\mathcal{O}(m+n \log n)$ time using $\mathcal{O}(n \log n)$ space, according to Fact 2.6 and Theorem 2.12. Next, oracle A was constructed in linear time using linear space, followed by the construction of oracle B which, according to Fact 2.17 was done in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(n \log n)$ space. Finally, the matrix D was constructed by first computing the graph $\mathcal{F}$. Then a single-source shortest path query was performed for each vertex in $\mathcal{F}$. Since the complexity of $\mathcal{F}$ is $\mathcal{O}(M)$ it follows that D was computed in time $\mathcal{O}(M^2 \log n)$ using $\mathcal{O}(M^2)$ space. Hence, adding these bounds gives the lemma. □

2.3.2 Querying

Given the two oracles and the matrices presented above the query algorithm is very simple, see pseudo-code below. Let $r(p)$ denote the representative

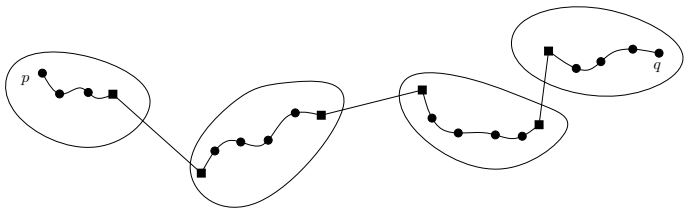

Figure 2.3: Illustrating the approximate shortest path between p and q. The boxes illustrate the "airports" along the path.

point of $p \in \mathcal{V}$. Now assume that we are given two points p and q. If p and q belong to the same islands then we query Oracle B with input p, q and return the value obtained from the oracle. If p and q does not belong to the same island we return the sum of $B(p, r(p))$, $D(r(p), r(q))$ and $B(r(q), q)$. Obviously this is done in constant time.

QUERY(p, q)
1. [SameIsland, $r(p), r(q)] \leftarrow A(p, q)$
2. $distance \leftarrow B(p, r(p)) + D(r(p), r(q)) + B(r(q), q)$
3. **if** SameIsland **then**
4. $\quad distance \leftarrow \min(distance, B(p, q))$
5. return $distance$

2.3.3 Correctness

Let $\delta_{\mathcal{G}}(p, q)$ be a shortest path in a graph $\mathcal{G}$ between two points p and q.

Observation 2.19 *Let p and q be any pair of points in $\mathcal{V}_i$ it holds that $B(p, q) \leq (1 + \tau_2) \cdot \delta_{\mathcal{G}_i}(p, q)$.*

Observation 2.20 *Given a point $p \in \mathcal{V}_i$ it holds that $\delta_{\mathcal{G}_i}(p, r(p)) \leq (1 + \tau_4) \cdot \delta_{\mathcal{G}}(p, r(p))$.*

Proof: Let h be the first point in $\mathcal{C}_i$ along the path $\delta_{\mathcal{G}}(p, r(p))$ from p to $r(p)$. If $r(p) = h$ then we are done, otherwise we will have two cases according to Theorem 2.12.

(a) If $|p, r(p)| \leq \tau_1 \cdot |p, h|$ then $\delta_{\mathcal{G}_i}(p, r(p)) \leq \tau_1 t \cdot |p, h|$ which is less than $|p, h|$, hence this case cannot occur.

(b) Otherwise, $|r(p), h| \leq \tau_1 \cdot |p, h|$ and hence $\delta_{\mathcal{G}_i}(p, r(p)) \leq \delta_{\mathcal{G}}(p, h) + \delta_{\mathcal{G}}(h, r(p)) \leq (1+\tau_1 t) \cdot \delta_{\mathcal{G}}(p, h)$. The observation follows by setting $\tau_4 = t \cdot \tau_1$. □

Lemma 2.21 *Let p and q be any pair of representative points in $\mathcal{V}_i$ it holds that $D(p,q) \leq (1+\tau_3) \cdot \delta_{\mathcal{G}_i}(p,q)$.*

Proof: Note that it suffices to prove that $D(p,q) \leq \frac{1+\tau_3}{1+\tau_2} \cdot B(p,q)$, according to Observation 2.19. The proof is done by induction on the Euclidean length of (p,q).

Base case: Recall that $\mathcal{F} = (\Gamma, \mathcal{E}')$ was constructed to build the matrix D. Assume that (p,q) is the closest pair of Γ. In this case there exists a well-separated pair $\{\mathcal{A}_j, \mathcal{B}_j\}$ such that $\mathcal{A}_j = \{p\}$ and $\mathcal{B}_j = \{q\}$ otherwise (p,q) could not be the closest pair. Hence the claim holds since there is an edge in $\mathcal{F}$ of weight $B(a_j, b_j)$.

Induction hypothesis: Assume that the lemma holds for all pairs in Γ closer than $|p,q|$ to each other.

Induction step: If $(p,q) \notin \mathcal{F}$ then there exists an edge (x,y) in $\mathcal{F}$ and a well-separated pair $\{\mathcal{A}_j, \mathcal{B}_j\}$ such that $x, p \in \mathcal{A}_j$ and $y, q \in \mathcal{B}_j$. According to the induction hypothesis there is path between p and x of weight $\frac{1+\tau_3}{1+\tau_2} \cdot D(p,q)$ and a path between y and q of weight $\frac{1+\tau_3}{1+\tau_2} \cdot D(p,q)$. Also, the weight of the edge (x,y) is $B(x,y)$. Putting together the weights we obtain that

$$\begin{aligned}
\delta_{\mathcal{F}}(p,q) &\leq \frac{1+\tau_3}{1+\tau_2} B(p,x) + B(x,y) + \frac{1+\tau_3}{1+\tau_2} B(y,q) \\
&\leq \left(2\frac{1+\tau_3}{1+\tau_2}(2/s) + (1+4/s)\right) \cdot B(p,q) \\
&= \frac{1+\tau_3}{1+\tau_2} B(p,q) = (1+\tau_3) \cdot \delta_{\mathcal{G}_i}(p,q)
\end{aligned}$$

In the third step we used the fact that $s = \frac{1+\tau_2+\tau_3}{\tau_3-\tau_2}$. □

Corollary 2.22 *Let p and q be any representative points in $\mathcal{V}$ it holds that $D(p,q) \leq (1+\tau_3) \cdot \delta_{\mathcal{G}}(p,q)$.*

Proof: Since all inter-connecting edges in $\mathcal{G}$ also is in $\mathcal{F}$ we can apply Lemma 2.21 to obtain the corollary. □

Lemma 2.23 *Given a pair of points $p, q \in \mathcal{V}$ it holds that*

$$\delta_{\mathcal{G}}(p, r(p)) + \delta_{\mathcal{G}}(r(p), r(q)) + \delta_{\mathcal{G}}(r(q), q) \leq (1+\tau_5) \cdot \delta_{\mathcal{G}}(p,q).$$

Proof: Let $h(p)$ and $h(q)$ denote the points in $\mathcal{C}$ that is first encountered when following the path $\delta_{\mathcal{G}}(p,q)$ from p to q and from q to p respectively. According to Theorem 2.12 there are four cases to consider.

1. $|p, r(p)| \leq \tau_1 \cdot |p, h(p)|$ and $|q, r(q)| \leq \tau_1 \cdot |q, h(q)|$.

$$\begin{aligned}
\delta_{\mathcal{G}}(p,q) &\leq \delta_{\mathcal{G}}(p,r(p)) + \delta_{\mathcal{G}}(r(p),r(q)) + \delta_{\mathcal{G}}(r(q),q) \\
&\leq \delta_{\mathcal{G}}(p,r(p)) + \delta_{\mathcal{G}}(r(p),p) + \delta_{\mathcal{G}}(p,h(p)) \\
&\quad + \delta_{\mathcal{G}}(h(p),h(q)) + \delta_{\mathcal{G}}(q,h(q)) \\
&\quad + \delta_{\mathcal{G}}(q,r(q)) + \delta_{\mathcal{G}}(r(q),q) \\
&\leq (1+2t\tau_1)\delta_{\mathcal{G}}(p,h(p)) + \delta_{\mathcal{G}}(h(p),h(q)) \\
&\quad + (1+2t\tau_1)\delta_{\mathcal{G}}(h(q),q) \\
&< (1+2t\tau_1) \cdot \delta_{\mathcal{G}}(p,q)
\end{aligned}$$

2. $|r(p), h(p)| \leq \tau_1 \cdot |p, h(p)|$ and $|r(q), h(q)| \leq \tau_1 \cdot |q, h(q)|$.

$$\begin{aligned}
\delta_{\mathcal{G}}(p,q) &\leq \delta_{\mathcal{G}}(p,r(p)) + \delta_{\mathcal{G}}(r(p),r(q)) + \delta_{\mathcal{G}}(r(q),q) \\
&\leq \delta_{\mathcal{G}}(p,h(p)) + \delta_{\mathcal{G}}(h(p),r(p)) + \delta_{\mathcal{G}}(r(p),h(p)) \\
&\quad + \delta_{\mathcal{G}}(h(p),h(q)) + \delta_{\mathcal{G}}(h(q),r(q)) \\
&\quad + \delta_{\mathcal{G}}(r(q),h(q)) + \delta_{\mathcal{G}}(h(q),q)) \\
&\leq (1+2t\tau_1)\delta_{\mathcal{G}}(p,h(p)) + \delta_{\mathcal{G}}(h(p),h(q)) \\
&\quad + (1+2t\tau_1)\delta_{\mathcal{G}}(h(q),q) \\
&< (1+2t\tau_1) \cdot \delta_{\mathcal{G}}(p,q)
\end{aligned}$$

3. $|p, r(p)| \leq \tau_1 \cdot |p, h(p)|$ and $|r(q), h(q)| \leq \tau_1 \cdot |q, h(q)|$.

$$\begin{aligned}
\delta_{\mathcal{G}}(p,q) &\leq \delta_{\mathcal{G}}(p,r(p)) + \delta_{\mathcal{G}}(r(p),r(q)) + \delta_{\mathcal{G}}(r(q),q) \\
&\leq \delta_{\mathcal{G}}(p,r(p)) + \delta_{\mathcal{G}}(r(p),p) + \delta_{\mathcal{G}}(p,h(p)) \\
&\quad + \delta_{\mathcal{G}}(h(p),h(q)) + \delta_{\mathcal{G}}(h(q),r(q)) \\
&\quad + \delta_{\mathcal{G}}(r(q),h(q)) + \delta_{\mathcal{G}}(h(q),q)) \\
&\leq (1+2t\tau_1)\delta_{\mathcal{G}}(p,h(p)) + \delta_{\mathcal{G}}(h(p),h(q)) \\
&\quad + (1+2t\tau_1)\delta_{\mathcal{G}}(h(q),q) \\
&< (1+2t\tau_1) \cdot \delta_{\mathcal{G}}(p,q)
\end{aligned}$$

4. $|p, r(p)| \leq \tau_1 \cdot |p, h(p)|$ and $|r(q), h(q)| \leq \tau_1 \cdot |q, h(q)|$. See case 3.

The lemma follows by setting $\tau_5 = 2t\tau_1$. □

Lemma 2.24 *Let p and q be any points in $\mathcal{V}$ it holds that*

$$\delta_{\mathcal{G}}(p,q) \leq B(p,r(p)) + D(r(p),r(q)) + B(r(q),q) \leq (1+\varepsilon) \cdot \delta_{\mathcal{G}}(p,q).$$

Proof:

$$\begin{aligned}
\delta_{\mathcal{G}}(p,q) &\leq B(p,r(p)) + D(r(p),r(q)) + B(r(q),q) \\
&\leq (1+\tau_2)\cdot\delta_{\mathcal{G}_i}(p,r(p)) + (1+\tau_3)\cdot\delta_{\mathcal{G}}(r(p),r(q)) \\
&\qquad +(1+\tau_2)\cdot\delta_{\mathcal{G}_j}(r(q),q) \\
&\leq (1+\tau_2)(1+\tau_4)\cdot\delta_{\mathcal{G}}(p,r(p)) + (1+\tau_3)\cdot\delta_{\mathcal{G}}(r(p),r(q)) \\
&\qquad +(1+\tau_2)(1+\tau_4)\cdot\delta_{\mathcal{G}}(r(q),q) \\
&< (1+\tau_4)(1+\tau_3)\cdot(\delta_{\mathcal{G}}(p,r(p)) + \delta_{\mathcal{G}}(r(p),r(q)) + \delta_{\mathcal{G}}(r(q),q)) \\
&\leq (1+\tau_4)(1+\tau_3)(1+\tau_5)\cdot\delta_{\mathcal{G}}(p,q) \\
&= (1+\varepsilon)\cdot\delta_{\mathcal{G}}(p,q)
\end{aligned}$$

On line 1 we used Observation 2.19 together with Lemma 2.21. On the following line we used Observation 2.20, applied Lemma 2.23 and finally replaced $(1+\tau_2)(1+\tau_3)(1+\tau_5)$ with $(1+\varepsilon)$. □

Putting together Lemma 2.18 and Lemma 2.24 gives us Theorem 2.1.

2.4 Refinements and extensions

Below is listed a number of refinements and extensions:

1. A refined analysis yields that the data structure of Theorem 2.1 only uses $\mathcal{O}(|\mathcal{C}|^2 + n\log n)$ space, where $\mathcal{C}$ is the set of all airports.

2. The data structure can be modified to handle the case when each island $\mathcal{G}_i$ is a t_i-spanner, i.e., every island has different (although constant) dilation.

3. If we allow $(2k-1)$-approximations, where k is a positive integer, we can construct a data structure in $\mathcal{O}(n\log n + kM|C|^{1/k})$ time using $\mathcal{O}(n\log n + k|C|^{1+1/k})$ space, by replacing the usage of Matrix D with the method for general graphs used by Thorup and Zwick [157].

Chapter 3

Restricted Mesh Simplification Using Edge Contractions

In computer graphics, objects are commonly represented using triangle meshes. One important problem regarding these meshes is how to efficiently simplify them, while maintaining a good approximation of the original mesh. As an example, scanners often produce information-redundant meshes containing millions of vertices and triangles. Further, often the simplification should be performed in several rounds, such that a level-of-detail hierarchy is constructed. One application of such a hierarchy is that an appropriate level may be chosen depending on viewing distance, as finer details tend to be unnecessary as the distance increases. Other applications include progressive transmission and efficient storing.

It is common to represent the level-of-detail hierarchy as a directed,

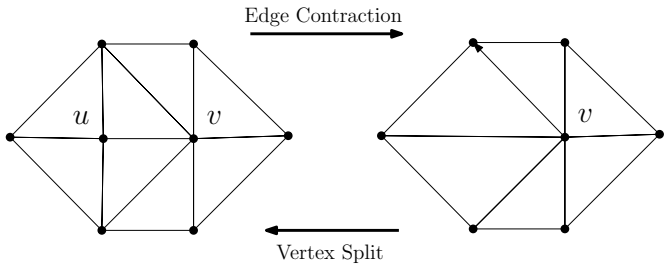

Figure 3.1: An edge contraction, and its inverse operation a vertex split.

acyclic and hierarchical graph, where each level in the graph corresponds to a level in the level-of-detail hierarchy, and where each node in the graph corresponds to a triangle. The first, top-most, level in the graph corresponds to the input mesh. When a contraction is made two triangles disappear, and one or more triangles are affected in such a way that their appearance change. In the graph this is represented with edges between disappearing triangles at some level i, and the affected triangles at level $i + 1$. The efficiency of a simplification algorithm is directly related to the size [66, 166] and depth of the hierarchy graph that it produces. Simplification algorithms constructing hierarchies of size $O(n)$ and depth $O(\log n)$ have been presented for several problem variants [34, 44, 53, 102].

Another related problem [9, 24] is where a triangulation is not simplified but transformed into a different triangulation, possibly on a set of different points. The transformation is done using edge flips and point moves.

Mesh simplification is generally regarded as a mature field (see [72] for a survey), consisting of several suggested methods and problem variants. In this chapter we consider the method of iteratively contracting edges [34, 73, 86, 91], where contractions are made such that no edge crossings occur during the process. Many of the results in previous papers were achieved using such standard edge contractions.

In this chapter we impose a new restriction, namely that an edge must be contracted onto one of its end vertices, see Figure 3.1. In order to achieve results under this restriction we concentrate on the planar setting, which is suitable for modeling terrains in $3D$. The effect of such a contraction is that one of its end vertices is removed. We call vertices that can be removed in this simple way *1*-step removable.

It would be preferable to reduce the size of the triangulation by identifying only 1-step removable vertices. For this purpose, we show that in the planar setting there is always at least one 1-step removable vertex inside every non-empty cycle of length smaller than six, see Theorem 3.11. This ensures us that if the outer frame consists of less than six vertices, then one can proceed reducing repeatedly the number of vertices in the interior by such simple 1-step removals. However, this result does not suffice to ensure a hierarchical graph of logarithmic size, nor does it guarantee that the user can specify a substantial amount of vertices (and/or edges) he wants to keep intact, while still being able to perform such simple edge contractions. To be able to guarantee both these desired options, we introduce a smooth alternative to 1-step reductions, namely 2-step removals. They result in a small modification of the mesh around the vertex to be removed. A 2-removal is geomorphic, i.e., it is visually smooth, it avoids degenerate intermediate

triangles of zero area, and it involves at most 2 straight movements. We call vertices which can be removed in this way 2-step removable. We also introduce the possibility that the user defines "important" vertices (or edges) which have to remain intact. Given m such important points we show that a hierarchical graph of size $O(n)$ and depth $O(\log(n/m))$ can be achieved using 2-step removals in linear time, while maintaining the m important vertices. Furthermore, some vertices are not even 2-step removable, and thus, we also define and study k-step removable vertices. These are also visually smooth operations, like 2-step removals, with the only difference that they may involve up to k straight movements. The option of performing k-step removals, for some constant $k > 2$ increases substantially the lower bound on the portion of removable vertices. We show that when removing a vertex v with degree q, then k is bounded from above by either $q - 4$, or by the number of concave (reflex, i.e. $> \pi$) corners on the link of v (see Section 3.2 for a definition), and from below by $(\lfloor q/3 \rfloor - 1)$.

In Section 3.1 we define the k-step remove operation, and in Section 3.2 we prove two upper bounds related to k through two different approaches, as well as a lower bound. Further, in Section 3.3 we derive lower bounds for the number of k-step removable vertices. In Section 3.4 we give additional arguments for the usefulness of the 2-step removals, by showing that they increase in number as the number of 1-step removable vertices decreases. Their exact relationship is determined. In Section 3.5 we show that a hierarchical graph of size $O(n)$ and depth $O(\log(n/m))$ can be achieved using 2-step removals in linear time, while maintaining m important vertices. Finally, we give a brief overview of future research in Section 3.7.

For simplicity, in the text we usually assume, unless mentioned otherwise, that we have no three collinear points, and hence we do not have to consider angles with degree 180. Our results also hold if we allow collinear points and in this case treat angles of 180 degrees as concave.

3.1 Basic definitions

In this section we define some basic operations and notations. As input we are given a triangulation $T = (V, E)$ with a simple polygon P as boundary, and a set $U \subseteq V$ of m important vertices, where V is the set of n vertices of the triangulation and E is the set of edges, see Fig. 3.2. We let P_e denote the edges of P, and we let P_v denote the vertices of P, where $P_v \subseteq U$, meaning that neither P_e or P_v will be removed during simplifications. We call the vertices in P_v *exterior* vertices and the rest *interior* vertices. Next,

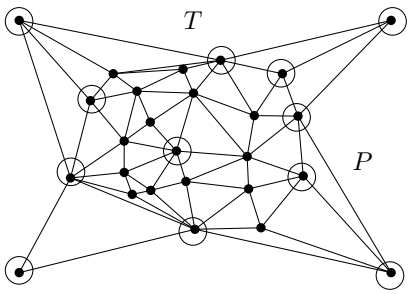

Figure 3.2: A planar triangulation T, with a simple polygon P as boundary, is given as input, as well as m important vertices (circled) that may not be removed.

let $N_T(v) := \{w \in V | (v, w) \in E\}$ denote the open neighborhood of a vertex $v \in V$. A *restricted edge contraction*, see Fig. 3.1, of an edge $e = (u, v) \in E$ *on (or onto)* v, is an operation that removes u, v and for each edge (u, w), if $w \neq v$, and $(v, w) \notin E$, then it replaces (u, w) by (v, w) in T, and if $w \neq v$, and $(v, w) \in E$, then (u, w) is removed from T. The inverse operation where vertex v is split into two vertices $u \in V$ and $v \in V$ is called a *vertex split*, see Fig. 3.1 In this chapter we only consider such restricted edge contractions. Furthermore, for the remainder of this chapter we say edge contraction as short for restricted edge contraction. In a graphics related context, an edge contraction can be shown smoothly, by continuously displaying a straight motion of u to v, while the edges adjacent to u are maintained as straight connections to u. Moreover, the aim is to simplify T by iteratively performing edge contractions.

A problem that often occurs during edge contractions of triangulations is that the resulting graph might not be a planar triangulation. An edge contraction is said to be *valid* if the resulting graph is still a planar triangulation (see Fig. 3.3a), and *invalid* otherwise (see Fig. 3.3b).

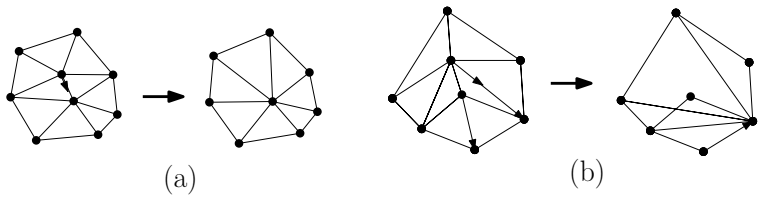

Figure 3.3: (a) A valid edge contraction. (b) An invalid edge contraction.

Definition 3.1 Given an interior vertex $v \in V$, the link of v, denoted $link(v)$, is the cycle of T passing through the neighbors of v, where the edges of the cycle form the boundary of the union of the triangles incident to v. Furthermore, let $I(v)$ denote the closed region bounded by $link(v)$. We say that two vertices $u, u' \in link(v) \cup \{v\}$ *see* each other if the straight-line segment between u and u' lies entirely within $I(v)$

In this chapter we consider a generalized edge contraction. For this purpose we first define a *split-and-contract* operation.

Definition 3.2 Given a vertex $v \in V$ and vertices $s, t, u \in link(v)$, let $C(s, t, u)$ be the vertices (s and t included) of the chain of $link(v)$ which connects s and t, and includes u. A split-and-contract from v to u, using s and t (illustrated in Fig. 3.4a-c) denotes an operation where v is split into two vertices, v and v_1, such that v_1 is connected to $C(s, t, u) \cup \{v\}$, and v is connected to $\{N_T(v) \setminus C(s, t, u)\} \cup \{s, t, v_1\}$. After this split the edge (v_1, u) is contracted on u. The split-and-contract operation is said to be *valid* if the triangulation is planar at every step of the operation.

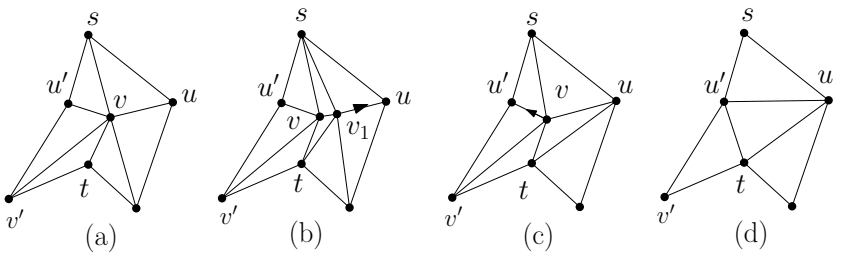

Figure 3.4: Illustrating a 2-step contraction of a degree 6 node v.

Note that a split-and-contract does not reduce the size of T. However, when an edge is contracted, two vertices are replaced by one. Thus, we define the concept of a 1-*step removable* vertex and generalize this concept into a k-*step removable* vertex.

Definition 3.3 If there exists an edge $e = (u', v) \in E$ such that e is validly contractible on v, then we say that u' is *1-step removable on (or onto)* v. Such an operation is called a 1-*step removal*.

Definition 3.4 A vertex $v \in V$ is said to be k-step removable if there exist vertices $s, t, u \in link(v)$ such that a valid split-and-contract from v to u,

using s and t can be made, and after this split-and-contract, v is $(k-1)$-step removable. Such an operation is called a *k-step removal*.

It is clear that a k-step removable vertex is also $(k+1)$-step removable. Figures 3.4a-d show a vertex v that is 2-step removable since a valid split-and-contract from v to u, using s and t is followed by a 1-step removal of v on u'.

3.2 Characterization of k-step removals

In this section we consider k-step removals, and show bounds (Lemma 3.6, Theorems 3.8 and 3.13) related to k and the removability of a vertex v. We start with the lower bounds.

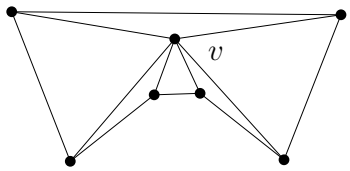

Figure 3.5: Example of a vertex v that is not 1-step removable.

Lemma 3.5 *In order to remove an interior vertex $v \in V$ with degree at least six a 2-step removal may be required.*

Proof: Figure 3.5 illustrates an example where no vertex on $link(v)$ can see all other vertices in $link(v)$, hence the lemma follows. □

For a more general lower bound we consider a vertex v as shown in Fig. 3.6a. The vertex is such that $link(v)$ consists of three concave chains C_1^0, C_2^0 and C_3^0, each such that the number of vertices differs by at most one.

Lemma 3.6 *In order to remove a vertex v of degree at least $q \geq 9$, a $(\lfloor q/3 \rfloor - 1)$-step removal may be required.*

Proof: Assume that a series of at most $(\lfloor q/3 \rfloor - 2)$ valid split-and-contracts are performed on v. This results in a series $l_0, \ldots, l_k$, $k \leq \lfloor q/3 \rfloor - 2$, where l_i denotes $link(v)$ after i split-and-contracts. We claim that for each l_i, $i \leq k$, it holds that it consists of three concave chains, C_1^i, C_2^i and C_3^i, connected either by an edge or a common convex corner, where each such concave

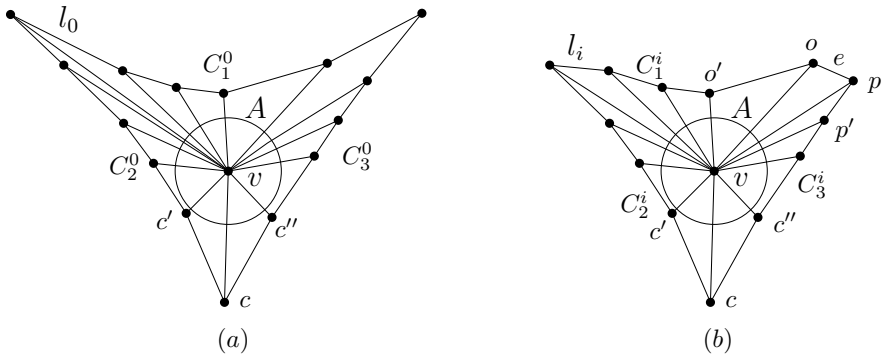

Figure 3.6: (a) The structure used to prove the lower bound. (b) The structure after i split-and-contracts, for $i = 2$.

chain consists of at least $\lfloor q/3 \rfloor - (i+1)$ concave vertices, see Fig. 3.6b. We show this claim by a structural induction.

First consider the base case, l_0. The claim holds immediately from the construction. For the induction hypothesis we assume that the claim holds for l_{i-1}. Finally, we consider l_i. In order to perform a valid split-and-contract, three consecutive vertices on $link(v)$ are required which all see each other. In l_{i-1} we have two types of consecutive vertices, where they all see each other. Either a convex vertex $c \in link(v)$ and its two neighbors $c' \in link(v)$ and $c'' \in link(v)$ all see each other, or the two endpoints of an edge $e = (o, p)$ connecting two concave chains, where $o, p \in link(v)$ and their two neighbors $o' \in link(v)$ and $p' \in link(v)$, all see each other.

Using any of these vertices in a valid split-and-contract results in that the number of concave vertices on any of the concave chains C_j^{i-1}, $1 \leq j \leq 3$, is reduced by at most one. Thus, the number of concave vertices on each concave chain C_j^i will be at least $\lfloor q/3 \rfloor - (i+1)$. Further, these concave chains will be pairwise connected by either an edge or a common vertex. Thus, the claim follows.

Further, as long as each concave chains C_j^i has at least two corners, no 1-step removal of v is possible. From the above claim it is clear that at least $\lfloor q/3 \rfloor - 2$ valid split-and-contracts have to be performed before any concave chain C_j^i can have at most one concave corner. This holds since $\lfloor q/3 \rfloor - (\lfloor q/3 \rfloor - 2 + 1) = 1$. The lemma follows. □

With regards to guaranteeing a hierarchical simplification graph of small size and depth, we mainly consider 2-step removals (Section 3.5). However,

depending on complexity, some areas of an object may require more triangles than others, such as, for example, the nose of a face, versus the more flat cheek. Thus, it would be desirable to be able to choose local areas in which to remove vertices. However, as Lemma 3.6 shows, 2-step removals may not always be sufficient for a specific area. Consider, for example, the area marked as A in Fig. 3.6. This area contains only one vertex v of degree $q \geq 12$ (a $\lfloor q/3 \rfloor - 1$ - step removal may be required). Thus, the generalized k-step removal is needed.

Next we focus on finding an upper bound. Assume without loss of generality that v has c concave vertices on $link(v)$, as shown in Fig. 3.11a. Let s_1 be a concave vertex farthest from v and order the concave vertices $s_1, \ldots s_c$ as they appear clockwise around v (in this context, let $i + 1 = 1$ if $i = c$, and let $i - 1 = c$ if $i = 1$). Next, let β_i denote the angle $\angle s_{i-1} s_i s_{i+1}$ and let α_i denote the angle $\angle s_i v s_{i+1}$. Moreover, let $C(s_i)$ denote the subchain of $link(v)$ clockwise from s_i to s_{i+1} and let $C_P(s_i)$ denote the convex polygon bounded by $C(s_i)$ and the edge (s_i, s_{i+1}). The following lemma will be needed:

Lemma 3.7 *If $\alpha_i \leq 180^\circ$ then the two consecutive concave vertices s_i and s_{i+1} must see each other.*

Proof: Since s_i and s_{i+1} are consecutive concave vertices of $link(v)$, the chain $C(s_i)$ and the vertex v must lie on opposite sides of the line $s_i s_{i+1}$, and the lemma follows. □

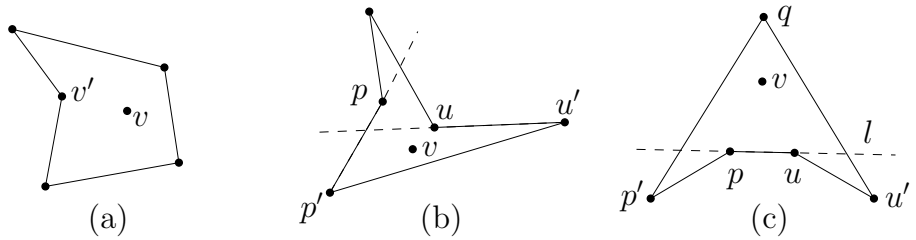

Figure 3.7: (a) A degree five vertex v with only one concave vertex in $link(v)$. (b) and (c) illustrates the two cases occurring in the proof of Theorem 3.8. Edges between v and $link(v)$ are not included in order to avoid cluttering of the figure.

Theorem 3.8 *Every interior vertex v with degree at most k is q-step removable, where $q = \max\{1, k - 4\}$.*

Proof: The theorem is proven by induction on the degree of v.

Base case: Vertices of degree at most four can easily be shown to be 1-step removable. We thus assume that v has degree five, as shown in Fig. 3.7. Consider $I(v)$. We say that a vertex q is concave/convex, if the angle of the two edges incident on q is concave/convex. If there exists a vertex v' of $link(v)$ which can see all other vertices of $link(v)$ then v is 1-step removable on v', and the theorem holds. Below we prove that $link(v)$ will always contain at least one vertex that can see all the other vertices of $link(v)$.

If $link(v)$ is a convex polygon then every vertex can see all the other vertices. If $link(v)$ has one concave vertex v' then v' can see all other vertices of $link(v)$, see Fig. 3.7a. Otherwise, $link(v)$ has two concave vertices (it cannot have three), and we have two cases as illustrated in Fig. 3.7b and 3.7c, respectively. In the first case the concave vertices p and u are not adjacent, while in the second they are.

First case: Note that p and u must lie inside the triangle defined by the three vertices of the convex vertices in $link(v)$, and that p and u always see each other. There exist two adjacent convex vertices p' and u', such that p' is adjacent to p and u' is adjacent to u. It is straightforward to see that p sees all vertices of $link(v)$ if the edge (p, p') does not intersect the extension ray of the edge (u, u'), as p then can see u'. The same holds for u, the edge (u, u') and the line extension of (p, p'). However, both cases cannot occur simultaneously as this implies that the edges (u, u') and (p, p') must cross. Thus, either p or u can see all of $link(v)$.

Second case: Consider the one convex vertex q not adjacent to either p or u. Let p', u' be the other two vertices such that p' and u' are adjacent to p and u, respectively. Since q is adjacent to both p' and u', q will see all vertices of $link(v)$ if and only if q sees both p and u. Next, consider the line l through p and u. Since p and u are concave vertices, p' and u' must lie on the same side of l. Furthermore, q must connect to p' and u' such that p' and u' are convex and p and u are concave. This means that q must lie on the opposite side of l with respect to p' and u', which immediately implies that q can see both p and u.

Thus, v is 1-step removable in both subcases and the base case holds.

Induction hypothesis: Assume that the theorem holds for all vertices of degree at most $m - 1$.

Induction step: Assume that v has degree m. We show that there always exists a valid split-and-contract, where the result of a split-and-contract is that one edge incident to v is flipped.

If $link(v)$ contains at most one concave vertex then v is 1-step removable as there exists a vertex in $link(v)$ that sees all other vertices. Thus, we may

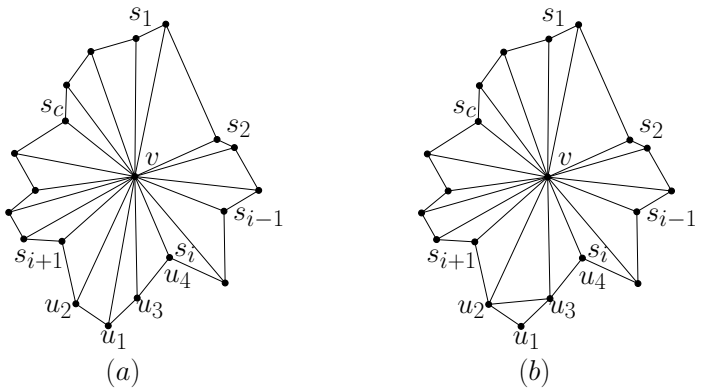

Figure 3.8: (a) A split-and-contract from v to u_3, using u_2 and u_4 is valid since u_3 sees u_1, u_2 and u_4 (b) The split-and-contract results in the edge (v, u) being flipped.

assume that $c \geq 2$. Since v sees all of $link(v)$, there exists at least one s_i such that $v \notin C_P(s_i) \setminus (s_i, s_{i+1})$. Consider a vertex $u_1 \in C(s_i) \setminus \{s_i, s_{i+1}\}$ and its two neighbors $u_2, u_3 \in C(s_i)$, as well as the neighbor $u_4 \in link(v)$ of u_3 as shown in Fig. 3.8a. Since v sees all of $link(v)$ we have that $u_3 \in C(s_i)$ must see u_1, u_2 and u_4. Thus, we can perform a split-and-contract from v on u_3, using u_2 and u_4, resulting in the edge (v, u) being flipped to (u_2, u_3), see Fig. 3.8b. Thus, the degree of v is now $m - 1$, and applying the induction hypothesis on v proves the theorem. □

Next we show a theorem that follows from Theorem 3.8. First we will need two simple observations. Recall that P_v is the set of vertices on the boundary of T.

Observation 3.9 *If $|P_v| = 3$, and the number of interior vertices is at least one, then each vertex in P_v must have degree at least three, see Fig. 3.9a.*

Observation 3.10 *Any triangulation of P has $(2n-2-|P_v|)$ triangles and $(3n - 3 - |P_v|)$ edges.*

This last observation follows from the proof of Theorem 9.1 in the book by de Berg et al. [46], which immediately implies that the total degree of T is $(6n - 6 - 2|P_v|)$.

Theorem 3.11 *If the simple polygon P, on the boundary of the triangulation T contains at most five vertices, the number of vertices of T is strictly*

greater than the number of vertices of P, and all vertices not in P are non-important, then there exists at least one 1-step removable vertex in T.

Proof: We have three cases:

Case 1: $|P_v| = 3$. From Observation 3.9 we know that all vertices in P_v must have degree at least three, thus at least nine in total. We know that the total degree of all interior vertices is at most $6n-6-2|P_v|-9 = 6n-21$. Next, from Theorem 3.8 we know that a vertex of degree at most five is 1-step contractible. Since no vertex in P_v is removed, this means that in the worst case all vertices in P_v have degree exactly three, while a maximum number of interior vertices have degree exactly six. However, not all interior vertices can have degree exactly six, since the total degree of all interior vertices would then be $6(n-3) = 6n-18$, which is a contradiction. Thus, there exists at least one interior vertex v of degree at most five, which means that v is 1-step removable.

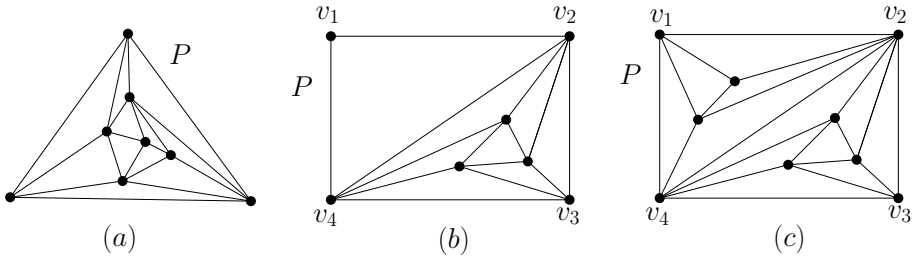

Figure 3.9: (a) If $|P_v| = 3$ then all vertices P_v must have degree at least three. (b) First subcase for $|P_v| = 4$, where vertex v_1 has degree two. (c) The second subcase for $|P_v| = 4$, where all vertices in P_v have degree at least three.

Case 2: $|P_v| = 4$. Let $v_1, \ldots, v_4$ be the vertices in P_v in clockwise order. We have two subcases. The first is that one of the vertices in P_v has degree two, see Fig. 3.9b. If v_1 has degree two then both of its neighbor endpoints $v_2, v_4 \in CH(T')$ must be connected with an edge in order to form a triangle. Further, this triangle must be empty, thus, all interior vertices of T must lie in the triangle v_2, v_3, v_4. This triangle must contain at least one 1-step removable vertex as shown in Case 1 above. The second subcase is that no vertex in P_v has degree two, see Fig. 3.9c. In this case the total degree of all vertices on P_v must be at least twelve. Thus, the total degree of all interior vertices is at most $6n-6-2|P_v|-12 = 6n-26$.

Not all interior vertices can have degree exactly six, since the total degree of all interior vertices would then be $6(n-4) = 6n-24 > 6n-26$, which

is a contradiction. Thus, there must exist at least one interior vertex v of degree at most five, which means that v is 1-step removable.

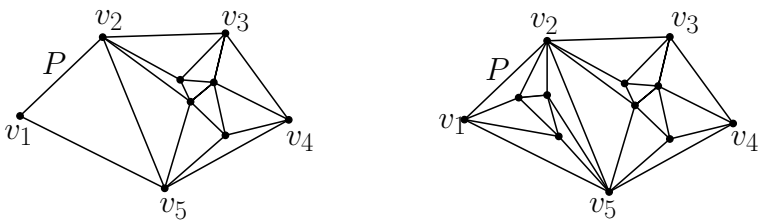

Figure 3.10: (a) First subcase for $|P_v| = 5$, where vertex v_1 has degree two. (b) The second subcase for $|P_v| = 5$, where all vertices in P_v have degree at least three.

Case 3: $|P_v| = 5$. Let $v_1, \ldots, v_5$ be the five vertices of P_v in clockwise order. We have two subcases. If one of the vertices, say $v_1 \in P_v$, has degree two then both of its neighbor endpoints $v_2, v_5 \in P_v$ must be connected with an edge in order to form a triangle. Further, this triangle must be empty, and thus, all interior vertices of T must lie in the quadrangle $v_2, v_3, v_4, v_5 \in P_v$. This quadrangle must contain at least one 1-step removable vertex as shown in Case 2 above. The second subcase is that no vertex in P_v has degree two, see Fig. 3.10b. In this case the total degree of all vertices on P_v must be at least fifteen. From Observation 3.10 we know that the total degree of T in this case is $6n - 6 - 2|P_v| = 6n - 16$, which means that the total degree of all interior vertices is at most $6n - 16 - 15 = 6n - 31$.

Finally, not all interior vertices can have degree exactly six, since the total degree of all interior vertices would then be $6(n-5) = 6n-30 > 6n-31$, which is a contradiction. Thus, there must exist at least one interior vertex v of degree at most five, which means that v is 1-step removable. □

From this theorem it immediately follows that if a region bounded by a non-empty cycle of length smaller than six in T contains no important vertices, then there exists at least one 1-step removable vertex in T. Further, the above Theorem is in a sense tight since Lemma 3.5 tells us that if P_v contains at least six vertices, it may be that no interior vertex of T is 1-step removable. One should also note that if only a few vertices are 1-step removable then almost all interior vertices in T must have degree 6, while simultaneously being not 1-step removable. However, we have not been able to construct any such examples, thus the bound stated in Lemma 3.11 might be very conservative.

Next, we present an upper bound on k, such that every vertex in T is k-step removable. As only concave corners restrict visibility, intuitively it should be easier to remove a vertex with few concave corners on its link. A valid split-and-contract which reduces the number of concave corners by at least one in the resulting $link(v)$ is denoted a *reducing* split-and-contract.

Lemma 3.12 *If $\beta_i < 180°$, $v \notin C_P(s_{i-1})$ and $v \notin C_P(s_i)$, then a split-and-contract from v to s_i, using s_{i-1} and s_{i+1}, is reducing.*

Proof: Since $v \notin C_P(s_{i-1})$ and $v \notin C_P(s_i)$ it immediately follows that $\alpha_{i-1} < 180°$ and $\alpha_i < 180°$, which means (Lemma 3.7) that s_{i-1} and s_i see each other, as do s_i and s_{i+1}. This makes the split-and-contract valid since all the vertices of $C(s_{i-1})$ and $C(s_i)$ see s_i, to which they will be connected after the split-and-contract. Further, since $\beta_i < 180°$, s_i will be convex after the split-and-contract and no new concave corners can appear on the resulting $link(v)$, and the lemma follows. □

The following theorem can now be shown (see Fig. 3.11 for an illustration). Recall that s_1 is the concave vertex farthest from v and that the concave vertices are ordered $s_1, \ldots s_c$ as they appear clockwise around v.

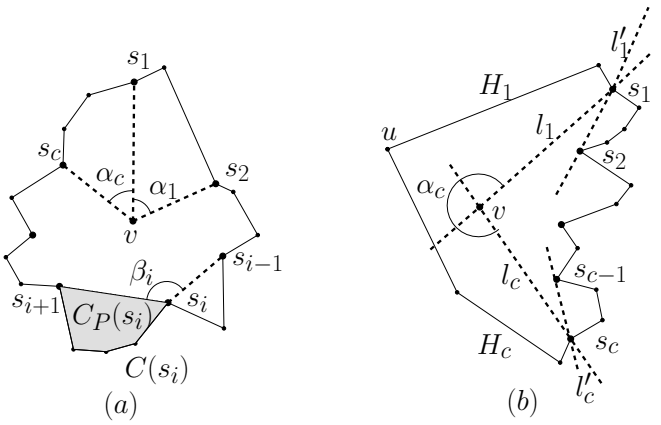

Figure 3.11: An illustration of the two cases of Theorem 3.13. Figure (a) illustrates Case 1, and figure (b) illustrates Case 2.

Theorem 3.13 *Every interior vertex $v \in V$ with at most c concave vertices on its link is c-step removable.*

Proof: The theorem is proven by induction on c.

Base cases: If $c = 1$ then let s be the concave vertex on $link(v)$. As s can see all other vertices on $link(v)$, v can be 1-step removable on s, thus the theorem holds for $c = 1$.

Induction hypothesis: Assume that the theorem holds when $link(v)$ contains at most $c - 1$ concave vertices.

Induction step: Assume $link(v)$ contains c concave vertices. We show that there always exists a reducing split-and-contract, and thus, the theorem is proved by applying the induction hypothesis. Consider the following cases:

Case 1: $\alpha_c < 180°$ and $\alpha_1 < 180°$. In this case $c \geq 3$ otherwise either $\alpha_1 \geq 180°$ or $\alpha_c \geq 180°$. We also immediately have that $v \notin C_P(s_c)$ and $v \notin C_P(s_1)$ since both α_c and α_1 have degree strictly smaller than 180°. Furthermore, since s_1 is a vertex in $link(v)$ furthest from v, it holds that $\beta_1 < 180°$. Thus, a split-and-contract from v to s_1, using s_c and s_2, is reducing according to Lemma 3.12.

Case 2: $\alpha_c \geq 180°$ or $\alpha_1 \geq 180°$. Assume without loss of generality that $\alpha_c \geq 180°$ which immediately implies that $v \in C_P(s_c)$, see Fig. 3.11b. Since s_1 and s_c both are visible from v there can be no other vertex s_i, $i \neq c$, such that $v \in C_P(s_i)$ (except for the case $c = 2$, where we have that $v \in C_P(s_1)$ and $v \in C_P(s_c)$ when $\alpha_1 = \alpha_c = 180°$. However, this does not change the validity of the proof below). Let l_1 and l'_1 be the lines through s_1 and v, and through s_1 and s_2, respectively. Let l_c and l'_c be the lines through s_c and v, and through s_c and s_{c-1}, respectively. Line l_1 defines two halfplanes, where we consider the halfplane containing none of the vertices $s_2, s_3, \ldots s_c$ (s_1 is on the border of this halfplane). Also, l'_1 defines two halfplanes, where we consider the one not containing $C_P(s_1)$. Let H_1 be the intersection of these two halfplanes, as shown in Fig. 3.11b. H_c is defined correspondingly using lines l_c and l'_c. Assume that H_1 contains a vertex $u \in C(s_c)$, and consider a split-and-contract from v to s_1, using u and s_2. All vertices between u and s_1 are visible to s_1, since $H_1 \cup H_c$ can contain vertices from $C(s_c)$ only. The same holds between s_1 and s_2 since $v \notin C_P(s_1)$. Moreover, the definition of l_1 guarantees that v will remain in the resulting $link(v)$ after the split-and-contract, and line l'_1 guarantees that the corner defined by edges (u, s_1) and (s_1, s_2) is convex. Thus, if H_1 contains a vertex $u \in C(s_c)$, the above split-and-contract must be reducing. Correspondingly, a split-and-contract from v to s_c, using u and s_{c-1} will be reducing if H_c contains a vertex $u \in C(s_c)$. Finally, the area $H_1 \cup H_c$ must contain a vertex from $C(s_c)$ since $\alpha_c > 180$ and $C(s_c)$ connects s_c and s_1. □

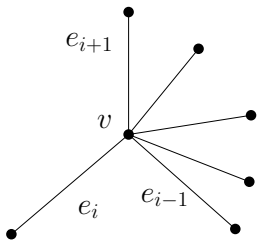

Figure 3.12: Illustrating Lemma 3.15

3.3 The number of k-step removable vertices

In this section we show two linear bounds on the number of k-step removable vertices. These bounds are compared, and further examined to see if they may be combined for even better bounds, something that is answered in the negative. Throughout the whole section we assume that $|P_v| = 4$ and that $m = 0$. The results for this restricted case will then be used in Section 3.5 to achieve more general results.

The first bound follows from Theorem 3.8 and the fact that the total degree is bounded.

Lemma 3.14 *If $|P_v| = 4$ and $m = 0$ then at least $(\frac{k-1}{k+2})(n-4)$ interior vertices are k-step removable, for every $k \geq 2$.*

Proof: Let L be the set of interior vertices of T that are k-step removable. From Theorem 3.8 we know that L contains all vertices of degree at most $d = k + 4$. Let N be the remaining set of interior vertices. Recall that the sum of the degrees over all vertices is exactly $6n - 6 - 2|P_v| = 6n - 14$. We also have that the total degree of P_v must be at least 12. This holds since a vertex $v \in P_v$ with degree two implies that its two neighbors $v', v'' \in P_v$ must have degree at least four in order to guarantee a planar triangulation. Moreover, as the vertices in N have degree at least $d+1$ and the vertices in L have degree at least three, we have the following equation:

$$12 + (d+1)|N| + 3|L| \leq 6n - 14 \text{ where } |N| + |L| = n - 4.$$

As a result it holds that $|L| \geq \big(\frac{d-5}{d-2}\big)(n-4) \geq \big(\frac{k-1}{k+2}\big)(n-4)$. □

3.3.1 Deriving a lower bound without using sums of degrees

Next we examine whether the above bound can be improved by using Theorem 3.13 instead. To see whether an improvement is possible, we addition-

ally assume that the points are in general position (this is needed in order to use lemma 3.15 below). Despite of this, we will see that no significant improvement is obtained by following this approach. In order to use Theorem 3.13 we need to consider the total number of concave corners on $link(v)$, for every vertex $v \in V$. Consider any vertex v, and order its adjacent edges $e_1, \ldots, e_k$ clockwise around v, see Fig. 3.12. Let θ_i denote the clockwise angle $\angle e_{i-1}e_{i+1}$. Furthermore, we say that a vertex *generates* a concave corner if θ_i is strictly greater than 180°, for any i. Thus, this generated corner will appear as a concave corner on $link(u)$ for the neighbor u of v, where both u and v are endpoints of e_i.

Lemma 3.15 *A vertex v generates at most three concave corners.*

Proof: Consider an edge e_i such that θ_i is concave. Since $\angle e_{i+1}e_{i-1}$ is convex, it follows that there can be no edge e_j between e_{i+1} and e_{i-1} such that θ_j is concave. Thus, in addition to θ_i, only θ_{i+1} and θ_{i-1} may be concave. □

A vertex may generate three concave corners. As an example, consider Fig. 3.12, but with edges e_{i-1}, e_i and e_{i+1} only.

Again, we assume that $|P_v| = 4$ and $m = 0$. Let L be the set of interior vertices of T that are k-step removable, and let N be the remaining set of interior vertices. From Lemma 3.15 we know that each vertex may generate at most three concave corners, that is, $3n$ in total. Moreover, since no vertex in N is k-step removable we know from Theorem 3.13 that each of them has at least $k+1$ concave corners on their link. Thus, we get

$$3n \geq (k+1) \cdot |N| \qquad \text{and} \qquad |N| + |L| = n - 4.$$

This means that $L \geq \left(\frac{k-2}{k+1}\right)n - 4$. Comparing this bound with Lemma 3.14, we see that Lemma 3.14 is still somewhat stronger. However, if the number of k-step removable (due to degree) vertices decreases, it is clear that the possible total number of concave corners must also decrease as only vertices of degree three may generate three concave corners. Thus, one may ask if the concave corner approach may improve the above bound from Lemma 3.14, as the number of k-step removable (due to degree) vertices approaches the lower bound.

Let $f(k)$ denote a lower bound on the number of vertices that are k-step removable. Assume that $f(k) = (\frac{k-1}{k+2})(n-4)$. Since all k-step removable vertices may have degree three, they all generate three concave corners in the worst case. The four exterior vertices may have degree three, but since

they all have two neighboring exterior vertices they may generate at most two concave corners such that these two corners appear on $link(v)$ of an internal vertex v. This means that we have

$$3 \cdot (\frac{k-1}{k+2})(n-4) + 2 \cdot (n - (\frac{k-1}{k+2})(n-4)) = 3n(\frac{k+1}{k+2}) - 8\frac{k-1}{k+2}$$

concave corners in total, in the worst case. Thus, at most

$$\left(3n(\frac{k+1}{k+2}) - 8\frac{k-1}{k+2}\right)/(k+1) = 3n(\frac{1}{k+2}) - 8\frac{k-1}{(k+2)(k+1)}$$

vertices are not k-step removable, and at least

$$n - 4 - 3n(\frac{1}{k+2}) + 8\frac{k-1}{(k+2)(k+1)} = (\frac{k-1}{k+2})n - 4 + 8\frac{k-1}{(k+2)(k+1)}$$

are k-step removable. Which is the same lower bound shown using only the degree approach (within a small additional factor).

We may also want to consider if the lower bound may be refined using Theorem 3.13. Using only Theorem 3.8 it is clear that we cannot guarantee that nodes of degree greater than $k+4$ are k-step removable. However, it is possible that k-step removal for these nodes could be guaranteed using Theorem 3.13, if they were not able to consume at least $k+1$ concave corners each. First, assume that $f(k) = (\frac{k-1}{k+2})(n-4)$, we may have at most

$$3n(\frac{k+1}{k+2}) - 8\frac{k-1}{k+2}$$

concave corners as seen above. Further, the number of nodes of degree greater than $k+4$ is less than

$$n - (\frac{k-1}{k+2})(n-4) = (\frac{3}{k+2})n + 4k - 4.$$

We also see that

$$\left((\frac{3n(k+1)}{k+2}) - \frac{8(k-1)}{k+2}\right) / \left((\frac{3n}{k+2}) - \frac{8(k-1)}{(k+2)(k+1)}\right) = k+1$$

which means that at most $(\frac{3n}{k+2}) - \frac{8(k-1)}{(k+2)(k+1)}$ vertices of degree at least $k+4$ may consume at least $k+1$ concave corners. Thus at least

$$(\frac{3n}{k+2}) + 4k - 4 - (\frac{3n}{k+2}) - \frac{8(k-1)}{(k+2)(k+1)} = \frac{4k^3 + 8k^2 + 4k + 16}{(k+2)(k+1)}$$

additional vertices will also be k-step removable. However, as the number of 2-step removable vertices increases, the number of generated corners may also increase. This means that above additional factor would have to be further examined to see whether or not it could be added to the lower bound of Lemma 3.14. We do not perform this examination, as it only involves a small additional factor which would make forthcoming calculations increasingly complex without significant gain. We can, however, show the following lemma, where 'at least' has been replaced in Lemma 3.14, by 'strictly more'.

Lemma 3.16 *Assuming that the input points are in general position, that* $|P_v| = 4$ *and* $m = 0$*, then strictly more than* $(\frac{k-1}{k+2})(n-4)$ *interior vertices are* k*-step removable, for every* $k \geq 2$*.*

3.4 The relationship between 1-step and 2-step removable vertices

In this section we show relations between the number of 1-step removable vertices and the number of 2-step removable vertices, assuming that no interior vertex is important. It follows from these relations that the lower bound on the number of 2-step removable vertices increases as the number of 1-step removable vertices decreases. A motivation for studying these bounds is that although it would be desirable to have a larger number of 1-step removable vertices, we show that a lack of such vertices is to a certain degree compensated by a guarantee that there is a larger portion of 2-step removable vertices. This serves also as a further motivation for considering 2-step removals.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ denote the set of 1-step and 2-step removable interior vertices, respectively. We say that a vertex v *consumes* x concave corners if $link(v)$ contains x concave corners. Moreover, for simplicity we assume that the total degree over all vertices is $6n$. This is done in order to simplify the calculations. Note, however, that increasing the total degree decreases $|\mathcal{T}_2|$ as the number of vertices of degree at least seven increases. Thus, the lower bounds are still valid.

First we show the following:

Lemma 3.17 *If* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *are the sets of* 1*-step and* 2*-step removable interior vertices, respectively, of the input triangulation, and* P_v *is the set of vertices on the boundary of the input triangulation, then*

$$|\mathcal{T}_1| \leq x(n - |P_v|) \Rightarrow |\mathcal{T}_2| \geq \frac{n}{3}(1-x) - 3|P_v|, 0 \leq x \leq 1$$

Proof: Only vertices of degree three can generate three concave corners, and from Theorem 3.8 we know that only 1-step removable vertices can have degree three. Also, in the worst case all vertices in P_v have degree three. Thus we have at most

$$3(x(n-|P_v|)+|P_v|)+2(n-(x(n-|P_v|)+|P_v|)) \leq n(2+x)+5|P_v|$$

concave corners which means that at most $\frac{n(2+x)+5|P_v|}{3} \leq \frac{n(2+x)}{3}+2|P_v|$ vertices can consume at least three concave corners, thus not being 2-step removable (Theorem 3.13) . In the worst case all these vertices are interior, and as a result at least

$$n-|P_v|-\frac{n(2+x)}{3}-2|P_v| = \frac{n}{3}(1-x)-3|P_v|$$

are 2-step removable. □

The above bound can be slightly improved, as shown in the following Theorem:

Theorem 3.18 *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are the sets of 1-step and 2-step removable interior vertices, respectively, of the input triangulation, and P_v is the set of vertices on the boundary of the input triangulation, then*

$$|\mathcal{T}_1| \leq x(n-|P_v|) \Rightarrow |\mathcal{T}_2| \geq n(1-3x)-3|P_v|, \quad \text{for} \quad 0 \leq x \leq 1$$

Proof: Since the 1-step removable vertices must have degree at least three, and in the worst case all vertices in P_v have degree three, we have at most $3(x(n-|P_v|)+|P_v|)$ vertices of degree at least seven. This follows since a vertex a degree three 'allows' three vertices of degree at least seven. Finally, we have at least $n-3(x(n-|P_v|)+|P_v|) = n(1-3x)-3|P_v|$ vertices of degree six (which are 2-step removable). □

Both lemmas 'fail' for $x \geq 1/4$ as the lower bound of $\mathcal{T}_2$ becomes smaller than the upper bound of $\mathcal{T}_1$. Further, for $x \leq 1/4$ the lower bound for $\mathcal{T}_2$ is better for Lemma 3.18, making this lemma stronger than Lemma 3.17.

We may also consider whether Theorem 3.18 can be refined using Theorem 3.13. Such a refinement would be possible if not all vertices of degree at least seven were able to consume at least three concave corners. However, since

$$3 \cdot 3(x(n-|P_v|)+|P_v|) \leq n(2+x)+5|P_v|$$

for $x \leq \frac{2n}{8n-|P_v|}$, this does not hold (unless we are able to improve bounds, such as, for example, the total number of generated concave corners).

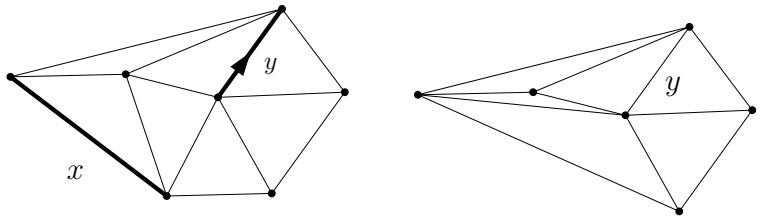

Figure 3.13: Initially edges x and y are contractible. After the contraction of x, y is no longer contractible.

3.5 The hierarchical graph

Next we create the *level-of-detail hierarchy*, L, of T where the general procedure is as follows. We start with T, which we denote the first *level*. A constant fraction of the vertices are then simultaneously removed, in a *round* of k-step removals, resulting in triangulation which we denote the second level. Correspondingly, level i of L is created through one round of k-step removals from level $i-1$ in L. Next, we represent L with a directed, acyclic graph H, denoted a *hierarchical graph*. A level i in L is a triangulation, while the corresponding level i in H is a set of nodes, where each triangle in level i of L is represented with a node in level i of H. Furthermore, one way to view a k-step removal of a vertex v is as removing v and all edges adjacent to v, leaving $I(v)$ empty, and then re-triangulating $I(v)$. Thus, we place a directed edge between a node u at level $i-1$ in H, and a node u' at level i of H, if both the triangle corresponding to u disappeared, and the triangle corresponding to u' appeared, as a result of the same k-step removal of a vertex v. We let U denote the set of nodes, and F denote the set of edges in H. Moreover, we say that the sum $|U| + |F|$ is the *size* of H, and that the number of levels of H is the *depth* of H.

In order to guarantee a hierarchical graph of small size and depth, several vertices must be simultaneously removed in each round. Ideally, these vertices would be 1-step removable. However, currently we are only able to guarantee at most one 1-step removable vertex, as seen in Lemma 3.11. Therefore we consider 2-step removable vertices. We show that using 2-step removals, given m *important* vertices or edges, we can achieve a hierarchical graph of size $O(n)$ and depth $O(\log(n/m))$, such that the simplified resulting triangulation contains the m important vertices and edges and at most $O(m)$ other vertices from T. Note that a previously valid contractible edge might become invalid after other edges have been contracted, as shown in

Fig. 3.13. In order to avoid this problem, for the purpose of finding simultaneously removable vertices, we consider non-adjacent vertices, that is *independent* vertices.

Theorem 3.19 *Given a triangulation $T = (V, E)$ with a simple polygon P as boundary, and m important (the vertices of P included) vertices $U \subset V$, one can perform $O(\log(n/m))$ rounds of 2-step removals to obtain a triangulation T' of a vertex set V' with complexity $O(m)$ such that $U \subseteq V' \subset V$. All rounds can be performed in $O(n)$ total time, and such that the corresponding hierarchical graph has size $O(n)$ and depth $O(\log(n/m))$.*

Proof: Assume that a rectangle R is added, see Fig. 3.14, such that T is included in R, and that edges are added such that the interior of R becomes triangulated. Denote this triangulation $Q = (S, D)$, where $n' = |S| = n + 4$, $m' = m + 4$, and the vertices of R are set as important.

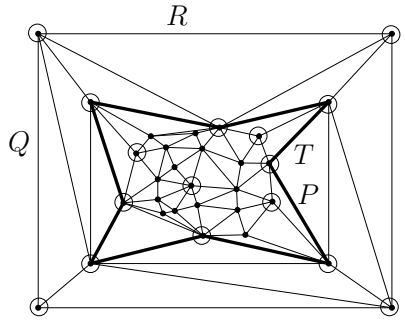

Figure 3.14: A rectangle R is added such that T is included in R. The vertices of R are set as important, and edges are added to the interior of R so that a triangulation Q is achieved.

Let S_2 be the set of 2-step removable interior vertices of Q of degree at most six, assuming that $m' = 0$. Lemma 3.14 was shown using Theorem 3.8, thus, we have that $|S_2| \geq \frac{n'-4}{4} \geq \frac{n'}{20}$, $n' \geq 5$. Since a vertex in S_2 has at most six neighbors we can choose at least $\frac{n'}{20 \cdot 7} = \frac{n'}{140}$ vertices from S_2 such that none of the chosen vertices has a neighbor from S_2. Thus, there exists a constant fraction $\gamma \geq \frac{1}{140}$ of independent 2-step removable vertices.

Let n'_i denote the number of vertices before round i and consider an arbitrary constant $\delta < \gamma$. Perform rounds on Q until $m' \geq \delta n'_i$, that is until the resulting vertex set S' has complexity $O(m')$. This is possible, since as long as $m' \leq \delta n'_i$, there are at least $\gamma n'_i - \delta n'_i = (\gamma - \delta) n'_i$ 2-step removable vertices remaining, containing no important vertex. Thus,

a triangulation Q' can be obtained using at most $O((\log_{\frac{1}{1-(\gamma-\delta)}} 2) \cdot (\log n' - \log m')) = O(\log(n'/m'))$ rounds of removals. Since $m = O(m')$ and $n = O(n')$ this immediately implies that one can perform $O(\log(n/m))$ rounds of 2-step removals on T to obtain a triangulation T' of a vertex set V' with complexity $O(m)$ such that $U \subseteq V' \subset V$. Hence, the construction of a hierarchical graph of depth $O(\log(n/m))$ is possible. Now, we estimate the size of hierarchical graph. The number of nodes in the hierarchical graph is, since $n_i \leq n\gamma^{i-1}$, at most $O(n + n\gamma + n\gamma^2 + \ldots + n\gamma^{O(\log(n/m))}) = O(n)$. Further, only 2-step removable vertices of degree at most six are used during the rounds of removals. This means that at most six triangles are removed and at most five triangles added as a result of the 2-step removal, which implies that each node in the hierarchical graph has at most five incident out-edges. Thus, in total the number of edges of the hierarchical graph is $O(n)$, which also means that the size is $O(n)$.

Finally we consider the time complexity of creating the hierarchical graph. Theorem 3.19 was shown using only 2-step removable vertices of constant degree (at most six). Thus, in each round i the set of γn_i independent 2-step removable vertices can be found in $O(n_i)$ time. Again, since $n_i \leq n\gamma^{i-1}$ the total running time is $O(n + n\gamma + n\gamma^2 + \ldots + n\gamma^{O(\log(n/m))}) = O(n)$. □

The above results also hold for m important edges (or m edges and vertices, in total), since each important edge restricts possible removals for only a constant (two) number of vertices.

3.6 Further research

Currently a number of possible extensions of the above results are being explored.

- To what extent do the results generalize to three dimensions?
- Is it possible to maintain bounded degree for all vertices during rounds of removals?
- Can the user specify to a greater degree which edges will be contained in certain levels of the hierarchical graph (maybe even including edges not in the original graph)?
- Can we obtain better upper and lower bounds for the number of 1-step removable vertices?

- Would implementing the algorithm and testing it on real data confirm an efficient performance and behavior?

3.7 Acknowledgement

We thank Tomas Akenine-Möller for valuable discussions and comments.

Chapter 4

Reporting Leaders and Followers Among Trajectories of Moving Point Objects

Movement is the spatio-temporal process par excellence. Technological advances of location-aware devices, surveillance systems and electronic transaction networks produce more and more opportunities to trace moving individuals. Consequently, an eclectic set of disciplines including geography [67], data base research [87], animal behavior research [92], surveillance and security analysis [130, 133, 154], transport analysis [98, 109], and market research [135] shows an increasing interest in movement patterns of various entities moving in various spaces over various times scales.

At the same time traditional geographic analysis suffers from the legacy of cartography's static perception of the world and is thus generally not suited for the analysis of individual movement trajectories [36, 138], sometimes referred to as geospatial lifelines especially in a GIScience context [122]. Many authors have therefore recently proposed to use geographical (and thus) spatio-temporal data mining as a promising alternative to overcome this methodological shortcoming [57, 125]. discusses variants of one specific movement pattern and algorithms reporting them efficiently.

As can be seen from the pattern terminology, this chapter is largely inspired by movement patterns observed in gregarious animals, such as flocking sheep or schooling fish. It follows a strategy to link the proposed patterns as close as possible to observable patterns. The proposed pattern definitions

are based on behavioral patterns discussed in the behavioral ecology literature and used for the modeling of realistic movement patterns of agent-based virtual life forms [93, 110].

This chapter addresses the movement pattern of one object leading others. We therefore define the movement pattern 'leadership' and subsequently presents algorithms to detect such patterns. Leadership, as defined in this chapter, bases on the geometrical relation of one individual moving in front of its followers. The algorithms presented for an efficient detection of 'leadership' make use of a set of auxiliary data structures, specifically developed for capturing those spatio-temporal relations amongst moving objects that constitute leadership.

Even though the leadership pattern in this chapter is motivated and investigated with respect to animal behavior research, its definition is held generic and is thus applicable to arbitrary types of entities moving in a 2D-space. In general the input is a set of n moving point objects $e_1, ..., e_n$ whose locations are known at τ consecutive time-steps $t_1, ..., t_\tau$, that is, the trajectory of each object is a polygonal chain that can self-intersect. For brevity, we will call moving point objects *entities* from now on. We assume that the movement of an entity from its position at a time-step t_j to its position at the next time-step t_{j+1} is described by the straight-line segment between the two coordinates, and that the entity moves along the segment with constant velocity.

This chapter is organized as follows. Section 4.1 links previous research on movement patterns similar to our leadership with the latest related research. Section 4.2 defines our notion of leadership and features definitions and preliminaries. In Section 4.3 and 4.4, we present algorithms for the detection of leadership. Then, in Section 4.6 we present experimental results and discuss their implications in Section 4.7. We conclude with final remarks and an outlook on future work in Section 4.8.

4.1 Related work

4.1.1 Inspiring Animals

Animals interact socially to gain from coordination of their behavior [38, 106]. Rands et al. [137] illustrated the spontaneous emergence of leaders and followers using a simulation model reproducing the decision process of a pair of foraging animals, balancing their energetic states. The idea and the term of leadership have been used in several different contexts in the field of animal behaviour research, see Dumont et al. [52] for an overview.

In general, one can distinguish two different readings:

1. (i) 'the event or process of one entity initiating a group movement (e.g. [28, 52, 115, 132])' Leading in this sense is an active behaviour, referring to individuals that consistently initiate displacement of the group they belong to. For example, Dumont et al. [52] found that in a group of 15 grazing heifers the same individual was reported to lead the group to new feeding places in 48% of all group movements. Similar leadership behaviour has also been studied in gray wolves (*Canis lupus*) [132].

2. (ii) 'the event or process of one entity in front, leading a group movement (e.g. [28, 85])' Leading in this sense involves the notion of a leader moving in front of followers. Gueron and Levin model the spatial constellation 'in front of' as a function of the relative position with respect to the averaged position of its neighbors within a given range. Even though it has been found for grazing animals that leaders may guide a group being in front or chasing from behind, animals in front are considered to be more relevant to determine where the group will graze [52].

The use of the geometrical arrangement of moving entities has furthermore a long tradition for realistically modeling group behaviour, be it in animal behaviour science [85] or in the animation industry [140]. Most prominent is the flocking model implemented in NetLogo [158, 163], which mimics the flocking of birds [162]. The moving agents dynamically coordinate their movement based on rules on alignment (turning in order to adopt direction of nearby agents), separation (turning to avoid getting to close to nearby agents) and cohesion (move towards other nearby agents). This model explicitly excludes the idea of an individual leading the others, but involves identical agents, each following the same set of rules. The basic model includes a maximal distance of vision r and 360 degree field of view. However, it is also possible in NetLogo to specify a cone of vision, a most interesting concept with respect to the investigation of further structure in flocking entities that can, for example, be seen in V-shaped flocks as with migrating geese. Such front priority is also often used for agent-based models of schooling fish, where only individuals in front are candidates as interacting neighbors [93]. Inada's and Kawachi's model uses a wide-angled cone of perception, directed in movement direction and thus omitting a blind region behind the fish (Figure 1, Figure 2 in [93]). Jadbabaie et al. give a theoretical explanation for the spontaneous coordination of agents despite

the absence of a centralized coordination and just following a simple nearest neighbor rule [95]. However, in an extension they also investigate the influence of a leader in their system.

All such research integrating biology with information science and computer science points out the potential of a systematic investigation of geometric relations of moving animals for analyzing, modeling and simulating movement processes. Above all, animal movement provides a set of very convincing metaphors for more generic movement patterns, as shall be exploited with the pattern 'leadership' in this chapter.

4.1.2 Limiting Databases

There is ample research on moving object databases (MOD) [89, 149, 164, 165]. Whereas most database research on MOD focuses on data structures, indexing and efficient querying techniques for moving objects [3, 63, 87, 88], only recently the potential of data mining for movement patterns has been acknowledged [104, 105, 141]. For example, Du Mouza and Rigaux propose mobility patterns that describe sequences of moves in a discrete 2D-space [50].

In a GIScience context, activity related movement patterns have been researched, often with respect to improving location-based services (LBS). Dykes and Mountain search episodes expressing distinctive characteristics of movement, including absolute speed, direction, sinuosity and measurements of their variations [56]. Smyth presents a data mining algorithm that assigns predefined activities to segments of trajectories by analyzing some measurable motion descriptors, such as speed, heading and acceleration [151].

A common approach in database research is to take an existing spatial query type and then study its generalizations to spatio-temporal data. An example of this is the recent work on continuous k-nearest neighbor querying over mobile data [128, 167]. The focus within data mining research is to design techniques to discover new patterns in large repositories of spatio-temporal data. For example, Mamoulis et al. [121] mine periodic patterns moving between objects and Ishikawa et al. [94] mine spatio-temporal patterns in the form of Markov transition probabilities. More recently Verhein and Chawla [160] used association rule mining for patterns such as, sinks, sources, stationary regions and thoroughfares.

Spatio-temporal proximity of entities is a reasonable first premise for many situations that assume interactions between individuals. One obvious analytical toolset to uncover proximity patterns in individual trajectories is clustering. Even though the spatio-temporal nature of movement data

adds additional complexity to clustering procedures, there have been some successful approaches for clustering trajectories [43, 119, 148]. However, spatio-temporal co-presence does not explicitly include the idea of interactions within individuals. Relations such as 'leading', 'following' or 'setting a trend' cannot be investigated by pure clustering alone.

In essence, conventional spatial and spatio-temporal querying and clustering are inherently static and thus limited in their ability to cope with dynamic movement. Hence complementing techniques have to be explored in order to cope with the emerging new generation of movement data. Shirabe [147] illustrates such an alternative and uses correlation analysis in order to discover leader and follower relationships amongst moving individuals.

4.1.3 Promising Patterns

Precursory to this research Laube and colleagues proposed the REMO framework (RElative MOtion) which defines similar behaviour in groups of entities [111, 112, 113]. They defined a collection of movement patterns based on similar movement properties such as speed, acceleration or movement direction. Laube et al. [114] extended the framework by not only including movement properties, but also location itself. They defined several movement patterns, including flock (coordinately moving close together), trendsetter (anticipating a move of others), leadership (spatially leading a move of others), convergence (converging towards a spot) and encounter (meeting at a spot) and gave algorithms to compute them efficiently. Later Gudmundsson et al. [83] considered the same problems and extended the algorithmic results by primarily focusing on approximation algorithms – 'Any exact values of m and r hardly have a special significance – 20 caribou meeting in a circle with radius 50 meters form as interesting a pattern as 19 caribou meeting in a circle with radius 51 meters.' Benkert et al. [23] and Gudmundsson and van Kreveld [82] only recently revisited the flock pattern and gave a more generic definition that bases purely on the geometric arrangement of the moving entities and thus excludes the need of an analytical space as with the initial definition of the patterns [111, 114].

The model used in the REMO framework considers each time-step separately, that is, given $m \in \mathcal{N}$ and $r > 0$ a flock is defined by at least m entities within a circular region of radius r and moving in the same direction at some point in time. Benkert et al. [23] argued that this is not enough for many practical applications, e.g. a group of animals may need to stay together for days or even weeks before it is defined as a flock. They proposed the

following definition of a flock:

Definition 4.1 (m, k, r)-flock - Let $m, k \in \mathcal{N}$ and $r > 0$ be given constants. Given a set of n trajectories, where each trajectory consists of τ line segments, a flock in a time interval $I = [t_i, t_j]$, where $j - i + 1 \geq k$, consists of at least m entities, such that for every point in time within I there is a disk of radius r that contains all the m entities.

We will use a similar model when defining the leadership patterns, see Section 4.2. Using this model, Gudmundsson and van Kreveld [82] recently showed that computing the longest duration flock and the largest subset flock is NP-hard to approximate within a factor of $\tau^{1-\varepsilon}$ and $n^{1-\varepsilon}$, respectively, for any constant $\varepsilon > 0$. In the same model, Benkert et al. [23] described an efficient approximation algorithm for reporting and detecting flocks, where they let the size of the region deviate slightly from what is specified. Approximating the size of the circular region with a factor of $\Delta > 1$ means that a disk with radius between r and Δr that contains at least m objects may or may not be reported as a flock while a region with a radius of at most r that contains at least m entities will always be reported. Their main approach is a $(2 + \varepsilon)$-approximation (for any constant $\varepsilon > 0$) with running time $T(n) = O(kn(2^k \log n + k^2/\varepsilon^{2k-1}))$. Note that even though the dependency on the number of entities (namely n) is small, the dependency on the duration of the flock pattern (namely k) is exponential. Al-Naymat et al. [7] handle the problem of considering many entities and long-duration patterns by using a preprocessing step where the number of dimensions (i.e. time-steps) is reduced by random projection.

A series of articles exploring simple flocking illustrated the potential of patterns based on the geometric arrangement of moving entities. This chapter shall achieve a similar definition for the more complex pattern leadership as well as efficient algorithms for its detection.

4.2 Leadership

We consider n entities moving in the two dimensional plane during the time interval $[t_1, t_\tau]$, see Figure 4.1(a) for an example. The infinite set T_p of time-points is defined as $T_p = \{t \mid t \in [t_1, t_\tau]\}$, and the set T_s of time-steps is the set of discrete time-points given as input, i.e. $T_s = \{t_1, ..., t_\tau\}$. We specify open and closed time intervals by (t_x, t_y) and $[t_x, t_y]$, respectively. A unit-time-interval is an open interval I between two consecutive time-steps, i.e. $I = (t_{x-1}, t_x)$, for a time-step t_x with $x > 1$.

4.2.1 Defining Leadership Patterns

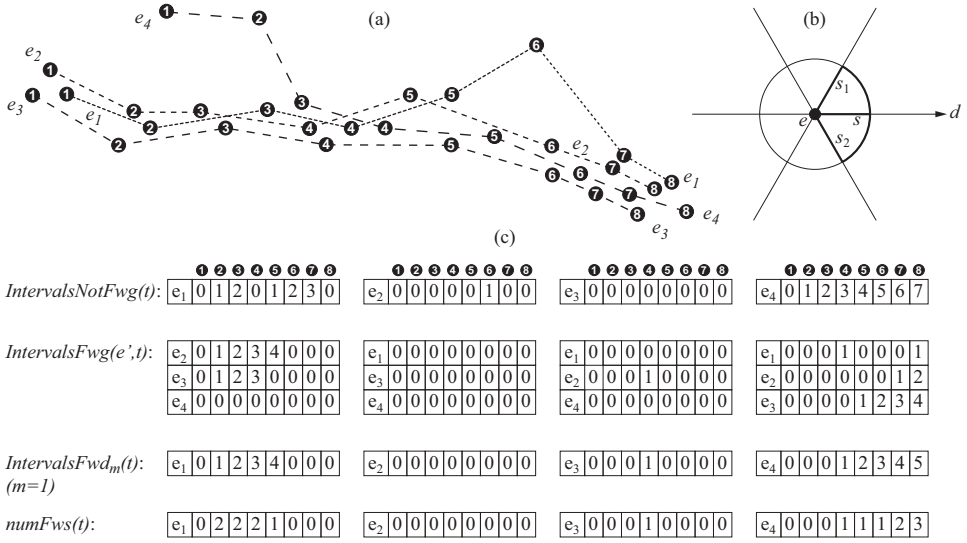

Figure 4.1: (a) A set of 4 entities moving from left to right over 7 unit-time-intervals, i.e. over 8 time-steps. (b) Illustrating the definition of the front-region as the disc-segment within bold lines. (c) The follow-arrays of the four entities, where we use the front-region as depicted in (b) and $\alpha = \beta$.

For describing our leadership patterns, we need a couple of parameters specifying these patterns. More specifically, we assume that we are given numbers m (specifying the size of a pattern, i.e. the minimum number of entities involved in a pattern), k (the minimum temporal length of a pattern), a radius r (influencing the spatial size of a pattern), an angle α (also influencing the spatial size of a pattern) and an angle β (determining spatial characteristics of a pattern). We consider them as constants during the rest of the chapter, i.e. we will not carry them along as parameters of functions or other notations.

At time-point t_x, an entity e_j is located at a position with coordinates $xpos(e_j, t_x)$ and $ypos(e_j, t_x)$. As we do not have spatial information of an entity between two time-steps we make the following assumption for the remainder of this chapter.

Assumption 1 *We assume that all entities move between two consecutive time-steps with constant direction and constant velocity.*

The same assumption has been used in earlier work [23]. It enables us to interpolate the positions of entities between time-steps. Even though we have no bound on the accuracy of this interpolation compared to the real positions of the entities, it appears to be a reasonable approach when tackling our leadership problems, as long as the sampling of points on the trajectories is sufficiently dense.

Suppose we are given an entity e_j at time-point t with $t_{x-1} < t < t_x$ for $t_x \in T_s$. We say e_j is heading into *direction* d where d is an angle in $[0, 2\pi)$ that is specified by the line segment e_j is moving along between time-steps t_{x-1} and t_x. (If e_j does not move between t_{x-1} and t_x then we define d to be the direction of the line segment e_j is moving along between the time-steps t_{x-2} and t_{x-1}. If no such time-steps exist, then we define $d := 0$.) The difference between two directions d_1 and d_2 is denoted by $||d_1 - d_2||$, and it is an absolute value, i.e. it is an angle in $[0, \pi]$. We declare the direction of an entity at a time-step t_x to be *undefined*, because at time-steps an entity might change its direction. However, the *direction* of an entity e_i at a time-step t_x *with respect to* (t_{x-1}, t_x) is the direction e_i is heading to at any time-point in (t_{x-1}, t_x). Therefore, when considering time intervals with certain properties of entities depending on direction, we implicitly exclude time-steps from those intervals in the remainder of this chapter.

Given an entity e and a time-point $t \notin T_s$, we define the *front-region* of e at time t in the following way. Consider the disk C with radius r centered at $(xpos(e,t), ypos(e,t))$. Furthermore, consider three line segments s_1, s_2 and s of length r, all having one end point at $(xpos(e,t), ypos(e,t))$. Segment s points in the direction d that e is heading to at time t, and segments s_1 and s_2 are the well defined segments forming angles of $\frac{\alpha}{2}$ and $-\frac{\alpha}{2}$ with s, respectively. The part of the disk C that contains s and is bounded by the segments s_1 and s_2 is the *front-region*, see Figures 4.1(b) and 4.2. We denote this wedge-shaped region by $front(e)$ *at time* t. An entity e_j is said to be *in front* of an entity e_i at time $t \notin T_s$ if and only if $e_j \in front(e_i)$ at time t.

Definition 4.2 Let d_i and d_j be the directions of the entities e_i and e_j at time $t \notin T_s$, respectively. Entity e_i is said to *follow* entity e_j at time t, iff $e_j \in front(e_i)$ at time t and $||d_i - d_j|| \leq \beta$.

An entity e_j is said to follow entity e_i at time $[t_x, t_y]$ for time-points t_x, t_y, if and only if e_i follows e_j at time t for all time-points $t \in [t_x, t_y] \setminus T_s$.

Definition 4.3 An entity e_i is said to be a *leader* at time $[t_x, t_y]$ for time-points t_x, t_y, if and only if e_i does not follow anyone at time $[t_x, t_y]$, and e_i

is followed by sufficiently many entities at time $[t_x, t_y]$. If there is an entity that is a leader of at least m entities for at least k time units, we have a *leadership pattern*.

See Figure 4.2 for an example of some notations.

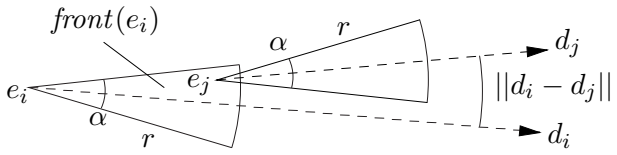

Figure 4.2: The front regions of e_i and e_j as wedges of edge length r and apex angle α. Entity e_j is in front of e_i. The entities are heading into directions d_i and d_j, respectively. If $||d_i - d_j|| \leq \beta$ then e_i follows e_j.

Example 1 *Consider the entities in Figure 4.1(a) where we use a front-region as depicted in Figure 4.1(b). We see that e_2 is following e_1 at time (t_1, t_5), e_1 is not following any other entity at time (t_1, t_3) and (t_4, t_7) and hence e_1 is a leader of e_2 at time (t_1, t_3) and (t_4, t_5).*

In the remainder of this section, we consider two entities e_i and e_j and two consecutive time-steps t_{x-1} and t_x. The next lemma tells us that if we want to check whether an entity is following any other entity during the entire interval (t_{x-1}, t_x), we only have to check this at the two end points with respect to (t_{x-1}, t_x). The lemma is rather intuitive and can be proven with very much the same ideas as in the proof of Lemma 2 in [23].

Lemma 4.4 *Let e_i and e_j be two entities, and let t_{x-1} and t_x be two consecutive time-steps. If e_i follows e_j at time-points t_y and t_z with $t_{x-1} < t_y \leq t_z < t_x$ then under Assumption 1, e_i follows e_j at any time-point $t \in [t_y, t_z]$.*

Note that the lemma is also true for $t_{x-1} = t_y$ and $t_z = t_x$, however, the directions of the entities at these time-points are with respect to (t_{x-1}, t_x). Therefore, the time that an entity e_i follows another entity e_j between two consecutive time-steps t_{x-1} and t_x is a single subinterval of $[t_{x-1}, t_x]$, and such an interval can be computed in a straightforward manner.

Lemma 4.5 *Given two entities e_i and e_j and two time-steps t_{x-1} and t_x, we can compute in constant time the subinterval of $[t_{x-1}, t_x]$ for which $e_i \in front(e_j)$ and for which e_j follows e_i, under Assumption 1.*

4.2.2 Problem Statement

A leadership pattern exists if there is an entity that is a leader of sufficiently many entities over a long enough series of time-steps or time-points. Such a pattern is characterized by two values m which is the size of the set of followers, and k which is the length of a pattern. As mentioned in related work [23, 83] specifying exactly which of the patterns should be reported is often a subject for discussion. For instance, a leadership pattern of length exactly $k+1$ (starting at time-step t_x) implies the existence of two leadership patterns of length exactly k (albeit 'overlapping', one starting at time-step t_x and the other starting at time-step t_{x+1}). However, the pattern of length $k+1$ might be more interesting to report from a practical point of view. Therefore, we consider the following problems where we assume that m and k are constants.

- LP-report-all: For each entity e, report all time intervals where e is a leader of at least m entities for at least k time units.
- LP-max-length: Compute the length of a longest leadership pattern of size at least m, i.e. compute the largest value k^{max} such that there is an entity e that is a leader of at least m entities for k^{max} time units.
- LP-max-size: Compute the size of a largest leadership pattern of length at least k, i.e. compute the largest value m^{max} such that there is an entity e that is a leader of m^{max} entities for at least k time units.

All these problems come in four different flavors which are combinations of the modeling of the time axis (discrete vs. continuous) and the consistency of the set of followers (varying vs. non-varying).

More specifically, we consider each of the problems in a discrete case, where patterns (and follow behaviour) can only start and end at the discrete time-steps. In this discrete model, we can ensure that patterns exist, since we have the coordinates of the entities for all time-steps. Unlike this, patterns can start and end at any time-points in the continuous case. As discussed above, the data for the continuous case relies on Assumption 1. Recall that we do not have any guarantee on the accuracy of the linear interpolation between time-steps. This possible inaccuracy carries over to a possible inaccuracy of the reported leadership patterns in the continuous model. However, the continuous model is likely to become more important in the future, when huge data sets over many time-steps are available, which might need to be simplified in order to reduce storage space and processing

time. Simplified trajectories are likely to be non-synchronous, yet they can approximate the original trajectory within a fixed specified error bound (see e.g. [27, 76]).

The other variation concerns the set of followers. If there is a subset S of entities such that for each time-point of the duration of the pattern all entities in S follow the leader (there may be additional followers as well at some time-points), then we call this a non-varying (subset) leadership pattern. In contrast to this, if we allow the subset of followers to change from one unit-time-interval to the next during the duration of the pattern (some entities may drop out, others may join in), then we call such a pattern a varying (subset) leadership pattern, as long as always at least m entities are following at each unit-time interval of the pattern. Depending on the application a non-varying or a varying set of followers might be desirable.

4.3 Algorithms for the Discrete Case

In the discrete case, patterns can only start and end at time-steps. We first describe arrays storing information about the follow behaviour of the entities with respect to a fixed entity e_i. Later, these arrays will be used to solve our leadership problems.

4.3.1 Getting Ready – Computing Follow-Arrays for an Entity e_i

For an entity e_i to determine whether it is a leader at the time (t_x, t_y), we need to know whether e_i is not following any other entity and whether e_i is followed by sufficiently many entities at (t_x, t_y). We consider e_i at this time as a potential leader, and we compute a couple of follow-arrays called '$IntervalsNotFwg(t_x)$', '$IntervalsFwg(t_x, e_j)$', '$IntervalsFwd_m(t_x)$' and '$NumFws(t_x)$'. The first three arrays store the number of consecutive unit-time-intervals that there is a certain follow-behaviour. In contrast to this, the fourth array stores the number of entities with a certain follow-behaviour.

IntervalsNotFwg: (short for 'the number of unit-time-intervals e_i is not following at t_x') The array $IntervalsNotFwg(t_x)$ is a one dimensional array storing nonnegative integers. Such an integer for time-step t_x specifies for how many past consecutive unit-time-intervals (the last one ending at t_x) e_i is not following any other entity. That is, if $IntervalsNotFwg(t_x) = y$, then e_i is not following any other entity during the time interval (t_{x-y}, t_x). To compute the values of the *IntervalsNotFwg*-array, we use two nested loops. The outer loop runs from $t_x = t_2, ..., t_\tau$ (we start at $t_x = 2$ and set

IntervalsNotFwg$(t_1) := 0$). The inner loop ranges over $e_j = e_1, ..., e_n$ and $e_j \neq e_i$. After each round of the inner loop we update *IntervalsNotFwg*(t_x) according to whether we found an entity e_j such that e_i follows e_j at time (t_{x-1}, t_x). According to Lemma 4.5 each such single test can be done in constant time.

IntervalsFwg: (short for 'the number of unit-time-intervals e_i is followed by e_j at t_x') The array *IntervalsFwg*(t_x, e_j) is a $(\tau \times n - 1)$ matrix storing nonnegative integers specifying for how many past consecutive unit-time-intervals (the last one ending at t_x) e_j is following e_i (with $e_j \neq e_i$). Filling the *IntervalsFwg*-array with the right values can also be done with two nested loops, one outer loop for $t_x = t_2, ..., t_\tau$ and one inner loop for $e_j = e_1, ..., e_n$ and $e_j \neq e_i$. Initially set *IntervalsFwg*$(t_1, e_j) := 0$. We test whether e_j follows e_i at the unit-time-interval (t_{x-1}, t_x), and if so, we update *IntervalsFwg*(t_x, e_j).

IntervalsFwd$_m$: (short for 'the number of unit-time-intervals e_i has at least m followers at t_x') The array *IntervalsFwd*$_m(t_x)$ is a one-dimensional array storing integers specifying for how many consecutive past unit-time-intervals (the last one ending at t_x) there are at least m entities following entity e_i. These m entities can be varying over time. Given the array *IntervalsFwg*, computing the *IntervalsFwd*$_m$-array can be done by looping over the *IntervalsFwg* array. We start at $t_x = 2$ and set *IntervalsFwd*$_m(t_1) := 0$. Now, we count in each column of *IntervalsFwg* (if we imagine the array *IntervalsFwg* to be arranged to have τ columns and $n - 1$ rows) the number of entities following e_i at the current time-step. If this number is smaller than m, we set *IntervalsFwd*$_m(t_x) := 0$, and if this number is at least m, we set *IntervalsFwd*$_m(t_x) :=$ *IntervalsFwd*$_m(t_{x-1}) + 1$.

NumFws: (short for 'the number of followers of e_i at t_x') Another array is *NumFws*(t_x) which is a one-dimensional array storing integers specifying how many entities are following entity e_i at time (t_{x-1}, t_x). Again, counting in each row of *IntervalsFwg* the number of entities following e_i at the current time-step yields the corresponding value of the *NumFws* array.

From the above discussion on the corresponding arrays, we conclude with the following lemma.

Lemma 4.6 *The IntervalsNotFwg, IntervalsFwg, IntervalsFwd$_m$ and NumFws-arrays for an entity e_i can be computed in $O(n\tau)$ time and space.*

Example 2 *Consider the entities in Figure 4.1(a) where we use a front-region as depicted in Figure 4.1(b). Figure 4.1(c) shows four columns (one for each entity) of follow-arrays. To fill the arrays IntervalsNotFwg and*

IntervalsFwg, we need the trajectories and the front-regions. Once that is done, the arrays IntervalsFwd$_m$ and NumFws can be computed according to their definition.

4.3.2 Solving LP Problems with a Non-varying Subset of Followers

LP-report-all

In the discrete leadership version we assume that patterns can only start and end at time-steps $T_s = \{t_1, ..., t_\tau\}$. We use the arrays *IntervalsNotFwg* and *IntervalsFwg*, and we combine their information to determine whether e_i is a leader of a non-varying-subset of followers. To this end, we look for time-steps t_x such that $\mathit{IntervalsNotFwg}(t_x) \geq k$. For each such time-step t_x, we inspect the array $\mathit{IntervalsFwg}(t_x, e_j)$ for $j = 1, ..., n$ and $j \neq i$, and we count the number of times that $\mathit{IntervalsFwg}(t_x, e_j) \geq k$. Let $m(k)$ denote this number. Now we can report e_i as a leader for every time-step t_x for which $m(k) \geq m$. As we only need to traverse our arrays at most once, this can be done on $O(n\tau)$ time.

Example 3 *Let $k = 1$ and $m = 2$. Looking at the follow-arrays of entity e_1 in Figure 4.3, we see (shaded region) that e_1 is not following anyone, but is followed by 2 entities, and this happens for at least $k = 1$ unit-time-intervals at the time-steps 2 and 3. Hence, we would report two leadership-patterns with e_1 as leader.*

So far, we have seen that we can compute in $O(n\tau)$ time and space at which time-steps an entity e_i is a leader. To find all leadership patterns amongst a set of entities we test any entity individually. As we only have to store one instance of each array at a time we can conclude with the following lemma.

Lemma 4.7 *Reporting all non-varying-subset leadership patterns of size at least m and length at least k, amongst n trajectories over τ time-steps can be done in $O(n^2\tau)$ time and $O(n\tau)$ space.*

LP-max-length

To compute the length of a longest pattern, where e_i is the leader, we utilise a variable k^{max}. Initially we set $k^{max} := 0$; we then loop once over all time-steps and at each time-step we may modify k^{max}, and at the end k^{max}

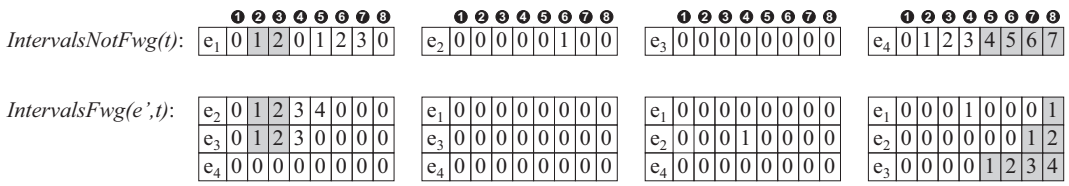

Figure 4.3: Follow-arrays with highlighted entries to mark patterns with a non-varying subset of followers.

will be equal to the length of a longest leadership pattern (for a specific m). Now, for each $t_x = t_1, ..., t_\tau$ we check whether $\mathit{IntervalsNotFwg}(t_x) > k^{max}$ and if so, we do the following. We inspect the column of the array *IntervalsFwg* corresponding to t_x. We traverse that column (i.e. we loop for $j = 1, ..., n,\ j \neq i$), and we count the number of entities e_j for which holds that $\mathit{IntervalsFwg}(t_x, e_j) > k^{max}$. Let this number be denoted by $m(k^{max})$. If $m(k^{max}) \geq m$, then we have at least m entities following e_i for more than k^{max} unit-time-intervals, and e_i is not following anyone during that time. Hence, we increase k^{max} by one and proceed with the next time-step t_{x+1}. Note that we only increase k^{max} by one as t_x is the first time-step for which $m(k^{max}) \geq m$. As we only traverse the entire arrays once, it takes $O(n\tau)$ time to compute the longest pattern, where e_i is the leader.

The following concluding lemma might surprise, as the longest duration flock pattern is NP-hard to compute and cannot even be approximated within a factor of $\tau^{1-\varepsilon}$ [82].

Lemma 4.8 *The longest duration leadership pattern for a non-varying-subset of followers of size at least m can be computed in $O(n^2\tau)$ time and $O(n\tau)$ space.*

Example 4 *Consider again Figure 4.3. For $m = 1$, the above described method would find entity e_4 to be the leader (of one entity, namely e_3) for four consecutive unit-time-intervals, which is the length of a longest pattern (for $m = 1$).*

LP-max-size

It is also possible to compute the size of a largest non-varying-subset of followers that follows a leader for at least k unit-time-intervals. We utilise the arrays *IntervalsNotFwg* and *IntervalsFwg* and a variable m^{max}, initially set to 0. We update this variable whenever we find a larger set of followers.

That is, for $t_x := t_1, ..., t_\tau$, we test if both $IntervalsNotFwg(t_x) \geq k$ and $m(k) > m^{max}$, and if so, we set $m^{max} := m(k)$, where $m(k)$ is defined in the same way as $m(k^{max})$ in the section above. Hence, we obtain the following lemma.

Lemma 4.9 *The size of a largest non-varying-subset of entities that follow a leader for at least k time-steps can be computed in $O(n^2\tau)$ time and $O(n\tau)$ space.*

Example 5 *Consider again Figure 4.3, and let $k = 1$. The algorithm above computes $m^{max} = 3$ as entity e_4 is a leader of 3 entities for $k = 1$ unit-time-interval at time-step 8.*

4.3.3 Solving LP Problems with a Varying Subset of Followers

The variants of the problem of finding leadership patterns where the set of followers can change during the leadership pattern can be solved in a similar way as proposed in Section 4.3.2. To determine if an entity e_i is a leader of a varying-subset of followers, we use the follow-arrays $IntervalsNotFwg(t_x)$, $IntervalsFwd_m(t_x)$ and $NumFws(t_x)$ as described in Section 4.3.1.

LP-report-all

In the same flavor as described above, we can find out if e_i is a leader. We look for and report time-steps t_x, such that $IntervalsNotFwg(t_x) \geq k$ and $IntervalsFwd_m(t_x) \geq k$. It is easy to see that reporting when e_i is a leader can be done in $O(n\tau)$ time.

Example 6 *Let $m = 1$ and $k = 2$. Consider e_1's follow-arrays in the upper half of Figure 4.4. Above method reports one time-steps (namely time-step 3) where e_1 is a leader of at least $m = 1$ entities for at least $k = 2$ unit-time-intervals.*

The complexity of finding all leadership patterns for n entities is summarized as follows.

Lemma 4.10 *Reporting all varying-subset leadership patterns amongst n trajectories over τ time-steps can be done in $O(n^2\tau)$ time and $O(n\tau)$ space.*

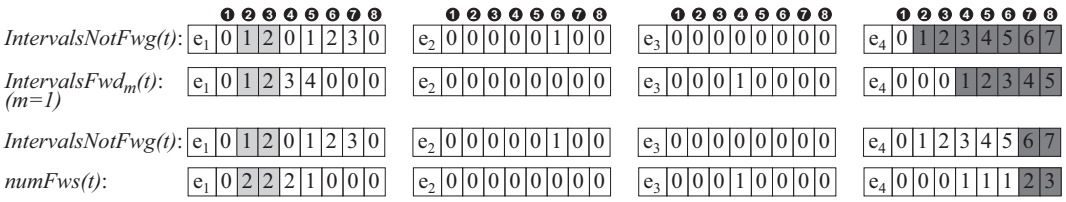

Figure 4.4: Follow-arrays with highlighted entries to mark patterns with a varying subset of followers.

LP-max-length

For computing the longest duration leadership pattern, we use the arrays *IntervalsNotFwg* and $IntervalsFwd_m$, and we search for the largest k^{max} (initially $k^{max} := 0$) such that there is a time-step t_x for which $IntervalsNotFwg(t_x) \geq k^{max}$ and $IntervalsFwd_m(t_x) \geq k^{max}$. This can be done as follows. For $t_x = t_1, ..., t_\tau$, we check if $\min\{IntervalsNotFwg(t_x), IntervalsFwd_m(t_x)\} > k^{max}$, and if so, we perform an update

$$k^{max} := \min\{IntervalsNotFwg(t_x), IntervalsFwd_m(t_x)\}$$

and proceed with the next time-step t_{x+1}.

Lemma 4.11 *The longest duration leadership pattern for a varying-subset of followers of size at least m can be computed in $O(n^2\tau)$ time and $O(n\tau)$ space.*

Example 7 *Looking at the follow-arrays in the upper half of Figure 4.4, we see that e_4 is a leader of at least $m = 1$ entity for $k^{max} = 5$ unit-time-intervals (starting at time-step 3 and ending at time-step 8).*

LP-max-size

If we would like to compute the size of a largest varying set of followers that follow e_i for at least k time-steps, we cannot use the array $IntervalsFwd_m$ directly as this array contains information only for one specific m. However, an easy way is to use binary search on m and recompute the $IntervalsFwd_m$ array for each value of m. This adds an additional $\log n$ factor to the running time.

We propose a slightly different approach. By spending linear preprocessing time, we can compute the minima of any substring of a sequence of

numbers in $O(1)$ time. For more information on this Range Minimum Query (RMQ), see e.g. [22]. Now, we use the array *NumFws* and we look for at least k consecutive unit-time-intervals such that the minimum number of followers in the array *NumFws* during that time is as large as possible and e_i can be a leader. That is, we are looking for k consecutive unit-time-intervals such that e_i does not follow any other entity and for the largest minimum (to be referred to as m^{max}) over all numbers of followers corresponding to those k consecutive unit-time-intervals. All minima can be computed in $O(\tau)$ time [22], hence, m^{max} can be computed in linear time.

Lemma 4.12 *The size of a largest varying-subset of entities that follow a leader for at least k time-steps can be computed in $O(n^2\tau)$ time and $O(n\tau)$ space.*

Example 8 *Consider the lower half of Figure 4.4 and let $k = 2$. The above algorithm computes $m^{max} = 2$ at time-step 3 for entity e_1 and at time-steps 8 for entity e_4.*

4.4 Algorithms for the Continuous Case

In contrast to the discrete version of the leadership pattern, where a pattern can only start or end at the given discrete time-steps, in the continuous version of the problem a pattern can start and end at any point in time. As we do not have spatial information of the entities between two consecutive time-steps we use Assumption 1 to tackle the continuous version in this section. The main ideas are similar to the discrete case, but instead of using arrays storing single numbers to represent follow-behaviour we will use sets of time intervals. First, we describe how to compute them for a fixed entity e_i and then we define two operations on (sets of) intervals. Later, these intervals and operations are used to solve our leadership problems.

4.4.1 Getting Ready – Follow-Intervals for an Entity e_i

Computing Follow-Intervals: A first step is to compute a set *SetNotFwg* of notfollowing-intervals representing when a fixed entity e_i is not following any other entity e_j. An interval $I = (t_{x_a}, t_{y_a}) \in \mathit{SetNotFwg}$ with $t_{x_a} \leq t_{y_a}$ means that entity e_i is not following any other entity during the whole time interval I. Because entities move on a straight line between two consecutive time-steps, cf. Assumption 1, e_i can be involved in at most two events that change its follow-behaviour (i.e. the events of beginning or ending to follow)

for each entity between two consecutive time-steps. That is why the set *SetNotFwg* contains $O(n\tau)$ intervals. We can compute this set with two nested loops one over all time-steps, another over all entities. By Lemma 4.5, this can be done in $O(n\tau)$ time in total.

We also need information about which entities follow e_i. This information is again stored in a set *SetFwd* of intervals. An interval $I = (t_{x_a}, t_{y_a}) \in$ *SetFwd* with $t_{x_a} < t_{y_a}$ means that e_i is followed by an entity, say e_j, during the whole time interval I. Also this set contains at most $O(n\tau)$ intervals, as an entity can change its follow behaviour with respect to e_i at most twice between two consecutive time-steps. We can compute this set with two nested loops one over all time-steps, another over all entities. By Lemma 4.5, this can be done in $O(n\tau)$ time in total.

Both sets of intervals can be computed in $O(n\tau)$ time. For the subsequent methods, however, we need the start- and end points of the intervals in non-decreasing order and that the intervals are maximal. Obtaining the sets such that the start- and end points are sorted can be done in $O(n\tau \log n)$ time in the following way. For each set we use two nested loops. The outer loop fixes an entity and the inner loop ranges over the time-steps. In that way it is easy to compute the intervals as maximal intervals. Whenever we compute start- or end points of an interval we can put them into $\tau - 1$ buckets, namely one for each unit-time-interval, i.e. pair of consecutive time-steps. As we can have at most $O(n)$ start- or end points in each bucket, we can sort all of them in all buckets in $O(n\tau \log n)$ time. Combining the sorted sequences of each bucket results in a sorted sequence of all start- and end points.

Lemma 4.13 *In $O(n\tau \log n)$ time and $O(n\tau)$ space, the sets SetNotFwg and SetFwd for an entity e_i can be computed such that all the start- and end points of the intervals in each set are output in non-decreasing order.*

Next, we define operations that take and return a set of intervals as input and output. We also briefly describe how to compute these operations, if the set of intervals is given along with the start- and end points in sorted order.

Combining Intervals: We call the first operation under consideration *interval-combination* denoted as $ic_x(S)$, where S is a set of intervals of $\mathbb{R}$. The operation outputs a set of non-intersecting intervals. Every point in $\mathbb{R}$ that is contained in at least x intervals of the input-set S will be in an interval of the output-set. Also, for every point that is contained in an interval of the output-set, there are at least x intervals in the input-set that

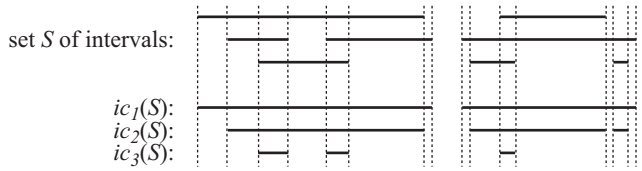

Figure 4.5: The set S of intervals on the real line and the results after applying the ic_x operation for $x \in \{1, 2, 3\}$. Note that $ic_x(S) = \emptyset$ for all $x \geq 4$ as the intersection of any 4 intervals in S is empty.

all contain that point, see Figure 4.5. Note that $ic_1(S)$ is the union of all intervals in S and $ic_{|S|}(S)$ is the intersection of all intervals in S. Let S be a set of intervals where the start- and end points are given in sorted order. The operation $ic_x(S)$ can be computed by a parallel scan over the sorted start- and end points and keeping track of how many intervals are currently 'active'.

Lemma 4.14 *Suppose S is a set of intervals. If the start- and end points of the intervals in S are given in non-decreasing order, then we can compute $ic_x(S)$ in $O(|S|)$ time.*

Clipping Intervals: We also define another operation, which modifies single intervals. For an interval $I = \{t_{x_a}, t_{x_b}\}$, we cut or clip a part of length k at the beginning of I. If the resulting interval I' is non-empty, then that interval I' is the result of the operation. This operation can also be applied to all intervals of an entire set (cf. Figure 4.6), such that the order of the start- and end points of the intervals remains stable.

Lemma 4.15 *Let be given a set S of intervals, where the start- and end points of the intervals in S are given in non-decreasing order. We can compute $S' := \{I' \mid I' = clip_k(I), I \in S\}$ and output the start- and end points of all intervals in S' in non-decreasing order in $O(|S|)$ time.*

4.4.2 Solving LP Problems with a Non-Varying Subset of Followers

LP-report-all

We first look at the non-varying-subset version. In the previous section we have seen that we can compute the interval-set *SetFwd* in $O(n\tau \log n)$ time, where an interval in this set means that an entity follows e_i for the time of

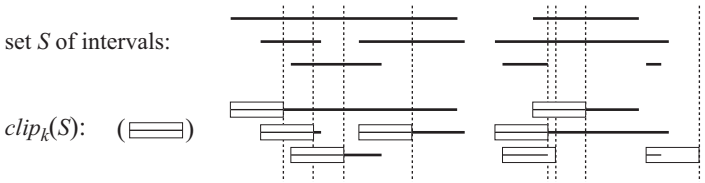

Figure 4.6: The set S of intervals on the real line and the results after applying the $clip_k$ operation. For the $clip_k$ operation, the length of the interval in parentheses determines k.

the interval. Now, we are going to modify the intervals in the set $SetFwd$. For each interval $I = \{t_{x_a}, t_{x_b}\} \in SetFwd$, we apply the operation $clip_k$ to obtain a set $SetFwd_{clipped} := \{I' \mid I' = clip_k(I), I \in SetFwd\}$. Note that $SetFwd_{clipped}$ (see Figure 4.7) only contains intervals whose originals had length at least k. The meaning of an interval $I' \in SetFwd_{clipped}$ is that there is an entity such that at each time-point $t \in I'$ this entity has already followed e_i for at least k time units (which is not necessarily the same as k unit-time intervals). The set $SetFwd_{clipped}$ can be computed in linear time with respect to the size of $SetFwd$, and this can be implemented such that the order of the (start- and end points of the) intervals remains stable.

We also clip the intervals of the set $SetNotFwg$ to obtain a set $SetNotFwg_{clipped} := \{I' \mid I' = clip_k(I), I \in SetNotFwg\}$ (see Figure 4.7). For each time-point in an interval in $SetNotFwg_{clipped}$, we have that e_i is not following any other entity for at least k time units.

The next step is to compute yet another set S of intervals as an interim result using one of the operations introduced in Section 4.4.1, $S := ic_m(SetFwd_{clipped})$. For any time-point in an interval in S there are at least m entities following e_i, where each of those entities already followed e_i for at least k time units. Finally, we combine the information represented by S and $SetNotFwg_{clipped}$. What we need is similar to a logical 'and' between intervals of those two sets, and this can be done by applying the ic_x again to obtain a set of $result$-intervals, $result := ic_2(S \cup SetNotFwg_{clipped})$. Note that the start- and end points of the set $S \cup SetNotFwg_{clipped}$ can be sorted in linear time if the start- and end points of S and $SetNotFwg_{clipped}$ are sorted. The set $result$ contains all intervals for which e_i is a leader of at least m entities for at least k time units. If we would like to report the leadership patterns of all entities, we apply the above method to each entity. Hence, we can conclude with the following lemma.

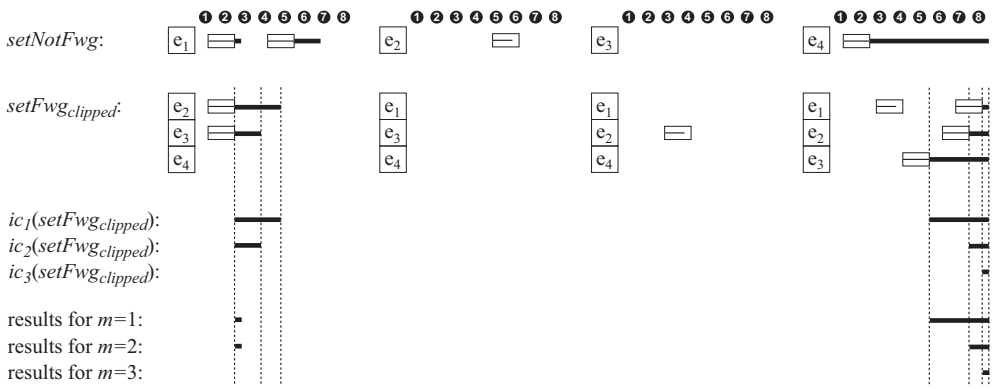

Figure 4.7: Illustration of clipping and combining the intervals, where the intervals represent the follow-behaviour of the entities in Figure 4.1. The *result*-intervals are shown for different values of m.

Lemma 4.16 *Let be given n trajectories over τ time-steps. Reporting all time intervals where there is an entity a leader of a non-varying subset of at least m entities for at least k time units, can be done in $O(n^2\tau \log n)$ time and $O(n\tau)$ space.*

LP-max-size

We can use the sets $SetNotFwg_{clipped}$ and $SetFwd_{clipped}$, where the intervals are given in non-decreasing order, to find the maximum m^{max} for which e_i is a leader of a non-varying set of m^{max} entities for at least k time units. To that end, we do not collapse the set $SetFwd_{clipped}$ into a set S as described above, but we utilise a parallel scan over the intervals in $SetNotFwg_{clipped}$ and $SetFwd_{clipped}$.

By a parallel scan we mean moving an imaginary vertical line over the horizontally arranged intervals, stopping at certain points and performing certain actions. In our case the points where we stop are the start- and end points of the intervals. For any position of the scan-line we say an interval I is *active*, if the scan-line ℓ intersects interval I.

During the parallel scan, we keep track of the number of active intervals in $SetFwd_{clipped}$, where the intervals in $SetNotFwg_{clipped}$ are used as a mask (see Figure 4.8). All this can be done in $O(n\tau)$ time.

Lemma 4.17 *Let be given n trajectories over τ time-steps. Computing the maximum size of a non-varying subset of followers which follow a leader for*

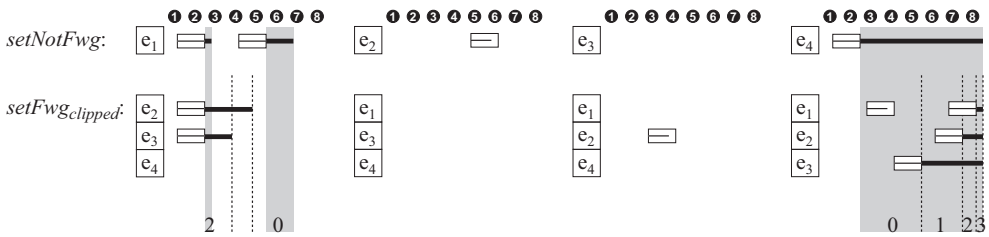

Figure 4.8: Illustrating the parallel scan approach. The shaded region indicates how the *SetNotFwg* intervals are used as a mask. The numbers indicate the number of active *SetFwd* intervals.

at least k time units, can be done in $O(n^2\tau \log n)$ time and $O(n\tau)$ space.

LP-max-length

A method similar to the one presented above cannot be used directly, as the sets of intervals are computed for specified values of k. We could use binary search on k, however, this would add another $\log \tau$ factor to the running time. The method described in this section also builds upon the sets *SetNotFwg* and *SetFwd* of intervals, introduced in Section 4.4.1. It finds the largest k^{max}, such that there is a non-varying subset of at least m entities following entity e_i for k^{max} time-units. However, it can also be used to report patterns where e_i is a leader of at least m non-varying followers for at least k time-steps. We do this by performing a parallel scan over the sets of intervals, assuming they are given such that the start- and end points of the intervals are in non-decreasing order.

During the parallel scan we keep track of the active intervals in *SetNotFwg*. Note that only one interval $I \in$ *SetNotFwg* can be active at a time. By keeping a pointer p_1 to I, we know for every time-point t, whether e_i is following any other entity. If e_i is following any other entity, then there is no interval in *SetNotFwg* active at time t (and p_1 becomes a null-pointer). On the other hand, if there is an interval $I \in$ *SetNotFwg* active at t, then we can compute for how long e_i is not following any other entity.

We also keep track of how many entities follow e_i and for how long. To this end, let t be a time-point during the parallel scan. Let $A \subseteq$ *SetFwd* be the set of all intervals in *SetFwd* that are active at time t. We will not maintain A, but only a variable m' with $m' = |A|$ (initially $m' := 0$). Furthermore, we will maintain a pointer p_2 to the interval in A with the m-th largest end point. If A contains less than m intervals, then p_2 points

to some interval. (As we will not use pointer p_2 if A contains less than m intervals it is not important where p_2 points to in that case.)

Before the parallel scan, we initialize $k^{max} := 0$, and after the parallel scan k^{max} will be the length of the longest leadership pattern with e_i as leader of a non-varying set of at least m entities. We also introduce an artificial interval which starts and ends before any other interval starts and we initialize p_2 to point to that interval. This interval is introduced merely to have pointer p_2 well initialized. The parallel scan does not take this interval into consideration. As mentioned above the points where we stop with the scan-line are the start- and end points of the intervals, and if two such points have the same time, we process them one after the other, as if one was infinitesimally later than the other. By maintaining all invariants it is easy to see that for every position of the scan-line with corresponding time t, we can check if there are at least m entities following e_i, i.e. if $m' \geq m$. In the case that there are at least m followers of e_i, we also can determine for how long in the future all these entities will follow e_i, by using the pointer p_2. Furthermore, we can check if e_i is following any other entity (by using p_1), and if not for how long in the future e_i will not follow any other entity. Therefore, we can determine whether there is a leadership pattern, with e_i as leader of a non-varying set of at least m entities, and if there is such a pattern, we can also determine its length k'. If $k' > k^{max}$ then we perform an update $k^{max} := k'$.

By doing this parallel scan approach for each entity, we can compute the overall longest duration leadership pattern.

Lemma 4.18 *Let be given n trajectories over τ time-steps. Computing the maximum length of a leadership pattern with a non-varying subset of followers of size at least m, can be done in $O(n^2\tau \log n)$ time and $O(n\tau)$ space.*

4.4.3 Solving LP Problems with a Varying Subset of Followers

LP-report-all

After considering the case for the non-varying subset in Section 4.4.2, the case for a varying subset is rather easy. Here, we do not require that all entities follow e_i for k time-units. Hence, with the terminology as used before we compute a set $S := ic_m(SetFwd)$. For any time-point in an interval in S there are at least m entities following e_i. As e_i still has to be followed for at least k time-units, we clip all intervals in S to obtain

$S' := \{I' \mid I' = clip_k(I), I \in S\}$. As before, our last step is to combine S' and $SetNotFwg_{clipped}$ to obtain the set of *result*-intervals, $result := ic_2(S' \cup SetNotFwg_{clipped})$.

Lemma 4.19 *Let be given n trajectories over τ time-steps. Reporting all time intervals where there is an entity a leader of a varying subset of at least m entities for at least k time units, can be done in $O(n^2\tau \log n)$ time and $O(n\tau)$ space.*

LP-max-size

In this case, we can use the approach mentioned in Section 4.3.3, where we spend additional time for binary search on m to find m^{max}.

Lemma 4.20 *Let be given n trajectories over τ time-steps. Computing the maximum size of a varying subset of followers which follow a leader for at least k time units, can be done $O(n^2\tau \log n)$ time and $O(n\tau)$ space.*

LP-max-length

To find the length of a longest duration leadership pattern of a varying set of at least m entities, we can use a similar approach as in Section 4.4.3. We also compute a set $S := ic_m(SetFwd)$, such that for each time-point in an interval $I \in S$, we know that there are at least m followers of e_i. To combine this with the information when e_i is not following any other entity, we apply the operation ic_x once again to obtain $result := ic_2(S \cup SetNotFwg)$. Now, for any interval in *result* it holds that e_i is not following any other entity, and also that e_i is followed by at least m entities. Searching for the length of the longest interval in *result* solves the problem at hand for entity e_i.

Lemma 4.21 *Let be given n trajectories over τ time-steps. Computing the maximum length of a leadership pattern with a varying subset of followers of size at least m, can be done $O(n^2\tau \log n)$ time and $O(n\tau)$ space.*

4.5 A Lower Bound for the Continuous Case

In this section we argue that it is likely that every algorithm for the continuous version of the leadership problem that detects leadership patterns between two consecutive time-steps in a set of n trajectories requires $\Omega(n^2\tau)$ time in the worst case. We will present a transformation from the problem Point-on-3-lines to a special case of our leadership problem.

Problem: POINT-ON-3-LINES
Instance: A set S of n lines in the plane.
Question: Is there a point in the plane that lies on at least three lines of S?

The POINT-ON-3-LINES problem was proven to be 3-SUM-hard [70]. This means that it is at least as hard as 3-SUM for which no subquadratic time algorithm has been found yet, which is an indication for an inherent hardness of the problems. For a weak model of computation a lower bound of $\Omega(n^2)$ for those problems exists [62].

Remark Note that the intersection of two or more intervals can be a single number. Therefore it is possible (although not very likely from a practical point of view) that a leadership pattern in the continuous case exists only for one time-point. Such a pattern has length $k = 0$.

The transformation is as follows. Consider a set $S = \{s_1, ..., s_n\}$ of n lines in the plane. Let ℓ_0 be a horizontal line such that all intersections between two lines of S lie below ℓ_0 (see Figure 4.9(a)), and let ℓ_{-1} be a line parallel to ℓ_0 such that all intersections between two lines of S lie above ℓ_{-1}. Such two lines can be computed in $O(n \log n)$ time. Without loss of generality, we assume that no line in S is parallel to ℓ_0. Let $\ell_{-n}, ..., \ell_{-2}, \ell_1, ..., \ell_{n-2}$ be $2n - 2$ lines parallel to ℓ_0, such that the distance between ℓ_i and ℓ_j equals $|i - j|$ times the distance between ℓ_{-1} and ℓ_0, for $i, j \in \{-n, ..., n - 1\}$, see Figure 4.9(b). For a line $s_i \in S$, let s'_i be the directed line segment that starts at the intersection of s_i and ℓ_{-i} and ends at the intersection of s_i and ℓ_{n-i}, as illustrated in Fig. 4.9(c). Let S' be the set of all directed line segments, i.e. $S' = \{s'_i \mid s_i \in S\}$.

The set S' can be viewed as a set of n trajectories over two time-steps, where every directed segment $s'_i \in S'$ corresponds to the trajectory of entity e_i starting at time t_{x-1} at the origin of s'_i and reaching the end point of s'_i at time t_x. For $\alpha = 0$, $\beta = \pi$ and $r = \infty$, an entity e_j follows an entity e_i if and only if s'_j and s'_i intersect and the entity e_i reaches the intersection point before e_j. Note that because of the construction, no two entities can be between ℓ_{-1} and ℓ_0 at the same time and hence when an entity passes through an intersection of line segments in S', no other entity passes through an intersection in S'.

Lemma 4.22 *There are at least three lines in S passing through the same point, if and only if there is a leadership pattern in S' with $m = 2$, $k = 0$, $r = \infty$, $\alpha = 0$ and $\beta = \pi$.*

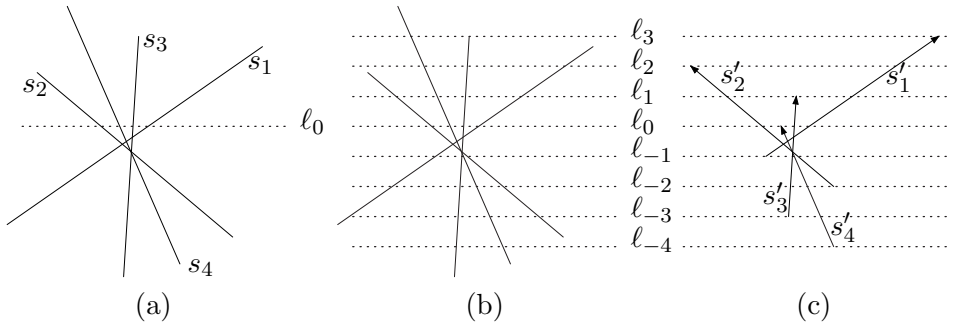

Figure 4.9: Illustrating a set S with $n = 4$ lines $s_1, ..., s_4$. (a) S and the horizontal line ℓ_0 (dotted). (b) S and the lines $\ell_{-n}, ..., \ell_{n-1}$ (dotted). (c) the set S'.

Proof: Suppose three lines s_i, s_j and s_k intersect in a single point p. Consider the three corresponding entities e_i, e_j and e_k with trajectories s'_i, s'_j and s'_k. W.l.o.g. assume that e_i reaches the intersection point p first. As p is an intersection point between lines in S it is also an intersection point between line segments in S', and therefore when e_i passes through p no other entity is between ℓ_{-1} and ℓ_0. Thus, no other entity can intersect s'_i when e_i passes through p, and hence, e_i is not following another entity when it passes through p. Furthermore, because $r = \infty$ and $\alpha = 0$ the entity e_i is in the front-regions of e_j and e_k. It follows that we have a leadership pattern in S' with $m = 2$ and $k = 0$.

Now, assume we have a leadership pattern in S' for $m = 2$, $k = 0$, $r = \infty$, $\alpha = 0$ and $\beta = \pi$. The leader e_i of that pattern will be followed by at least two entities, say e_j and e_k, at some point in time. This implies that s'_j and s'_k must intersect s'_i in the same point, and hence there are three lines in S passing through the same point. □

As the transformation described above can be done in $O(n \log n)$ time, we can conclude with the following lemma.

Lemma 4.23 *Finding continuous leadership patterns between two consecutive time-steps in a set of trajectories is* 3-SUM*-hard.*

4.6 Experimental Evaluation

This section is devoted to reporting the experimental results. The algorithms were implemented in Java[1] and all experiments were performed on a Linux operated PC with an Intel 3.6 GHz processor and 2 GB of main memory.

4.6.1 Input Data

All input files were generated artificially with NetLogo [163]. More specifically, we modified NetLogo's Flocking Model [162] such that entities do not wrap around the world-borders, but will be repulsed smoothly from walls, see Figure 4.10. Furthermore, we added some code for moderate random changes in an entity's direction and saving the coordinates into a file. There are many parameters to modify the behaviour of the entities and thus also to modify how many flocks and leadership-patterns are created. However, we have no direct control over the exact number or length or size of patterns.

We generated files with variable number of entities (128-4096), two different sizes of the underlying universe U (i.e. coordinate space 512×512 and 1024×1024) and two different characteristics CH (i.e. $CH = u$ and $CH = c$) of the entity distribution. $CH = u$ means that the parameters of the Flocking Model were chosen such that the entities are more *u*niformly distributed, i.e. only small clusters emerge. Flocks (and thus leadership patterns) still exist but their size and length are likely to be smaller than those of the other characteristic. $CH = c$ means that the parameters of the Flocking Model were chosen such that the entities form few but rather large *c*lusters, and hence, the flocks tend to contain more entities and have a longer duration. The number of time-steps is $\tau = 1000$.

4.6.2 Methods

We performed experiments with two variants of our algorithms for the discrete case. The first one is a straight forward implementation of the method described in Section 4.3. This method contains (among others) two nested loops ranging over all entities. The disadvantage from a practical point of view is that when looking for entities that might be in a front-region, then also entities that are too far away will be considered. Therefore, our second method tries to overcome this drawback by dividing the underlying plane into buckets (squares of side length r). Now when looking for entities that

[1] Java was chosen because this increases the platform independence and it makes it easier to integrate the code into an existing larger framework.

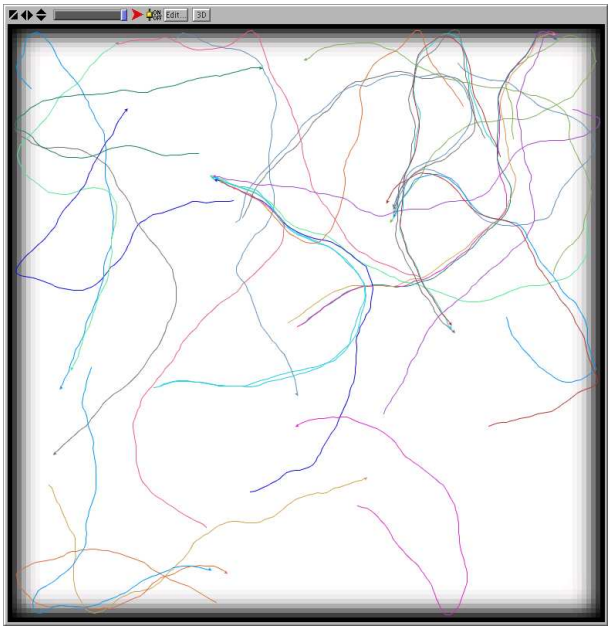

Figure 4.10: This screenshot of NetLogo's modified Flocking model shows the trajectories of 32 entities in a universe with side lengths 128×128, run for 100 timesteps.

might be in a front-region, only those entities will be considered that are in the nine neighboring buckets (including the bucket at the centre). Note that for each of our leadership problems, all methods always compute all arrays from scratch. Especially the arrays *IntervalsNotFwg* and *IntervalsFwg* could be used three times after computing them once. For an easier comparison however, we refrained from doing so.

4.6.3 Results

Tables 4.1, 4.2 and 4.3 show the results of our algorithms for $m = 10$, $k = 20$, $r = 20$, $\alpha = \pi$ and $\beta = \frac{\pi}{2}$. From our point of view the running times and their asymptotic behaviour are much more interesting than for example the exact number of patterns found as we deal with artificial data. Nevertheless, in Tables 4.1 and 4.2 we can see how many entities have been leaders (leaders), the number of leadership patterns found (report-all), the length of a longest duration pattern (max-length) and the number of entities in a pattern with most followers (max-size). Note that patterns with length $> k$ will be reported multiple times as patterns of length k.

We observed that the vast majority of the running time is spent on computing the arrays *IntervalsNotFwg* and *IntervalsFwg* (which can be done in $O(n^2\tau)$ time). Once these two arrays are computed, computing more arrays and/or extracting information to solve the leadership problem is very efficient (linear time). Therefore, our methods for the three different leadership problems result almost always in the same running times (they differ on average less than three percent), as they compute all arrays from scratch. Hence, Table 4.3 depicts the running times of our methods only for the report-all leadership problem.

4.6.4 Observations

Non-Varying vs. Varying:

As we could expect running times for the patterns with a varying subset of followers are often higher, as one more array is computed for the 'varying' problems. However, this increase is very marginal compared to other influencing factors. We can also observe that the values for the 'varying' patterns are at least as big (sometimes slightly larger) as for the 'non-varying' patterns. This is because a non-varying pattern is also a varying pattern by definition.

n	U	CH	non-varying			
			leaders	report-all	max-length	max-size
128	512^2	c	2	89	37	23
256	512^2	c	10	211	56	66
512	512^2	c	11	329	44	200
1024	512^2	c	27	676	46	197
2048	512^2	c	33	689	57	384
4096	512^2	c	44	966	55	812
128	1024^2	c	0	0	0	4
256	1024^2	c	0	0	2	8
512	1024^2	c	19	360	46	26
1024	1024^2	c	36	954	53	166
2048	1024^2	c	80	1536	47	219
4096	1024^2	c	98	2521	59	257
128	512^2	u	0	0	0	3
256	512^2	u	0	0	9	7
512	512^2	u	1	7	25	10
1024	512^2	u	5	15	24	13
2048	512^2	u	8	40	34	15
4096	512^2	u	6	36	29	14
128	1024^2	u	0	0	0	3
256	1024^2	u	0	0	0	5
512	1024^2	u	0	0	0	7
1024	1024^2	u	1	1	20	10
2048	1024^2	u	6	26	25	11
4096	1024^2	u	16	87	42	24

Table 4.1: Resulting values of our methods, for a non-varying set of followers.

n	U	CH	varying			
			leaders	report-all	max-length	max-size
128	512^2	c	2	161	41	23
256	512^2	c	12	389	71	71
512	512^2	c	13	520	68	238
1024	512^2	c	31	1142	66	208
2048	512^2	c	36	1022	72	419
4096	512^2	c	50	1541	78	959
128	1024^2	c	0	0	0	4
256	1024^2	c	0	0	2	8
512	1024^2	c	27	643	60	26
1024	1024^2	c	47	1591	73	183
2048	1024^2	c	97	2833	101	224
4096	1024^2	c	117	4299	78	324
128	512^2	u	0	0	0	4
256	512^2	u	0	0	13	7
512	512^2	u	6	54	41	13
1024	512^2	u	9	73	32	21
2048	512^2	u	29	187	40	25
4096	512^2	u	19	109	34	37
128	1024^2	u	0	0	0	3
256	1024^2	u	0	0	0	5
512	1024^2	u	0	0	0	7
1024	1024^2	u	1	2	20	10
2048	1024^2	u	20	152	40	17
4096	1024^2	u	49	279	42	24

Table 4.2: Resulting values of our methods, for a varying set of followers.

n	U	CH	without buckets		with buckets	
			non-varying	varying	non-varying	varying
128	512^2	c	9.44	9.56	2.35	2.86
256	512^2	c	40.91	41.89	12.58	14.69
512	512^2	c	203.53	212.88	102.74	119.15
1024	512^2	c	664.01	683.83	191.85	221.47
2048	512^2	c	3393.17	3457.26	972.34	1099.19
4096	512^2	c	14903.81	15046.69	5250.53	5651.59
128	1024^2	c	8.19	8.47	0.91	1.24
256	1024^2	c	32.79	33.82	2.83	3.63
512	1024^2	c	132.97	139.13	12.00	15.01
1024	1024^2	c	595.24	622.29	110.06	129.27
2048	1024^2	c	2809.12	2875.07	324.43	375.57
4096	1024^2	c	11143.28	11300.18	1477.70	1705.00
128	512^2	u	8.37	8.08	1.33	1.52
256	512^2	u	32.73	32.96	3.74	4.67
512	512^2	u	129.16	130.56	12.88	15.91
1024	512^2	u	529.79	523.12	47.54	54.52
2048	512^2	u	2184.42	2178.64	221.99	234.74
4096	512^2	u	11126.42	10978.71	1024.22	1037.39
128	1024^2	u	7.85	7.86	0.87	1.04
256	1024^2	u	31.45	32.79	2.28	3.01
512	1024^2	u	127.92	128.21	7.47	8.80
1024	1024^2	u	512.59	515.91	24.67	27.96
2048	1024^2	u	2268.77	2251.06	89.00	95.24
4096	1024^2	u	11201.63	11295.04	350.20	381.82

Table 4.3: Running times of our methods for the report-all problem. Reported times are in seconds.

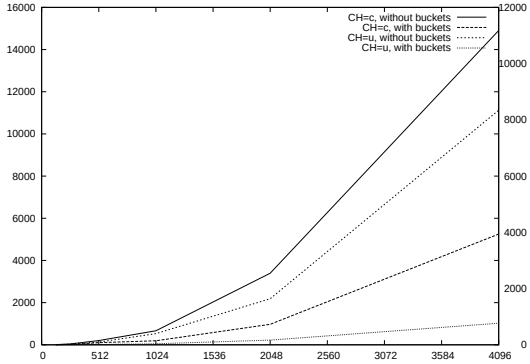

Figure 4.11: Running times depending on input size for non-varying report-all patterns for $U = 512^2$.

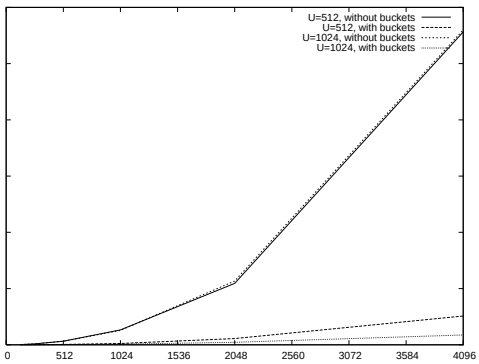

Figure 4.12: Running times depending on input size for non-varying report-all patterns for $CH = u$.

Without Buckets vs. With Buckets:

The approach to subdivide the space into buckets does not influence the reported values of our methods, however, it can have an impressive impact on the running times (see Figures 4.11 and 4.12). Depending on the input characteristics, we can observe speed-up factors between 2 and 32. The running time of the methods without 'buckets' is clearly quadratic in the number of entities. An asymptotic behaviour of the methods with 'buckets' is more difficult to identify, but note that also this method has a quadratic worst case running time.

$CH = u$ **vs.** $CH = c$**:**

Almost always the input files with characteristic $CH = c$ contain more patterns, longer patterns and larger patterns, which was expected as those files are much more likely to contain more and larger flocks. Hence, the input files with $CH = u$ result in smaller running times (see also Figure 4.11). Interestingly these characteristics also indicate that the 'bucket' approach for speeding-up the computations has its limitations, because the speed-up factor of the 'buckets' method is strongly influenced by the characteristics. For $CH = u$, we observe speed-up factors around between 5 and 11 for the instances with $U = 512^2$, and between 7 and 32 for the instances with $U = 1024^2$. On the other hand, for $CH = c$, the speed-up factors are

between 2 and 4 for the instances with $U = 512^2$, and between 5 and 11 for the instances with $U = 1024^2$. This can be explained by noting that the files with characteristic $CH = c$ contain more and bigger flocks, and hence it is more likely that our algorithms encounter neighboring buckets that are filled with more entities.

$U = 512^2$ vs. $U = 1024^2$:

The difference between the universe with $U = 512^2$ and $U = 1024^2$ is that the former is much denser when filled with the same number of entities. As a result, in the larger universe ($U = 1024^2$) less and smaller patterns exist. Also the running times are affected (see Figure 4.12). The methods with buckets run faster on instances with a larger universe, because we have more buckets and therefore, buckets are likely to contain less entities on average.

4.7 Discussion

The analysis of the interrelations of moving individuals has in the last five years attracted increasing attention, as a general reaction to the striking need for more powerful methods for surveillance and geospatial intelligence. Geographical information scientists are commissioned to develop methods that detect the expected and discover the unexpected from massive streams of disparate data, potentially originating from various sources [154]. Such methods need to be *scalable*, *flexible* and *reliable*. This section discusses our leadership approach with respect to these three properties and discusses the found algorithm running times.

Balancing the matching of formalized movement patterns (such as the presented leadership) with the inferring of unexpected space-time behaviors from visualized space-time paths, we argue that the former copes much better with increasing data sets. Whereas inferring form visualization might be adequate for the analysis of individual events of interest [98], keeping track of hundreds of individuals cruising in the space-time aquarium is literally impossible [109]. By contrast, when detecting movement patterns such as flock or leadership, the number of entities n is just a performance factor but not an obfuscation factor.

Approaches detecting leader and follower relationships using pair-wise cross-correlation of trajectories suffer from their intrinsic limitation to very small numbers of involved entities. Thus, lead-follow events in [147], for example, can only be detected for pairs of individuals at a time. Our leadership pattern, in contrast, allows individuals to lead groups of followers.

Since they operate on local-instantaneous events they can be detected in trajectories of variable lengths, as long as there is certain temporal overlap. Furthermore the approach in [147] has rather demanding constraints with respect to the analyzed data set. It requires trajectories of equal length and strongly synchronous sampling. Even though we assume the input data to have the same characteristics, our algorithms for the continuous case can be easily applied to data without a synchronized sampling. The running times for sorting the sets of intervals for an entity would slightly increase, however, from $O(n\tau \log n)$ to $O(n\tau \log n\tau)$. We argue that our leadership algorithms are thus flexible and applicable to diverse data from various sources.

Movement patterns that are defined from the geometric arrangement of the involved entities (e.g. leadership), are more reliable than movement patterns that base on the intermediate step of an analysis matrix, as do the REMO patterns depend on an analysis matrix in Laube et al. [112]. The deterministic discretisation of the movement descriptors in eight cardinal direction classes introduces edge effects. An example shall illustrate such edge effects. Let $22.5°$ be one threshold of the discrete movement azimuth class 'North'. Let furthermore the pattern under study be a flock pattern of four entities moving in the same direction at any time t. Why should a set of entities S_1 with azimuths $[21°, 22°, 22°, 21°]$ be a flock when another set S_2 with azimuths $[22°, 23°, 23°, 22°]$ is not? A definition requiring the entities to have a mean azimuth and some variance (e.g. $\pm 22.5°$) is a much more natural and thus reliable definition of flock. The definition of leadership in this chapter follows for exactly the same reasons the road of using a geometrical arrangement instead of scanning a discretised matrix.

When comparing the running times in this chapter with those reported in [23], we observe that the running times in the present work are much higher. This is because the used methods are different. The methods in [23] are faster but only report patterns of a specified length with a specified start- and end-time. The methods in this chapter, however, are more flexible. Once the arrays *IntervalsNotFwg* and *IntervalsFwg* are computed we can very efficiently use them to report patterns of different lengths, and with different start- and end-times. We also developed and implemented an approximation algorithm and performed initial experiments. They show a better asymptotic behaviour of the approximation algorithm. However, the constant factors seem to be too large for practical purposes, because for our test-files the exact algorithms always outperformed the approximation algorithm. More details on this algorithm can be found in [12].

4.8 Conclusions and Outlook

Movement patterns detect structure in large tracking data sets and are thus key to a better understanding of the interactions amongst moving agents. We provide a formal description of the pattern 'leadership' and subsequently algorithms for its efficient detection. 'Leadership' describes the event or process of one individual in front leading the movement of a group. Our approach is inspired by movement patterns documented in the animal behaviour and behavioural ecology literature.

Our experiments give indications which input-size can be processed within a reasonable amount of time, and they have shown that we are able to efficiently report leadership patterns. The resulting running times match the theoretical bounds, however for improved methods (with buckets) the running times strongly depend on the characteristics of the instance.

In this chapter we assumed that all the trajectories fit into main memory. If this is not the case then we would have to develop I/O-efficient algorithms or use spatio-temporal index structures. Both these techniques would probably improve performance if the input does not fit into main memory. However, this is an extension that would require much more future research.

One drawback of the given definition of leadership is that a leader has to be in the front region of all followers. For instance, for a very big flock of gnus this definition might not be applicable, as some gnus at the end of the flock are too far away from the front-line to be able to see leading animals. Hence, one direction for future research could be the definition and analysis of cascading leaders or followers, where a cascading follower is a follower of a leader or a follower of another cascading follower.

For the many fields interested in movement, the overall challenge lies in relating movement patterns with the surrounding environment, in order to understand *where*, *when* and ultimately *why* the agents move the way they do. Conceptualizing detectable movement patterns and the development of algorithms for their detection is a first important step towards this ambitious long-term goal. With its traditional spatial awareness, computational geometry can make immense contributions to the theoretical framework underlying movement analysis in geographical information science, behavioural ecology or surveillance and security analysis.

4.9 Acknowledgements

We thank Karin Schütz, AgResearch Ruakura, Hamilton, New Zealand for valuable commets on animal movement patterns, Bojan Djordjevic for implementing the algorithms and the anonymous reviewers of the article on which this chapter was based.

Part II

Computing Networks

Chapter 5

Balanced Partition of Minimum Spanning Trees

Scheduling problems [99] arise in a variety of settings. In general, scheduling problems involve m jobs that must be scheduled on $k \leq m$ machines subject to certain constraints while optimizing an objective function. In parallel computation [139], often the problem is to minimize the time complexity of the parallel algorithm.

In this chapter we consider a geometric version of these problems, namely given a geometric task divide it into k subtasks such that the size of the largest subtask is minimized. Such problems arise in applications from the shipbuilding industry [145]. The task is to cut out a set of prespecified regions from a sheet of metal while minimizing the completion time. Typically the size of the sheet is 10×30 meters and the number of regions that are to be cut out can vary from a few regions to several hundreds. In most cases there is only one single robot to handle this task but lately there are also examples where the number of robots is greater. In the case when there is just one robot the problem is closely related to the problem known in the literature as the *Traveling Salesperson problem with Neighborhoods* (TSPN) and it has been extensively studied [15, 45, 51, 77, 123, 127]. The problem asks for the shortest tour that visits all the regions, and it was recently shown to be APX-hard [45]. A variant of the Euclidean TSP when k robots are available was considered by Fredrickson *et al.* [69]. They showed that by computing a TSP-tour of the given point set and then partitioning the tour into k parts a $(2+\varepsilon-1/k)$-approximation could be obtained in the restricted case when the k robots must start and end at the same point. The need for partitioning the input set such that the optimal substructures are balanced

gives rise to many interesting theoretical problems. One such example is the so-called *Balanced Graph Partitioning* problem [13], where the aim is to partition the vertices of a graph into k subsets, while minimizing the capacity of the edges between the subsets. The partition has to be balanced in the sense that one subset can contain at most $\nu \cdot n/k$ vertices, for a given constant $\nu \geq 1$. In this chapter we consider the problem of partitioning the input so that the sizes of the minimum spanning trees of the subsets are balanced. More formally, the *k-Balanced Partition Minimum Spanning Tree problem* (k-BPMST) is stated as follows:

Problem 1 *(k-BPMST) Given a set of n points $\mathcal{S}$ in the plane, partition $\mathcal{S}$ into k sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the largest minimum spanning tree,*

$$W = \max_{1 \leq i \leq k} (|M(\mathcal{S}_i)|)$$

is minimized. Here $M(\mathcal{S}_i)$ is the minimum spanning tree of the subset $\mathcal{S}_i$ and $|M(\mathcal{S}_i)|$ is its weight.

Guttmann-Beck and Hassin studied a similar problem [90] for arbitrary metrics, where the aim was to minimize the total weight of all MST's of the partition, given a prespecified size $|S_i|$ for each subset S_i.

We also formulate the following problem below, the k-BPTSP problem. This is relevant since, given a c-approximation for the k-BPMST problem, we can easily achieve a $2c$-approximation for the k-BPTSP problem, by traversing the produced minimum spanning trees (MSTs).

Problem 2 *(k-BPTSP) Given a set of n points $\mathcal{S}$ in the plane, partition $\mathcal{S}$ into k sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the largest traveling salesperson tour,*

$$W = \max_{1 \leq i \leq k} (|TSP(\mathcal{S}_i)|)$$

is minimized. Here $TSP(\mathcal{S}_i)$ is the minimum traveling salesperson tour of the subset $\mathcal{S}_i$ and $|TSP(\mathcal{S}_i)|$ is its weight.

From the formal definitions of these problems it is clear that they can be classified as geometric k-clustering problems. As such they are very fundamental and have possible applications in a wide variety of settings such as for example statistics, image understanding, learning theory and computer graphics.

This chapter is organized as follows. We first show, in Section 5.1, that the k-BPMST problem is NP-hard. Next, in Section 5.2 we present a k-approximation greedy algorithm. This is followed by Section 5.3, where we

present an approximation algorithm for the problem with approximation factor $(4/3 + \varepsilon)$ for the case $k = 2$, and with approximation factor $(2 + \varepsilon)$ for the case when k is a fixed constant greater than 2. The algorithm runs in time $O(n \log n)$. Finally, in Section 5.4, conclusions and future research are presented.

5.1 NP hardness

In this section we show that the k-BPMST problem is NP-hard. In order to do this we need to state the recognition version of the k-BPMST problem:

Problem 3 *Given a set of n points $\mathcal{S}$ in the plane, and a real number $\mathcal{L}$, does there exist a partition of $\mathcal{S}$ into k sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the largest minimum spanning tree is bounded by $\mathcal{L}$, i.e., is it true that*

$$W = \max_{1 \leq i \leq k} (|M(\mathcal{S}_i)|) \leq \mathcal{L}?$$

The NP-hardness proof is done by a polynomial-time reduction from the following recognition version of PARTITION.

Problem 4 *(PARTITION) Given integers $A = \{a_1, \ldots, a_n\}$, such that $0 \leq a_1 \leq \ldots \leq a_n$, does there exist a subset $P \subseteq I = \{1, 2, \ldots, n\}$ such that*

$$|P| = h = n/2 \quad \textit{and} \quad \sum_{j \in P} a_j = \sum_{j \in I/P} a_j.$$

The above version of PARTITION is NP-hard [71].

Lemma 5.1 *The k-BPMST problem is NP-hard.*

Proof: The reduction is done as follows. Given an instance of PARTITION, we create an instance of 2-BPMST in polynomial time, such that it is a yes-instance if and only if the PARTITION-instance is a yes-instance. The input consists of n integers where we assume that n is an even number, if not, we just add a zero to the input. Given that the PARTITION-instance contains n integers $a_1, \ldots, a_n$ in sorted order, we create the following 2-BPMST instance. A set of points $\mathcal{S} = A' \cup L \cup R \cup L' \cup R$, as shown in figure 5.1(a) is created, with interpoint distances as shown in figure 5.1(b). The sets A', L, R, L' and R' are closer described below, where $\lambda = 11n(a_n + n)$ and $\delta = 7n(a_n + n)$.

- $A' = \{a'_1, \ldots, a'_n\}$, where $a'_i = (0, i\lambda)$,

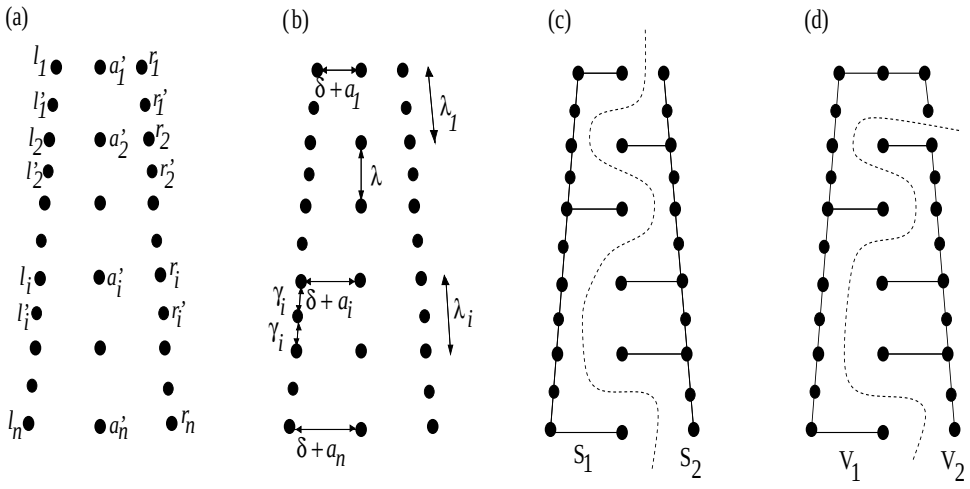

Figure 5.1: The set of points $\mathcal{S}$ created for the reduction. In Figure (a) all notations for the points are given. Similarly, in Figure (b) the notations for the distances between points are given. Figure (c) illustrates a class 1 partition, and (d) illustrates a class 2 partition.

- $L = \{l_1, \ldots, l_n\}$, where $l_i = (-\delta - a_i, i\lambda)$,
- $R = \{r_1, \ldots, r_n\}$, where $r_i = (\delta + a_i, i\lambda)$,
- $L' = \{l'_1, \ldots, l'_{n-1}\}$, where l'_i is the midpoint of the segment (l_i, l_{i+1}), and
- $R' = \{r'_1, \ldots, r'_{n-1}\}$, where r'_i is the midpoint of the segment (r_i, r_{i+1}).

For any given partition $P = \{P[1], P[2], \ldots, P[h]\} \subseteq \{1, 2, \ldots, n\}$, we define $A_P = \{a_{P[1]}, \ldots, a_{P[n]}\}$. Set $\lambda_i^2 = \delta^2 + \lambda^2$ which implies that $\lambda_i \leq 12n(a_n + n)$, which in turn gives that $\gamma_i = \lambda_i/2 \leq 6n(a_n + n)$. Finally let

$$\mathcal{L} = \frac{1}{2} \sum_{i \in I} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i.$$

Since the number of points in $\mathcal{S}$ is polynomial it is clear that this instance can be created in polynomial time. Next we consider the "if" and the "only if" parts of the proof separately.

If: If partition P exists and we have a yes-instance of PARTITION, then we will show that the corresponding 2-BPMST instance is also a yes-instance. This follows when the partition $\mathcal{S}'_1 = A_P \cup L \cup L'$, $\mathcal{S}'_2 = \mathcal{S} - \mathcal{S}_1$ (a

class 1 partition, as defined below) is considered. The general appearance of $M(\mathcal{S}'_1)$ and $M(\mathcal{S}'_2)$ is determined as follows. The set of points $L \cup L'$ and the set of points $r \cup R'$ will be connected as illustrated in figure 5.1(c), which follows from the fact that $\gamma_i < \delta < \delta + a_1$. Next consider the remaining points A'. Any point a'_i will be connected to either l_i (in $M(\mathcal{S}'_1)$) or r_i (in $M(\mathcal{S}'_2)$), since r_i and l_i are the points located closest to a'_i (follows since $\lambda > \delta + a_n$). Thus,

$$|M(\mathcal{S}'_1)| = |M(\mathcal{S}'_2)| = \frac{1}{2}\sum_{i \in I} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i$$

and we have that the created instance is a yes-instance.

Only if: We have that P does not exist and we therefore want to show that the created 2-BPMST is a no-instance. For this we examine two classes of partitions referred to as Class 1 and Class 2 partitions.

Class 1: All partitions $\{\mathcal{V}_1, \mathcal{V}_2\}$ such that $L \cup L' \subseteq \mathcal{V}_1$ and $R \cup R' \subseteq \mathcal{V}_2$

Class 2: All other partitions $\{\mathcal{U}_1, \mathcal{U}_2\}$ not belonging to Class 1.

We start by examining class 1 – see figure 5.1(c). A simple examination of the edges that will be picked by Kruskal's algorithm on the entire set S shows that an optimal MST for S will contain the edges connecting the l_is and the l'_is, and also the edges connecting the r_is and the r'_is. Also, for each point $a'_i, i > 1$, it will contain an edge connecting it to either l_i or to r_i, Finally, for point a_1, it will contain edges connecting it to both l_1 and r_1. So clearly, $|M(S)| = 2 \cdot \mathcal{L} + \delta + a_1$. Its longest edge is of length $\delta + a_1$. Therefore, $|M(\mathcal{V}_1)| + |M(\mathcal{V}_2)| \geq 2 \cdot \mathcal{L}$. Consequently, $max\{|M(\mathcal{V}_1)|, |M(\mathcal{V}_2)|\} \geq \mathcal{L}$.

Let P_1 and P_2 be the subset of points from A' that are in $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively. If $|P_1| = |P_2| = n/2$, then clearly for $j = 1, 2$, we have:

$$|M(\mathcal{V}_j)| = \sum_{i \in P_j} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i.$$

Since we have assumed that no solution to the instance of PARTITION exists, it is clear that $|M(\mathcal{V}_1)| \neq |M(\mathcal{V}_2)|$, implying that $max\{|M(\mathcal{V}'_1)|, |M(\mathcal{V}'_2)|\} > \mathcal{L}$, and that the instance of 2-BPMST is a no-instance.

If, on the other hand, $|P_1| \neq |P_2|$, then w.l.o.g., we assume that $|P_1| > n/2$. Then we know that

$$|M(\mathcal{V}_1)| \geq \delta + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i > \sum_{i \in I} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i > \mathcal{L},$$

again proving that the instance of 2-BPMST is a no-instance.

Next consider the class 2 partitions, illustrated in figure 5.1(d). There is always an edge of weight γ_i $(1 \leq i \leq n)$ connecting the two point sets of any such partition. This means that there can not exist a class 2 partition $\mathcal{U}_1, \mathcal{U}_2$ such that $\max\{|M(\mathcal{U}_1)|, |M(\mathcal{U}_2)|\} \leq \mathcal{L}$, because we could then build a tree with weight at most $2 \cdot \mathcal{L} + \gamma_i < |M(\mathcal{V}_1)| + |M(\mathcal{V}_2)| + \delta + a_1 = |M(\mathcal{S})|$, which is a contradiction. Thus, $\max\{|M(\mathcal{U}_1)|, |M(\mathcal{U}_2)|\} > \mathcal{L}$, which concludes this lemma. □

Note that the problems of computing square roots can be avoided by adding vertices, one vertex at each place where there may be a non-isothetic edge in our construction, in order to ensure that this MST edge is replaced by two isothetic edges.

5.2 Greedy algorithm

In this section we present a straight-forward greedy algorithm with approximation factor k. The basic idea of this algorithm is to remove the $k-1$ heaviest edges of $M(\mathcal{S})$. Let $w_1, \ldots, w_{k-1}$ denote the weights of these $k-1$ edges. The components created $M(\mathcal{S}_1), \ldots, M(\mathcal{S}_k)$ are MST's for the points that they span. Further, these MST's define a partition of $\mathcal{S}$ into k subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$.

Next an upper bound on the approximation factor for the greedy algorithm is shown. For this the following lemma is needed:

Lemma 5.2 *The optimal solution of the k-*BPMST *problem is bounded from below by*

$$\frac{1}{k}(|M(S)| - \sum_{i=1}^{k-1} w_i).$$

Proof: This is shown by contradiction. Assume that there exists an optimal solution $\mathcal{S}_1, \ldots, \mathcal{S}_k$ for $\mathcal{S}$ where

$$|opt| < \frac{1}{k}(|M(\mathcal{S})| - \sum_{i=1}^{k-1} w_i)$$

Let $\mathcal{C}$ denote the set of components $M(\mathcal{S}_1), \ldots, M(\mathcal{S}_k)$. The first step in realizing the contradiction is to build a tree by connecting the components of $\mathcal{C}$ using $k-1$ unique edges from $M(\mathcal{S})$. Such a tree $\mathcal{T}$ can always be built. Start with a forest $\mathcal{F}$ containing the components of $\mathcal{C}$ and then consider an arbitrary cut of $\mathcal{S}$ that respects the edges of $\mathcal{F}$. Since $M(\mathcal{S})$ is a tree there

must be at least one edge in $M(\mathcal{S})$ that crosses this cut. Add one such edge to $\mathcal{F}$. This edge connects two of the components in the forest, thus creating a larger component. Continue in this manner; consider cuts that do not cross any edge in $\mathcal{F}$ and add a crossing edge from $M(\mathcal{S})$ to $\mathcal{F}$. Eventually all components in $\mathcal{F}$ will be connected and the spanning tree $\mathcal{T}$ will have been built.

The contradiction is then seen if we consider the total weight $|\mathcal{T}|$ of this tree. The weight is equal to the sum of the components of $\mathcal{C}$ plus the sum of the added edges. First consider the total weight of $\mathcal{C}$. Since

$$|opt| = \max_{i=1}^{k} |M(\mathcal{S}_{opt_i})| < \frac{1}{k}(M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i)$$

we have that

$$|M(\mathcal{S}_{opt_i})| < \frac{1}{k}(M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i) \quad for\ all\ i,$$

and that

$$|\mathcal{C}| = \sum_{i=1}^{k} |M(\mathcal{S}_{opt_i})| < M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i$$

Further, the weight of the $k-1$ added edges from $M(\mathcal{S})$ is less than or equal to the weight of the $k-1$ heaviest edges of $M(\mathcal{S})$. This means that

$$|\mathcal{T}| < |\mathcal{C}| + \sum_{i=1}^{k-1} w_i < \left(M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i \right) + \sum_{i=1}^{k-1} w_i = M(S)$$

which is a contradiction. □

Lemma 5.3 *The greedy algorithm is an approximation algorithm with an upper ratio bound of k for the k-*Bpmst *problem.*

Proof: Since the greedy algorithm simply removes the $k-1$ heaviest edges of $M(\mathcal{S})$, the total weight of *all* edges in the solution is $M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i$. In the worst case the algorithm yields a partition with the total weight collected in one MST of one of the subsets. Therefore $|greedy| \leq M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i$. Further, from Lemma 5.2 we have that

$$|opt| \geq \frac{1}{k}(M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i)$$

This means that

$$\frac{|greedy|}{|opt|} \leq \frac{M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i}{\frac{1}{k}(M(\mathcal{S}) - \sum_{i=1}^{k-1} w_i)} \leq k$$

□

Next consider the complexity of the algorithm. First $M(\mathcal{S})$ needs to be constructed, which can be done in time $O(n \log n)$. Next the $k-1$ heaviest edges of $M(\mathcal{S})$ are marked, but not removed, since their presence in $M(\mathcal{S})$ is needed when determining $\mathcal{S}_1, \ldots, \mathcal{S}_k$ (see below). In order to mark the edges they are sorted. We have that $|E| = n - 1$ in a MST which means that the sorting takes $O(n \log n)$ time. Finally $\mathcal{S}_1, \ldots, \mathcal{S}_k$ are determined with k breadth-first searches. The searches are done with the adjacent nodes of the $k-1$ heaviest edges as source nodes. Before the searches begin, a vector $\mathcal{V}$ is created, which represents these adjacent nodes. An element i set to true means that node i will be used as a source node in a search. The algorithm, then, is to iterate this vector and perform a search for each element i set to true, with its corresponding node i used as a source node. The breadth-first search is modified to consider the marked edges as non-existing so that only one component is examined in each search. Further, when a node adjacent to a marked edge is found during a search its corresponding element in $\mathcal{V}$ is set to false. This reduces the number of possible searches of each component from two to one. The result of each search is a subset $\mathcal{S}_i$ and after k searches all subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ are determined. The total running time of the k breadth-first searches is the same as one breadth first search on $M(\mathcal{S})$, $O(|\mathcal{E}| + n) = O(n)$. This is due to the fact that the same number of nodes and edges, in total, are scanned in both cases. Thus, the total running time of the greedy algorithm is $O(n \log n)$.

The following theorem concludes the results of this section.

Theorem 5.4 *The greedy algorithm produces a partition of weight at most k times the optimal in time $O(n \log n)$.*

5.3 Approximating the k-BPMST

In this section a $(2+\varepsilon)$-approximation algorithm is presented. The main idea is to partition $\mathcal{S}$ into a constant number of small components, test all valid

combinations of these components and, finally, output the best combination. In order to do this efficiently, as will be seen later, one will need an efficient partitioning algorithm.

A partition of a point set $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ is said to be *valid* if $\max\{|M(\mathcal{S}_1)|, |M(\mathcal{S}_2)|\} \leq \frac{2}{3} \cdot |M(\mathcal{S})|$ and $|M(\mathcal{S}_1)| + |M(\mathcal{S}_2)| \leq |M(\mathcal{S})|$. The partition is denoted by the collection $\{\mathcal{S}_1, \mathcal{S}_2\}$. It is known that a valid partition always exists and can be computed efficiently [21]. For completeness we provide a detailed description of this algorithm.

5.3.1 ValidPartition

In this section we describe an algorithm, denoted ValidPartition or VP for short; given a set of points $\mathcal{S}$, VP computes a valid partition. First the algorithm is described and then it is shown that it outputs a valid partition. We need the following definition.

Definition 5.5 A point q is said to be a "hub" of $M(\mathcal{S})$ if and only if:

1. It has at least four (at most six) incident edges $e_1 = (q, v_1), \ldots e_s = (q, v_r)$ (in clockwise-order).
2. Every subtree of $M(\mathcal{S})$ that has q as a leaf has weight less than $1/3 \cdot M(\mathcal{S})$, see Fig. 5.2.

Note that a spanning tree has at most one hub. We need the following notations. Let $M(\mathcal{T}_1), \ldots, M(\mathcal{T}_r)$ be the r maximal subtrees of $M(\mathcal{S})$, in clockwise order, that have q as a leaf (one for each incident edge of q), and let $\mathcal{T}_i$ be the set of points included in $M(\mathcal{T}_i)$. Finally, let $\mathcal{T}_i' = \mathcal{T}_i - \{q\}$. As mentioned earlier, a partition of a point set $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ is said to be valid if $\max\{|M(\mathcal{S}_1)|, |M(\mathcal{S}_2)|\} \leq 2/3 \cdot |M(\mathcal{S})|$.

Now, consider the following straight-forward algorithm, ValidPartition, for partitioning a set of points $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$. Select an arbitrary leaf ν of $M(\mathcal{S})$ as a starting point, and set $\mathcal{S}_1 = \{\nu\}$ and $\mathcal{S}_2 = \mathcal{S} - \{\nu\}$. During the whole process, vertices are added to $\mathcal{S}_1$ and deleted from $\mathcal{S}_2$, and both sets correspond to connected sets of vertices in $M(\mathcal{S})$. While $\{\mathcal{S}_1, \mathcal{S}_2\}$ is not a valid partition, expand $\mathcal{S}_1$ with points of $\mathcal{S}$ by following the tree in such a way that $M(\mathcal{S}_1) \subset M(\mathcal{S})$, $M(\mathcal{S} - \mathcal{S}_1) \subset M(\mathcal{S})$, and the weight of $M(\mathcal{S}_1)$ minimally increases in each iterative expansion. The algorithm will terminate after $O(n)$ steps. It is clear that the algorithm will either find a valid partition with $M(\mathcal{S}_1), M(\mathcal{S}_2) \subset M(\mathcal{S})$, or a "hub" of $M(\mathcal{S})$ would have been reached without finding a valid partition.

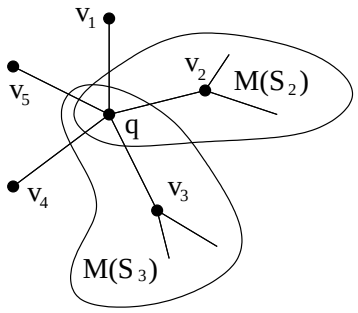

Figure 5.2: A possible hub q with five incident edges.

If, upon termination, $\mathcal{S}_1$ and $\mathcal{S}_2$ is not a valid partition then ValidPartition combines the $r+1$ trees $\{q, M(\mathcal{T}_1'), \ldots, M(\mathcal{T}_r')\}$ into an "optimal" partition, $\mathcal{S}_1$ and $\mathcal{S}_2$, by adding $r-1$ edges. Note that since a minimum spanning tree for points in the plane has maximum degree 6, $r+1 \leq 7$, which makes it possible for the above "optimal" partitioning of the $r+1$ trees to be done in constant time.

The following lemma can now be shown:

Lemma 5.6 *Given a set of points $\mathcal{S}$,* ValidPartition *computes a valid partition, $\mathcal{S}_1$ and $\mathcal{S}_2$, of $\mathcal{S}$.*

Proof: If a hub was not located by ValidPartition, then the lemma is clearly true. If a hub q was located, then we know that

$$\max\{|M(\mathcal{T}_1)|, \ldots, |M(\mathcal{T}_r)|\} < 1/3 \cdot |M(\mathcal{S})|$$

and $r \geq 4$ (otherwise we would have a valid partition). Hence, it follows that there exists an $r' < r-1$ such that $M(\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_{r'})$ has weight between $1/3 \cdot |M(\mathcal{S})|$ and $2/3 \cdot |M(\mathcal{S})|$. We will have two cases (the other cases cannot occur):

$r' = 2$ **or** $r - r' = 2$**:** Assume for simplicity that $r' = 2$, then a valid partition is $\mathcal{S}_1 = \mathcal{T}_1' \cup \mathcal{T}_2'$ and $\mathcal{S}_2 = \mathcal{S} - \mathcal{S}_1 = \mathcal{T}_3 \cup \ldots \mathcal{T}_r$. We know that $|M(\mathcal{S}_2)| < 2/3 \cdot |M(\mathcal{S})|$ since $|M(\mathcal{T}_1)| + |M(\mathcal{T}_2)| \geq 1/3 \cdot |M(\mathcal{S})|$. Now, since the shortest edge connecting $M(\mathcal{T}_1')$ with $M(\mathcal{T}_2')$ is obviously shorter than $|e_1| + |e_2|$ it also holds that $|M(\mathcal{S}_1)| < 2/3 \cdot |M(\mathcal{S})|$. Note that in this case $|M(\mathcal{S}_1)| + |M(\mathcal{S}_2)| \leq |M(\mathcal{S})|$.

$r' = 3$ **and** $r = 6$**:** We have that $|M(\mathcal{T}_4 \cup \mathcal{T}_5 \cup \mathcal{T}_6)| = M(\mathcal{S}) - |M(\mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3)| \leq \frac{2}{3}|M(\mathcal{S})|$. Further, it can be shown (see below) that $|M(\mathcal{T}_4' \cup$

$\mathcal{T}'_5 \cup \mathcal{T}'_6)| \leq |M(\mathcal{T}_4 \cup \mathcal{T}_5 \cup \mathcal{T}_6)|$, which means that $\mathcal{S}_1 = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$ and $\mathcal{S}_2 = \mathcal{T}'_4 \cup \mathcal{T}'_5 \cup \mathcal{T}'_6$ is a valid partition. Also, $|M(\mathcal{T}'_4 \cup \mathcal{T}'_5 \cup \mathcal{T}'_6)| \leq |M(\mathcal{T}_4 \cup \mathcal{T}_5 \cup \mathcal{T}_6)|$, because if we consider the edges $e_4 = (q, v_4), e_5 = (q, v_5)$ and $e_6 = (q, v_6)$. The angle between $e_i = (q, v_i)$ and $e_{i+1} = (q, v_{i+1})$ is 60° since $M(\mathcal{S})$ is a Euclidean minimum spanning tree. It holds that $M(\mathcal{T}'_4)$, $M(\mathcal{T}'_5)$ and $M(\mathcal{T}'_6)$ can be connected by two segments of total length less than $|e_4| + |e_5| + |e_6|$.

Thus the lemma follows. □

If a minimum spanning tree of $\mathcal{S}$ is given as input then it is easy to see that the time needed for VP to compute a valid partition is $O(n)$.

5.3.2 Repeated ValidPartition

One of the main ideas in the approximation algorithms (presented in Section 5.3.3) is to repeatedly use VP in order to create the many small components. The algorithm constructing these components is therefore called RepeatedValidPartition, or RVP for short.

RVP is described as follows: Given $M(\mathcal{S})$ and an integer m, first partitions $\mathcal{S}$ into two components using VP. Then repeatedly partition the largest component created thus far, again using VP, until m components have been created.

The following lemma describes an important property of RVP.

Lemma 5.7 *Given a minimum spanning tree of a set of points $\mathcal{S}$ and an integer m,* RVP *will partition $\mathcal{S}$ into m components $\mathcal{S}_1, \ldots, \mathcal{S}_m$ such that*

$$\max\{|M(\mathcal{S}_1)|, \ldots, |M(\mathcal{S}_m)|\} \leq \frac{2}{m}|M(\mathcal{S})|.$$

Proof: Consider the following alternative algorithm $\mathcal{A}$, which is similar to RVP. Given $M(\mathcal{S})$, algorithm $\mathcal{A}$ uses VP to divide $M(\mathcal{S})$ until all resulting components weigh at most $\frac{2}{m}|M(\mathcal{S})|$. The order in which the components are divided is arbitrary but when a component weighs at most $\frac{2}{m}|M(\mathcal{S})|$ it is not divided any further. Compare algorithm $\mathcal{A}$ with RVP, which always applies VP to the largest component and stops as soon as m components are computed. A component of weight at most $\frac{2}{m}|M(\mathcal{S})|$ will not be divided by RVP unless all other components created thus far also weigh at most $\frac{2}{m}|M(\mathcal{S})|$.

Now, if the number of resulting components of $\mathcal{A}$ is at most m then the lemma would follow. This is because RVP will first create the same

components as $\mathcal{A}$ and then possibly divide these components further. Since these additional divisions are performed using VP we have that the resulting m components obviously will weigh at most $\frac{2}{m}|M(\mathcal{S})|$.

The process of $\mathcal{A}$ can be represented as a tree. In this tree each node represents a subset of $\mathcal{S}$ or a subtree of $M(\mathcal{S})$ on which VP is applied. The root represents $M(\mathcal{S})$, and the children of a node represent the components created when that node is divided using VP. We use the notation $M(v)$ to denote the subtree of $M(\mathcal{S})$ associated with node v. Note that the leaves of this tree represent the output components created by $\mathcal{A}$. We divide these leaves into two categories, where the first category consists of all leaves whose sibling is not a leaf and the second consists of all remaining leaves (that is, those whose siblings are also leaves). We let m_1 and m_2 denote the number of leaves of the first and second category correspondingly.

We start by examining the leaves of the first category, denoted $L = \{l_1, \ldots, l_{m_1}\}$. Consider any leaf $l \in L$, its sibling s, and its parent p. To each l we attach a weight $w(l)$ which is defined as $w(l) = |M(p)| - |M(s)|$. Since s is not a leaf it holds that $|M(s)| > \frac{2}{m}|M(\mathcal{S})|$, and since VP was applied we know that $|M(s)| \leq \frac{2}{3}|M(p)|$. Thus $|M(p)| > \frac{3}{m}|M(\mathcal{S})|$, which implies that $w(l) \geq \frac{1}{3}|M(p)| > \frac{1}{m}|M(\mathcal{S})|$ and, hence, $\sum_{i=1}^{m_1} w(l) > m_1 \cdot \frac{1}{m}|M(\mathcal{S})|$.

Next the second category of leaves, denoted $L' = \{l'_1, \ldots, l'_{m_1}\}$, is examined. Consider any such leaf l' and its corresponding parent p'. Since there are m_2 leaves of this category and each leaf has a leaf sibling, these leaves have a total of $m_2/2$ parent nodes. Furthermore, for each of the $m_2/2$ parent nodes, p', we have that $|M(p')| > \frac{2}{m}|M(\mathcal{S})|$, since they are not leaves. Thus, $\sum_{p' \in L'} |M(p')| > \frac{m_2}{2} \cdot \frac{2}{m}|M(\mathcal{S})| = m_2 \cdot \frac{1}{m}|M(\mathcal{S})|$.

Finally, consider the total weight of the components examined. We have that $m_1 \cdot \frac{1}{m}|M(\mathcal{S})| + m_2 \cdot \frac{1}{m}|M(\mathcal{S})| < \sum_{i=1}^{m_1} w(l) + \sum_{p' \in L'} |M(p')| \leq |M(\mathcal{S})|$, which implies that $m_1 + m_2 < m$. Thus, the number of leaves does not exceed m, and the lemma follows. □

5.3.3 The approximation algorithm

Now we are ready to present the approximation algorithm, which we will denote CA. As input we are given a set $\mathcal{S}$ of n points, a fixed integer k and a positive real constant ε. The algorithm considers two cases: $k = 2$ and $k \geq 3$. First the case $k = 2$ is examined.

Case: $[k = 2]$

step 1: Divide $M(\mathcal{S})$ into $\frac{2}{\varepsilon'}$ components, using RVP, where $\varepsilon' = \frac{\varepsilon}{4/3+\varepsilon}$.

The reason for the value of ε' will become clear below. Let w denote the weight of the heaviest component created.

step 2: Combine all components created in step 1, in all possible ways, into two groups.

step 3: For each combination generated in step 2, compute the MST for each of its two created groups.

step 4: Output the pair of MSTs with the least maximum weight.

Theorem 5.8 *For $k = 2$, the approximation algorithm* CA *has a time complexity of $O(n \log n)$, and produces a partition whose total weight is within a factor $(\frac{4}{3} + \varepsilon)$ of the optimal partition.*

Proof: Let $\{V_1, V_2\}$ be the partition obtained from CA. Assume that $\mathcal{S}_1$ and $\mathcal{S}_2$ is the optimal partition, and let e be the shortest edge connecting $\mathcal{S}_1$ with $\mathcal{S}_2$. In the following discussions, we assume that the weight of the output of CA is denoted by $|CA|$ and the weight of an optimal solution is denoted by $|opt|$. According to Lemma 3 it follows that $w \leq \frac{2}{2/\varepsilon'}|M(\mathcal{S})| = \varepsilon'|M(\mathcal{S})|$. We have two cases, $|e| > w$, and $|e| \leq w$, which are illustrated in figure 5.3(a) and 5.3(b), respectively.

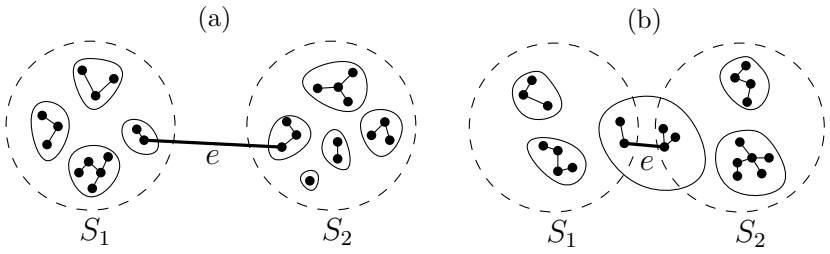

Figure 5.3: The two cases for CA, $k = 2$. The edge e (marked) is the shortest edge connecting $\mathcal{S}_1$ with $\mathcal{S}_2$.

In the first case every component is a subset of either $\mathcal{S}_1$ or $\mathcal{S}_2$. This follows since a component consisting of points from both $\mathcal{S}_1$ and $\mathcal{S}_2$ must include an edge with weight greater than w. Thus, no such component can exist among the components created in step 1. Further, this means that the partition $\mathcal{S}_1$ and $\mathcal{S}_2$ must have been tested in step 2 of CA and, hence, the optimal solution must have been found.

In the second case, $|e| \leq w$, there may exist components consisting of points from both $\mathcal{S}_1$ and $\mathcal{S}_2$, see Fig. 5.3. To determine an upper bound

of the approximation factor we start by examining an upper bound on the weight of the solution produced by CA. The dividing process in step 1 of CA starts with $M(\mathcal{S})$ being divided into two components $M(\mathcal{S}'_1)$ and $M(S'_2)$, such that $\max\{|M(\mathcal{S}'_1)|, |M(\mathcal{S}'_2)|\} \leq \frac{2}{3}|M(\mathcal{S})|$. These two components are then divided into several smaller components. This immediately reveals an upper bound for $|CA|$ of $\frac{2}{3}|M(S)|$. Next the lower bound is examined. We have:

$$|opt| \geq \frac{|M(\mathcal{S})| - |e|}{2} \geq \frac{|M(\mathcal{S})|}{2} - \frac{\varepsilon' M(\mathcal{S})}{2} = (1 - \varepsilon')\frac{M(\mathcal{S})}{2}.$$

Then, if the upper and lower bounds are combined we get:

$$|CA|/|opt| \leq \frac{\frac{2}{3}|M(\mathcal{S})|}{(1-\varepsilon')\frac{M(\mathcal{S})}{2}} \leq \frac{4/3}{1-\varepsilon'} \leq 4/3 + \varepsilon.$$

In the third inequality we used the fact that $\varepsilon' \leq \frac{\varepsilon}{4/3+\varepsilon}$.

Next consider the complexity of CA. In step 1 $M(\mathcal{S})$ is divided into a constant number of components using RVP. This takes $O(n)$ time, according to Lemma 3. Then, in step 2, these components are combined in all possible ways, which takes constant time since there are a constant number of components. For each tested combination there is a constant number of MST's to be computed in step 3. Further, since there are a constant number of combinations and $M(\mathcal{S})$ takes $O(n \log n)$ to compute, step 3 takes $O(n \log n)$ time. □

Next we consider the case $k \geq 3$. In this case the following steps are performed:

Case: [$k \geq 3$]

step 1: Compute $M(\mathcal{S})$ and remove the $k-1$ heaviest edges $e_1, \ldots, e_{k-1}$ of $M(\mathcal{S})$, thus resulting in k separate trees $M(U'_1), \ldots, M(U'_k)$.

step 2: Divide every tree $M(U'_i)$, $1 \leq i \leq k'$, into $\frac{4k}{\varepsilon'}$ components, using RVP. Set $\varepsilon' = \frac{\varepsilon}{2+\varepsilon}$. The reason for the value of ε' will become clear below. Denote the resulting components $M(U_1), \ldots, M(U_r)$, where $r = \frac{4k}{\varepsilon'} \cdot k$. Also, let $w = \max\{|M(U_1)|, \ldots, |M(U_r)|\}$.

step 3: Combine $U_1, \ldots, U_r$ in all possible ways into k groups.

step 4: For each such combination do:

- Compute the MST for each group.

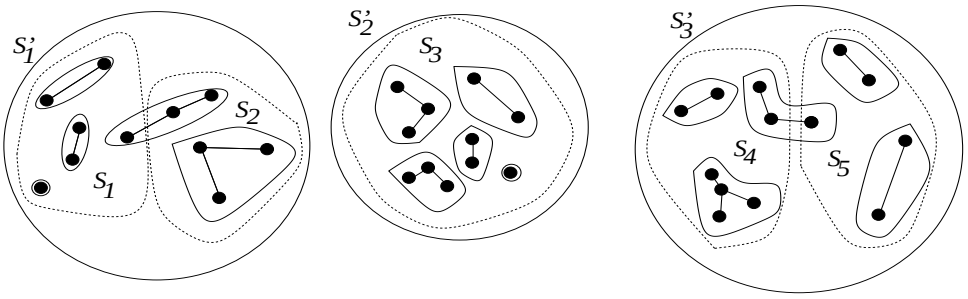

Figure 5.4: $\mathcal{S}_1, \ldots, \mathcal{S}_k$ is an optimal partition of $\mathcal{S}$. All subsets that can be connected by edges of length at most w are merged, thus creating the new set $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$.

- Divide each MST in all possible ways, using RVP. That is, each MST is divided into $1, \ldots, i$, where $i \leq k$, components, such that the total number of components resulting from all the divided MST's equals k. Each such division defines a partition of $\mathcal{S}$ into k subsets.

step 5: Of all the tested partitions in step 4, output the best.

Theorem 5.9 *For fixed k greater than 2 the approximation algorithm CA produces a partition which is within a factor of $(2 + \varepsilon)$ of the optimal in time $O(n \log n)$.*

Proof: A constant number of components are created which means that the time complexity is the same as for the case when $k = 2$, that is $O(n \log n)$. To prove the approximation factor we first give an upper bound on the weight of the solution produced by CA and then we provide a lower bound for an optimal solution. Combining the two results will conclude the proof.

Consider an optimal partition of $\mathcal{S}$ into k subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$. Merge all subsets that can be connected by edges of length at most w. From this we obtain the sets $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$, where $k' \leq k$ as illustrated in figure 5.4. Let m'_i denote the number of elements from $\mathcal{S}_1, \ldots, \mathcal{S}_k$ included in S'_i. The purpose of studying these new sets is that every component created in step 2 of CA belongs to exactly one element in $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$. A direct consequence of this is that every possible partition of $\{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ into k' groups must have been tested in step 4.

Step 4 guarantees that $M(\mathcal{S}'_1), \ldots, M(\mathcal{S}'_{k'})$ will be calculated, and that these MSTs will be divided in all possible ways. Thus, a partition will be

made such that each $M(\mathcal{S}'_i)$ will be divided into exactly m'_i components. This partitions $\mathcal{S}$ into k subsets $\mathcal{V}_1, \ldots, \mathcal{V}_k$. Let $\mathcal{V}$ be a set in $\mathcal{V}_1, \ldots, \mathcal{V}_k$ such that $|M(\mathcal{V})| = \max_{1 \leq i \leq k}\{|M(\mathcal{V}_i)|\}$. We wish to restrict our attention to exactly one element of the set $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$, hence we note that $\mathcal{V}$ is a subset of exactly one element $\mathcal{S}'$ in $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$. Assume that $M(\mathcal{V})$ was created in step 4 of the algorithm, when $M(\mathcal{S}')$ was divided into m' components using RVP, then it holds that $M(\mathcal{V}) \leq \frac{2}{m'}|M(\mathcal{S}')|$, according to Lemma 3. Since the partition $\mathcal{V}_1, \ldots, \mathcal{V}_k$ will be tested by CA we have that $|CA| \leq |M(\mathcal{V})| \leq \frac{2}{m'}|M(\mathcal{S}')|$.

Next a lower bound of an optimal solution is examined. Let $|opt'|$ be the value of an optimal solution for $\mathcal{S}'$ partitioned into m' subsets. Note that $\mathcal{S}'$ consists of m' elements from $\mathcal{S}_1, \ldots, \mathcal{S}_k$. Assume w.l.o.g. that $\#\mathcal{S}' = \#\mathcal{S}_1 + \ldots + \#\mathcal{S}_{m'}$. This means that $\mathcal{S}_1, \ldots, \mathcal{S}_{m'}$ is a possible partition of $\mathcal{S}'$ into m' subsets. Thus, $|opt| \geq \max_{1 \leq i \leq m'}\{|M(S_i)|\} = |opt'|$. Assume w.l.o.g. that $e'_1, \ldots, e'_{m'-1}$ are the edges in $M(\mathcal{S})$ connecting the components in $\mathcal{S}'$. We have:

$$\begin{aligned} |opt| \geq |opt'| &\geq \frac{1}{m'}(|M(S')| - \sum_{i=1}^{m'-1} |e'_i|) \\ &\geq \frac{1}{m'}(|M(S')| - (m'-1)w) \end{aligned} \tag{5.1}$$

To obtain a useful bound we need an upper bound on w. Consider the situation after step 1 has been performed. We have $\max_{1 \leq i \leq k}(|M(U'_i)|) \leq |M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|$. Since each U'_i is divided into $\frac{4k}{\varepsilon'}$ components we have that the resulting components, and therefore also w, have weight at most $1/(\frac{2k}{\varepsilon'}) \cdot (|M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|)$, according to Lemma 3. Using the above bound gives us:

$$\frac{w}{|opt|} \leq \frac{1/(\frac{2k}{\varepsilon'}) \cdot (|M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|)}{\frac{1}{k}(|M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|)} \leq \frac{\varepsilon'}{2} \quad \Longrightarrow \quad w \leq \frac{\varepsilon'}{2}|opt| \tag{5.2}$$

Note that $|opt| \leq |M(\mathcal{V})| \leq \frac{2}{m'}|M(\mathcal{S}')|$. Further, by combining (5.1) and (5.2) gives us:

$$|opt| \geq \frac{1}{m'}\left(|M(S')| - (m'-1)\frac{\varepsilon'}{2}|opt|\right) \geq (1-\varepsilon')\frac{|M(\mathcal{S}')|}{m'}.$$

Combining the two bounds together with the fact that $\varepsilon' \leq \varepsilon/(2+\varepsilon)$ concludes the proof of the theorem.

$$|CA|/|opt| \leq \frac{\frac{2}{m'}|M(\mathcal{S}')|}{(1-\varepsilon')\frac{|M(\mathcal{S}')|}{m'}} \leq \frac{2}{1-\varepsilon'} \leq 2+\varepsilon.$$

□

5.4 Conclusion and further research

In this chapter it was first shown that the k-BPMST problem is NP-hard. After this was established, the next step was to design approximation algorithms for the problem. First a greedy k-approximation algorithm was shown. Next an algorithm based on partitioning the point set into a constant number of smaller components and then trying all possible combinations of these small components was shown. This approach revealed a $(4/3 + \varepsilon)$-approximation in the case $k = 2$, and a $(2 + \varepsilon)$-approximation in the case when we are given a fixed $k \geq 3$. The time complexity of the algorithm is $O(n \log n)$. However, the running time is exponential with respect to $1/\varepsilon$ and factorial with respect to k. Hence, one of the main research questions is to find a faster algorithm with respect to $1/\varepsilon$ and k.

For several generalizations of the k-BPMST problem it is straightforward to see that the results of this chapter are immediately valid once a valid partition can be guaranteed. This is true, for example, in a higher-dimensional geometric setting. For other settings, such as metric graphs, it is obvious that we can't always guarantee a valid partition, since we have to consider non-complete graphs. In this case, however, the results of this chapter are valid if we allow vertices to be included in more than one of the output subsets.

An interesting question to consider is if Lemma 2 is tight, i.e., is it possible to improve the upper bound of 2/3? If so, the approximation factor for the case when $k = 2$ would be improved and maybe also the result for larger values of k.

Chapter 6

Extending the Gap Theorem

The Gap property was introduced by Chandra et al. [30] in 1995. Let $G = (V, E)$ be a directed graph on a set of points in $\mathcal{R}^d$. The edge set E satisfies the so-called w-gap property if for any two distinct edges (p, q) and (r, s) in E, we have that the sources are at least a distance $w \cdot min(|pq|, |rs|)$ apart.

Chandra et al. [30] also proved the Gap theorem that states that any directed edge set fulfilling the Gap property has weight $O(\frac{1}{w} \cdot \log n \cdot wt(MST(V))$, where $wt(MST(V))$ denoted the weight of a minimum spanning tree of the point set.

Note that if a graph has an edge set satisfying the w-gap property then the outdegree of each vertex is at most one. In this chapter we propose a relaxed version of the Gap property and show a similar weight bound. An edge set E satisfies the relaxed w-gap property if for any two distinct edges (p, q) and (r, s) in E, if the sources are closer than $w \cdot min(|pq|, |rs|)$ apart then their sinks are at least $w \cdot min(|pq|, |rs|)$ apart (the definitions are given in Section 2). We also prove that any edge set fulfilling the relaxed Gap property has weight $O(2^d \log n \cdot wt(MST(V)))$.

We believe that the extended result can be applied to a wider range of geometric graphs. As an example we use it to develop a very simple algorithm that constructs a sparse low weight t-spanner. Given a set V of n points in $\mathbb{R}^d$ and a real number $t > 1$, a *t-spanner* for V is a graph G with vertex set V such that any two vertices u and v are connected by a path in G whose length is at most $t \cdot |uv|$, where $|uv|$ is the Euclidean distance between u and v. If this graph has $O(n)$ edges, then it is a sparse approximation of the (dense) complete Euclidean graph on V.

Many algorithms are known that compute t-spanners with $O(n)$ edges

[10, 100, 117, 142, 159] that have additional properties such as bounded degree [16, 25], small spanner diameter [16] (i.e., any two vertices are connected by a t-spanner path consisting of only a small number of edges), low weight [40, 41, 78] (i.e., the total length of all edges is proportional to the weight of a minimum spanning tree of V), planarity [14, 101] and fault-tolerance [39, 118, 2]; see also the surveys [59, 84, 150] and the book [129]. All these algorithms compute t-spanners for any given constant $t > 1$. Throughout this chapter we will assume that $t \leq 2$. If this is not the case we run the algorithm with $t = 2$.

Three approaches have been proven to generate t-spanners of bounded weight. The greedy algorithm was first presented in 1989 by Bern. Althöfer et al. [10] gave the first theoretical bounds and since then the greedy algorithm has been subject to considerable research [29, 30, 40, 41, 42, 78, 152]. It was shown to produce t-spanners of weight $O(\frac{1}{(t-1)} \cdot wt(MST(V)))$ [41], for a complete proof see [129]. However, a simple and naive implementation has a running time of $O(n^3 \log n)$. Das and Narasimhan [41] developed an approximate greedy algorithm with running time roughly $O(\frac{1}{(t-1)^d} \cdot n \log^2 n)$ that produces a t-spanner of weight $O(\frac{1}{(t-1)^{2d}} \cdot wt(MST(V)))$. The time complexity was later improved to $O(\frac{1}{(t-1)^{4d-1}} \cdot n \log n)$ by Gudmundsson, Levcopoulos and Narasimhan [78]. Both these algorithms are quite involved and non-trivial to implement. The algorithm by Das and Narasimhan requires a hierarchy of cluster graphs in which a linear number of shortest-path queries are performed. The improvement by Gudmundsson et al. [78] speeds up the clustering and the shortest path queries by running Dijkstra's algorithm in parallel on a cluster graph where the edge weights have been rounded to integer weights.

Callahan and Kosaraju showed that a t-spanner can be obtained in $O(\frac{n}{(t-1)^d} + n \log n)$ time by using the well-separated-pair decomposition, abbreviated WSPD. Arya et al. [16] used a modified version of the standard WSPD spanner algorithm, and then analyzed the weight of the graph using the so-called Dumbbell Theorem to prove that the weight is bounded by $O(\log n \cdot wt(MST(V)))$. However, in [64, 65] Farshi and Gudmundsson performed extensive experiments on the most common algorithms and one of the surprising results was the poor performance of the WSPD-graph algorithm, both in terms of running time and in terms of quality (number of edges, total weight, maximum degree, ...) of the produced spanners. This follows from the fact that the separation constant needed to guarantee a t-spanner is required to be very small. Thus, the number of well-separated pairs becomes very large (although linear with respect to n in theory).

The third approach that guarantees the spanner to have weight $O(\log n \cdot wt(MST(V)))$ is the use of the strong w-gap property [30] (definition is given in Section 6.1). In 1997 Arya and Smid [18] showed an algorithm, denoted GapGreedy that produced a t-spanner of V in $O(\frac{1}{(t-1)^{d-1}} \cdot n \log^2 n)$ time that fulfills the w-gap property, thus has total weight $O(\frac{\log n}{(t-1)^d} \cdot wt(MST(V)))$. The special case when $w = 0$, was discovered by Salowe [152] and, according to Vaidya [159], also by Feder and Nisan.

In this chapter we will prove and make use of the relaxed w-gap property, which makes our algorithm very simple and easy to implement. The most complicated tool used is a data structure that supports orthogonal range queries in $O(\log^{d-1} n + k)$ time, where k is the number of reported vertices. Note that this is a standard range tree that is very easy to implement.

6.1 Preliminaries

In this section the gap property and the gap theorem are introduced. We then relax the property and extend the theorem in the next section.

Recall that for any directed edge (p, q), p is called the source, and q is called the sink.

Definition 6.1 (Gap Property)
Let $w \geq 0$ be a real number, and let E be a set of directed edges in $\mathbb{R}^d$.

1. We say that E satisfies the w-gap property if for any two distinct edges (p, q) and (r, s) in E, we have

$$|pr| > w \cdot min(|pq|, |rs|).$$

2. We say that E satisfies the strong w-gap property if for any two distinct edges (p, q) and (r, s) in E, we have

$$|pr| > w \cdot min(|pq|, |rs|) \quad \text{and} \quad |qs| > w \cdot min(|pq|, |rs|).$$

The gap property was introduced by Chandra et al. [30]. They also proved the Gap Theorem. The theorem bounds the total length of any set of directed edges that satisfies the gap property.

Theorem 6.2 *(Gap Theorem [30])*
Let V be a set of n vertices in $\mathbb{R}^d$, and let $E \subset V \times V$ be a set of directed edges that satisfies the w-gap property.

1. If $w \geq 0$, then each vertex of V is the source of at most one edge of E.

2. If $w > 0$, then $wt(E) < (1+2/w) \cdot wt(MST(V)) \log n$, where $MST(V)$ denotes a minimum spanning tree of V.

3. If $w \geq 0$, and E satisfies the strong w-gap property, then each vertex of V is the sink of at most one edge of E.

Lenhof, Salowe, and Wrege [116] (see also Alon and Azar [8] and Chandra, Karloff, and Tovey [31]), proved a lower bound saying that there exists a set V of n vertices in the plane and a set $E \subset V \times V$ of directed edges such that E satisfies the w-gap property, and $wt(E) \geq \frac{\log n}{6 \log \log n} \cdot wt(MST(V))$. This bound was later improved to $\Omega(\log n \cdot wt(MST(V)))$ by Narasimhan and Smid [129].

6.2 The extended gap theorem

Using logical transposition ($A \rightarrow B$ is equivalent to $\neg B \rightarrow \neg A$) the gap theorem states that if the total weight of E is at least $((1+2/w) \cdot wt(MST(V)) \log n)$ then at least two edges of E must be such that their sources are relatively close. In this section we extend the gap theorem. Unless a non-constant number of edges of E are almost identical, that is, have very similar length, slope and position, then the length of the graph G is within $O(\log n \cdot wt(MST(V)))$.

Definition 6.3 (Relaxed Gap Property)
Let $w \geq 0$ be a real number, and let E be a set of directed edges in $\mathcal{R}^d$. Consider any two distinct edges $e = (p, q)$ and $e' = (p', q')$ in E. We say that E satisfies the relaxed w-gap property if

$$|pp'| \leq w \cdot \min(|e|, |e'|)$$

then

$$|qq'| > w \cdot \min(|e|, |e'|).$$

The Relaxed Gap Property relaxes the condition that the sources of the edges may not be too close. Instead we only require that either the sources or the sinks may not be too close.

For simplicity we say that the *angle* between two directed edges is the smallest angle between the two vectors coinciding with the two edges. Before we prove the extended gap theorem, we will need the following:

Definition 6.4 Let E be a set of directed edges in $\mathbb{R}^d$ for which the relaxed gap property holds, and let $0 < \delta < w$ and $0 < \theta$ be two real constants.

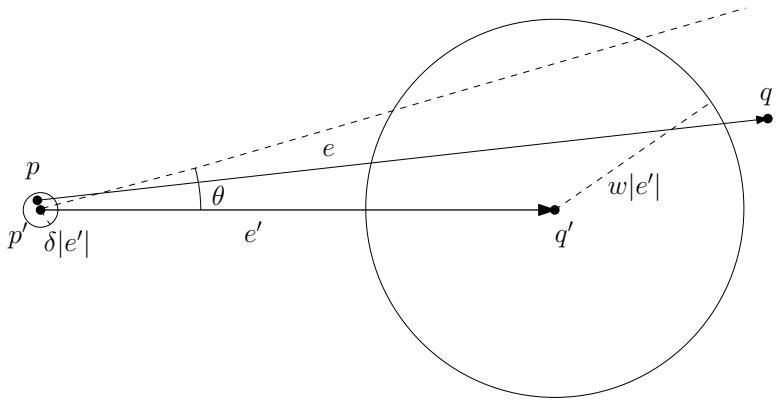

Figure 6.1: Two edges $e, e' \in E$ fulfilling the θ, δ-gap property.

We say that two edges $e_1 = (p_1, q_1)$ and $e_2 = (p_2, q_2)$ fulfill the (θ, δ)-gap property if and only if (see Fig. 6.1):

(1) $|p_1 p_2| \leq \delta \cdot \min(|e_1|, |e_2|)$ and

(2) the angle between e_1 and e_2 is at most θ.

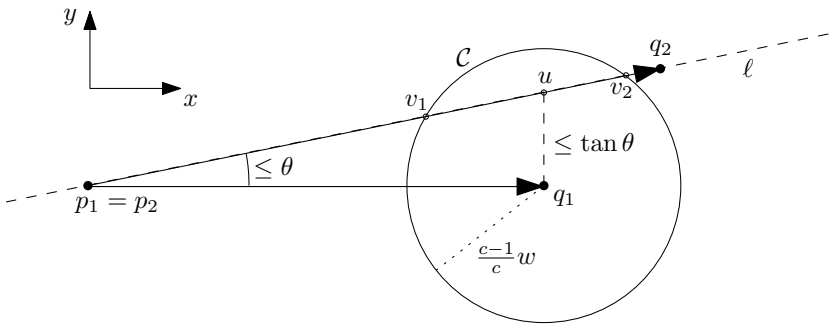

Figure 6.2: Illustration of the proof of Observation 6.6.

Observation 6.5 *Given two edges e_1 and e_2 in $\mathcal{R}^d$ that fulfill the (θ, δ)-gap property with $0 < \delta < w/c$ and $0 < \theta < \arcsin w/k$, we have*

$$\max\left(\frac{|e_1|}{|e_2|}, \frac{|e_2|}{|e_1|}\right) > 1 + \beta, \quad \text{where} \quad (1+\beta) = \sqrt{1 + \sin^2\theta}.$$

Proof: Assume without loss of generality that $|e_1| \leq |e_2|$, and that $e_1 = (p_1, q_1)$ and $e_2 = (p_2, q_2)$. Consider the plane $\mathcal{P}$ spanned by e_1 and e_2. Rotate the two edges such that e_1 is horizontal in $\mathcal{P}$ and p_1 lies to the left of q_1. To simplify the discussion we assume that $\mathcal{P}$ is in the xy-plane, where the x-axis is horizontal and the y-axis is vertical.

Since e_1 and e_2 fulfill the (θ, δ)-gap property we have $|p_1p_2| \leq \delta \cdot |e_1|$ and that $|q_1q_2| > w \cdot |e_1|$. To simplify the calculations we translate e_2 such that $p_1 = p_2$, which implies that $|q_1q_2| > (w - \delta) \cdot |e_1| \geq (1 - 1/c)w \cdot |e_1|$.

Let u be the point on the line ℓ through p_2 and q_2 with the same x-coordinate as q_1, as shown in Fig. 6.2. Let $\mathcal{C}$ be the circle with center at q_1 and radius $(1 - 1/c)w$, and let v_1 and v_2 be the intersection points between $\mathcal{C}$ and ℓ, where v_1 lies to the left of v_2. To prove the observation we need to show that the distance along ℓ between u and v_1, and u and v_2 is at least β.

Using standard trigonometry we get $|p_2v_1| < |e_1|$ which implies that q_2 must lie to the right of v_1. Furthermore, we get

$$|p_2v_2| > \sqrt{1 + (((1 - 1/c)k)^2 - 2)\sin^2\theta}$$

which implies the observation if we set $c = 4$ and $k = 4$. □

We will also need the following lemma:

Lemma 6.6 *Let e be a directed edge, let E be a directed edge set in $\mathcal{R}^d$, and let $0 < \delta < w/4$ and $0 < \theta < \arcsin w/4$ be two given constants. The number of edges in E is bounded by $O(2^d(1 + \beta)^d)$ if for every edge e' in E it holds that:*

(1) the angle between e and e' is at most θ,
(2) e and e' fulfill the $(\theta/2, \delta)$-gap property, and
(3) $(1 + \beta)^j \cdot |e'| < |e| \leq (1 + \beta)^{j+1} \cdot |e'|$.

Proof: Consider an arbitrary edge $e_1 = (p_1, q_1)$ in E and let $e_2 = (p_2, q_2)$ be the shortest edge in E. From (3) we have $|e_1|/|e_2| < (1 + \beta)$ which implies that e_1 and e_2 do not fulfill the (θ, δ)-gap property, according to Observation 6.5. However, the angle between e_1 and e_2 is at most θ which means $|p_1p_2|$ must be greater than $\delta \cdot |e_2|$, otherwise, e_1 and e_2 would fulfill the (θ, δ)-gap property according to the definition. Note that this implies that the d-dimensional ball with center at p_1 and radius $r_1 = \delta \cdot |e_2|$ must be empty of points.

Consider the edge $e = (p, q)$. From (2) we have that both $|p_1p|$ and $|p_2p|$ are bounded by $\delta \cdot |e_1|$ and $\delta \cdot |e_2|$, respectively. As a result we have $|p_1p_2| \leq \delta(|e_1| + |e_2|) \leq 2\delta(1 + \beta)|e_2|$. Hence, all the sources of the edges in

E must be contained in a d-dimensional ball with center at p_1 and radius $2(1+\beta)|e_2|$.

Recall that the volume of a d-dimensional ball of radius r is $\frac{(2\pi)^{d/2}\cdot r^d}{2\cdot 4\cdots d}$ if d is even, otherwise the volume is $\frac{(2\pi)^{d/2}\cdot r^d}{1\cdot 3\cdots d}$. By using a standard packing argument on the two bounds we obtain:

$$\#E = O(2^d(1+\beta)^d).$$

□

We can now show the following theorem:

Theorem 6.7 *(Extended Gap Theorem) Let $w \geq 0$ be a real number, let V be a set of vertices in $\mathcal{R}^d$, and let $E \subset V \times V$ be a set of directed edges satisfying the relaxed w-gap property. It holds that*

$$wt(E) = \sum_{e \in E} |e| = O(2^d(1+\beta)^d \cdot (1+8/w) \cdot \log n \cdot wt(MST(V))).$$

Proof: Consider a partition of E into $k = O(2^{d-1}(1+\beta)^{d-1})$ sets $E_1, \ldots, E_k$ where the angle between any two edges in E_i, $1 \leq i \leq k$, is bounded by $\theta = \arcsin w/4$.

Next, for each set E_i we consider a maximum subset $E_i' \subseteq E_i$ such that no pair of edges in E_i' fulfills the (θ, δ)-gap property, where $\delta = w/4$. The edges in E_i' are chosen such that an edge e in E_i' is longer than any edge e' that fulfills the (θ, δ)-gap property with e.

Note that the edges in E_i' fulfill the relaxed gap property but not the (θ, δ)-gap property. Since the edges in E_i' has approximately the same direction it follows from the construction of E_i' that for any two edges $e_1 = (p_1, q_1)$ and $e_2 = (p_2, q_2)$ in E_i' the distance between p_1 and p_2 must be at least $\delta \cdot \min(|e_1|, |e_2|)$, which implies that E_i' fulfills the δ-gap property (Definition 6.1), and case (2) of Theorem 6.2. Consequently, $wt(E_i') \leq (1 + 2/\delta) \cdot \log(\#E_i) \cdot wt(MST(V_i'))$, where V_i' is the set of points spanned by E_i'.

To get the bound stated in the theorem it remains to prove that $wt(E_i) = O(2^d(1+\beta)^d \cdot wt(E_i'))$. Consider an edge e of E_i' and let $D_i(e)$ be the set of edges in E_i that fulfill the (θ, δ)-gap property with e. Recall that e is longer than any edge e' in $D_i(e)$. We claim that $wt(D_i(e)) = O(|e|)$ which would complete the theorem, and according to Observation 6.5 we have $|e|/|e'| > 1 + \beta$ for any edge e' in $D_i(e)$. Group the edges in $D_i(e)$ into sets $D_i^j(e)$ with respect to their length, such that the length of the edge

in $D_i^j(e)$ is in the interval $\left(\frac{|e|}{(1+\beta)^{j+1}}, \frac{|e|}{(1+\beta)^j}\right]$. From Lemma 6.6 we have $\#D_i^j(e) = O(2^d(1+\beta)^d)$, hence:

$$\sum_{j=1}^{\infty} wt(D_i^j(e)) = O(2^d(1+\beta)^d) \cdot \sum_{j=1}^{\infty} \frac{|e|}{(1+\beta)^j} = O(2^d(1+\beta)^d)|e|.$$

We can now put together the results:

$$\begin{aligned} \sum_{i=1}^{k} wt(E_i) &= O(2^d(1+\beta)^d) \cdot \sum_{i=1}^{k} wt(E_i') \\ &= O(2^d(1+\beta)^d \cdot (1+2/\delta) \cdot \log n \cdot wt(MST(V))). \end{aligned}$$

□

Even though the theorem is only stated for directed edges, it can easily be generalized to the undirected case.

6.3 A fast and simple construction of a low weight t-spanner

Next we consider how to construct a t-spanner of low weight fast and simple, using the extended gap theorem. As input we are given a set of points V in $\mathcal{R}^d$ and the general idea is straight-forward and consists of two main steps: (1) compute a (directed) $\sqrt{t}$-spanner $G = (V, E)$ that is then (2) pruned by removing all edges that do not fulfill the relaxed gap-property. After replacing each directed edge with an undirected edge the resulting graph $G' = (V, E')$ will be shown to be a $\sqrt{t}$-spanner of G, thus a t-spanner of V. The idea is that the graph G' should have the added property that the weight of the edge set is bounded.

As discussed in the introduction there are many different t-spanner algorithms that can be used for the first step of our algorithm [129]. At the end of this section we will discuss this in more detail. For now we will just assume that the first step of the algorithm generates a $\sqrt{t}$-spanner G of V with N edges. To make it directed we just replace each edge with a directed edge (arbitrary direction).

The second step of the algorithm is presented in detail in the next section and the correctness of the algorithm is given in Section 6.3.2.

6.3.1 The pruning step

In this section we are given a $\sqrt{t}$-spanner $G = (V, E)$ and the aim is to prune G such that the resulting graph $G' = (V, E')$ is a $\sqrt{t}$-spanner of G and the edges in E' fulfill the relaxed gap property, and thus has bounded weight.

First we direct all the edges from left to right, to make the graph directed. To simplify the arguments we set $\min\{\sqrt{t} - 1, 1\} = \varepsilon$. Thus, we aim to construct a $(1 + \varepsilon)$-spanner G' of G.

The first step is to partition the edge set E into sets $E_1, \ldots, E_m$ such that the length of the edges in a set differ by at most a factor $(1 + \lambda)$, where $\lambda = \varepsilon/4$. The set E_1 contains all edges of E of length at most $(1 + \lambda) \cdot |e|$, where e is the shortest edge of E. In a generic step the set E_i contains the shortest edge e in $E \setminus \{E_1 \cup E_2 \cup \ldots \cup E_{i-1}\}$ and all edges in $E \setminus \{E_1 \cup E_2 \cup \ldots \cup E_{i-1}\}$ of length at most $(1 + \lambda) \cdot |e|$.

Next partition each set E_i into a minimum number of sets $E_{i,1}, \ldots, E_{i,k}$ where the angle between to edges in $E_{i,j}$ differ by at most θ. Note that $k = O(2^{d-1})$ and that we set $\theta = \arcsin(\varepsilon/8)$.

Process each set $E_{i,j}$ as follows. Consider the source vertex of each edge in $E_{i,j}$ and denote this vertex set $S_{i,j}$. Note that each vertex p in $S_{i,j}$ corresponds to an edge $e(p)$ in $E_{i,j}$, thus if two edges e_1 and e_2 share a source p then two identical vertices, p_1 and p_2, are added to $S_{i,j}$. Build a range tree $T_{i,j}$ on

For each set $S_{i,j}$ let $\mathcal{S}_{i,j}$ denote the union of sets $S_{g,h}$ such that there may exist a point q in $S_{g,h}$ and a point p in $S_{i,j}$ such $e(p)$ and $e(q)$ differ in angle by at most θ and such that $\max\{e(p)/e(q), e(q)/e(p)\} \leq (1 + \lambda)$. Finally, let $\mathcal{E}_{i,j}$ denote the edge set corresponding to $\mathcal{S}_{i,j}$.

Fact 6.8 *[46] A set of n vertices in the plane can be preprocessed in order to construct a data structure of size $O(n \log^{d-1} n)$. The data structure takes $O(n \log^{d-1} n)$ time to construct, and supports orthogonal range queries in $O(\log^{d-1} n + k)$ time, where k is the number of vertices reported.*

Next, we incrementally build the edge set E', where E' is initially empty. Process the edges of E in order of increasing length. For each processed edge e, consider its source vertex $p \in S_{i,j}$. Let $Q(p)$ be the square with center at p and side length $\gamma \cdot |e(p)|$, where $\gamma = \frac{\varepsilon}{32\sqrt{d}}$. Move the edge $e(p)$ from E to E' and perform an orthogonal range query in $T_{i,j}$ with $Q(p)$. Let $T_{i,j}(p)$ be the set of reported vertices. For each vertex q in $T_{i,j}(p)$ test if $e(q)$ and $e(p)$ violate the relaxed w-gap property, where $w = \gamma/2$. If so, remove $e(q)$ from E. That is, $e(q)$ will never be tested and $e(q)$ will never

be added to E'. Continue until E is empty. We claim that the resulting graph $G' = (V, E')$ is a t-spanner of V.

Lemma 6.9 *The graph $G' = (V, E')$ can be constructed from $G = (V, E)$ using $O(d2^{d-1}N \log^{d-1} N)$ time and space.*

Proof: Consider the construction of G'. First the edges of G are partitioned into sets $E_1, \ldots E_m$. This can be done by sorting the edges by increasing length, and then iterate through the edges in order. Each group E_i is further subdivided into $k = O(2^{d-1})$ groups $E_{i,1}, \ldots, E_{i,k}$, where each edge can be placed in the correct group in time $O(d)$. Thus, the edges are partitioned into groups, which requires $O(dN + N \log N)$ time.

Next, consider a range tree $T_{i,j}$, where we let $N_{i,j}$ denote the total number of points in $\mathcal{S}_{i,j}$ Since any set $S_{i,j}$ is included in $O(2^d)$ of the created range trees, we have that $\sum_{i=1}^{m} \sum_{j=1}^{k} N_{i,j} = O(2^d N)$. Thus, preprocessing takes $O(2^d N \log^{d-1} 2^d N)$ time in total.

Next we consider how many times a vertex may be reported in total during the queries. An edge e is removed from E to which it belongs once it has been reported. Its corresponding vertex p is not removed from any range trees. However, each vertex will only be reported $O(2^d)$ times. This follows since only $O(2^d)$ query squares, of similar size, can cover p, such that none of the square centers are contained within another square (other than its own). This means that the total number of reported vertices is $O(2^d N)$ and that all queries and deletions of corresponding reported vertices, takes $O(2^d N \log^{d-1} 2^d N)$ time in total. This completes the lemma. □

6.3.2 Correctness proof

It remains to prove that the algorithm produces a graph that is a t-spanner of V (Lemmas 6.10-6.11) and has low weight (Lemma 6.12).

Lemma 6.10 *Given an edge $e = (p, q) \in E \setminus E'$ there exists an edge $e' = (p', q') \in E'$ such that*

$$|pp'| < \sqrt{d}\gamma|pq| \quad \text{and} \quad |qq'| < \sqrt{d}\gamma|pq| + \lambda \cdot |pq| + (1 + \lambda)|pq| \sin\theta.$$

Proof: Consider the step in the algorithm where e is discarded, and assume that the edge $e' = (p', q')$ was processed at that time. Since e was discarded p must be inside $Q(p')$. Recall that the side length of $Q(p')$ is $\gamma|pq|$ and that the angle between e and e' is at most θ.

As in Observation 6.5 we consider the plane $\mathcal{P}$ spanned by e and e'. Rotate the two edges such that e' is horizontal in $\mathcal{P}$ and p' lies to the left of q'. To simplify the discussion we assume that $\mathcal{P}$ is in the xy-plane, where the x-axis is horizontal and the y-axis is vertical.

Let $p.x$ and $p.y$ denote the x-coordinate and y-coordinate of p. Obviously $|qq'| \leq |q.x - q'.x| + |q.y - q'.y|$. First we bound the first term.

Since $p \in Q(p')$ we have $|p.x - p'.x| \leq \sqrt{d} \cdot \frac{\gamma}{2} \cdot |pq|$. Also, $|p.x - q.x| \leq \lambda \cdot |pq|$. This gives that $|q.x - q'.x| \leq \sqrt{d}\gamma/2 \cdot |pq| + \lambda \cdot |pq|$.

Next, we consider the second term. Since the angle is at most θ we have that $|q.y - q'.y| \leq \sqrt{d}\gamma/2 + (1+\lambda)|pq| \sin\theta$, see Fig. 6.3. Putting together the two terms proves the lemma. □

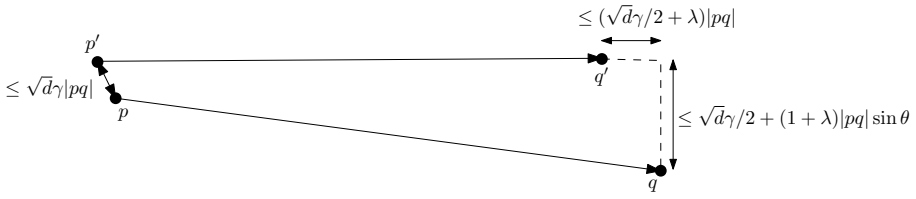

Figure 6.3: Illustrating the proof of Lemma 6.10.

Lemma 6.11 *The graph G' is a t-spanner of V.*

Proof: Recall that $G = (V, E)$ is a $\sqrt{t}$-spanner of V so it suffices to prove that G' is a $(1+\varepsilon)$-spanner of G. The proof will show that for every edge (p, q) in G there is a path in G' of length at most $(1+\varepsilon) \cdot |pq|$, which immediately implies that $\delta_{G'}(s, t) \leq (1+\varepsilon) \cdot \delta_G(s, t)$ for any pair of vertices $s, t \in V$.

The proof is by induction on the length of the edges in E. As the base case consider the shortest edge $(p, q) \in E$. Assume that p lies to the left of q. Following the algorithm it is obvious that the vertex p corresponding to (p, q) is the first vertex processed, and hence must be added to E'.

Consider an arbitrary edge $e = (p, q)$ in E, and assume that the claim is true for all edges of E shorter than e.

If $(p, q) \in G'$ then we are done. Otherwise, there must be an edge (r, s) such that they belong to the same set $E_{i,j}$ and (r, s) was processed before (p, q). Furthermore, if r is the left end vertex of (r, s) then r must lie within the query region $Q(p)$. This implies that the distance between p and r is

smaller than w. We have:

$$\begin{aligned}
\delta_{G'}(p,q) &\leq \delta_{G'}(p,r) + |rs| + \delta_{G'}(q,s) \\
&< (1+\varepsilon)\gamma \cdot |pq| + |pq| + (1+\varepsilon) \cdot (\sqrt{d}\gamma|pq| + \lambda \cdot |pq| \\
&\qquad +(1+\lambda)|pq|\sin\theta) \\
&= ((1+\varepsilon)\gamma + 1 + (1+\varepsilon)(\sqrt{d}\gamma + \lambda + (1+\lambda)\sin\theta)) \cdot |pq| \\
&= ((1+\varepsilon)\frac{\varepsilon}{32\sqrt{d}} + 1 + (1+\varepsilon)(\frac{\varepsilon}{32} + \frac{\varepsilon}{4} + (1+\frac{\varepsilon}{4}) \cdot \frac{\varepsilon}{8}))|pq| \\
&< (1+\varepsilon) \cdot |pq|.
\end{aligned}$$

This completes the proof of the lemma. □

Next we show that the created edge set E' fulfills the relaxed w-gap property.

Let $E'_{i,j}$ be the set of edges in $E' \cap E_{i,j}$.

Lemma 6.12 *The edge set E' fulfills the relaxed w-gap property, where $w{=}\gamma/2$.*

Proof: Consider any two distinct edges $e_1 = (p,q)$ and $e_2 = (p',q')$ in E', where we assume without loss of generality that e_1 is shorter than e_2, and $e_1 \in E'_{i,j}$. Again we consider the plane $\mathcal{P}$ spanned by e_1 and e_2. Rotate the two edges such that e_1 is horizontal in $\mathcal{P}$ and p_1 lies to the left of q_1. To simplify the discussion we assume that $\mathcal{P}$ is in the xy-plane, where the x-axis is horizontal and the y-axis is vertical.

We must show that if $|p_1,p_2| \leq w \cdot \min(|e_1|,|e_2|) = w|e_1|$, then it holds that $|q_1,q_2| > w \cdot \min(|e_1|,|e_2|) = w|e_1|$.

First consider the case when e_2 is included in the set $\mathcal{E}_{i,j}$. Since the algorithm explicitly tests e_1 and e_2 in this case, the w-gap property immediately holds, with $w = \gamma/2$.

If the above case does not hold, i.e., $e_2 \notin \mathcal{E}_{i,j}$, then we know from the algorithm that either the angle between e_1 and e_2 is at least θ, or that $|e_2| \geq (1+\lambda)|e_1|$. We have two cases:

- *Angle between e_1 and e_2 is at least θ.* Since the slope of e_2 is at least θ and since $|e_2| \geq |e_1|$ we have $|p_2.y - q_2.y| \geq \sin\theta = \frac{\varepsilon}{8} \cdot |p_1q_1|$. This observation immediately implies that $|q_1.y - q_2.y| \geq \frac{3\varepsilon}{32}|p_1q_1| > \frac{\gamma}{2}|p_1q_1|$ since $|p_1.y - p_2.y| \leq \frac{\sqrt{d}\gamma}{2}|p_1q_1| = \frac{\varepsilon}{64}|p_1q_1|$, see Fig. 6.4.

- $|e_2| \geq (1+\lambda)|e_1|$. Assume that e_1 is horizontal. Consider $Q(p_1)$ and $Q(q_1)$. Consider any edge $e = (p,q)$ with $p \in Q(p_1)$ and $q \in Q(q_1)$. It

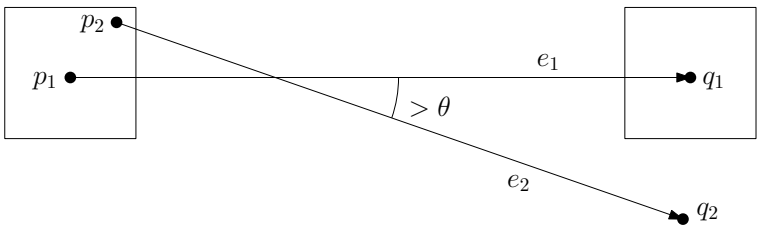

Figure 6.4: Illustrating the case that e_1 and e_2 are not included in the same set $E'_{i,j}$.

is easy to see that the length of e must be shorter than $(1+\varepsilon/16)\cdot|p_1q_1|$, see Fig. 6.5. From the definition of the edge sets it follows that $|e_2| \geq (1+\lambda)\cdot|e_1| = (1+\varepsilon/4)\cdot|e_1|$, which implies that either $p_2 \notin Q(p_1)$ or $q_2 \notin Q(q_1)$.

This completes the proof of the lemma. □

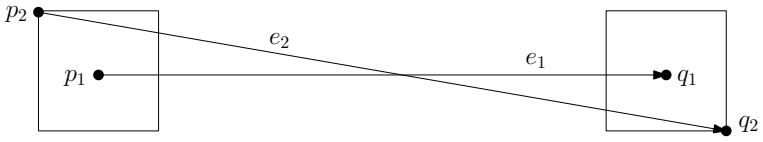

Figure 6.5: Illustrating the case $|e_2| \geq (1+\lambda)|e_1|$.

Next we bound the weight of the spanner by combining the results.

Lemma 6.13 *The weight of the edges in G' is bounded by*

$$O\Big(\frac{\sqrt{d}\cdot 2^d}{\varepsilon}\cdot\log n\cdot wt(MST(V))\Big).$$

Proof: The algorithm (Section 6.3.1) initialized θ and γ to $\arcsin\varepsilon/8$ and $\frac{\varepsilon}{32\sqrt{d}}$, respectively. According to Observation 6.5, we have $(1+\beta) = \sqrt{1+\sin^2\theta} = \sqrt{1+\varepsilon^2/64}$, and according to Lemma 6.12 we have $w = \gamma/2$. The Extended Gap Theorem (Theorem 6.7) states:

$$wt(E) = \sum_{e\in E}|e| = O(2^d(1+\beta)^d\cdot(1+8/w)\cdot\log n\cdot wt(MST(V))).$$

Thus plugging in the values gives:

$$wt(E) = \sum_{e \in E} |e| = O(2^d(1+\varepsilon^2/64)^d \cdot (1 + \frac{8 \cdot 32 \cdot \sqrt{d}}{\varepsilon}) \cdot \log n \cdot wt(MST(V)))$$

$$= O(\frac{\sqrt{d}2^d}{\varepsilon} \log n \cdot wt(MST(V))).$$

□

In the previous section we assumed that a $\sqrt{t}$-spanner of V was given with N edges. However, we did not specify how this spanner was constructed. There are many ways such a spanner can be constructed and we may choose different approaches for different purposes, see for example the survey by Gudmundsson and Knauer [84].

If it is desirable to also have bounded degree one may choose to use the sink spanner [16] which can be constructed using the sink-spanner algorithm by Arya et al. [16]. The resulting graph $G = (V, E)$ has maximum degree $O(\frac{1}{(t-1)^{2d-2}})$ and $O(\frac{n}{(t-1)^{d-1}})$ edges and can be constructed using $O(\frac{n}{(t-1)^{d-1}} \log^{d-1} n)$ time.

Since G has $O(\frac{n}{(t-1)^{d-1}})$ edges the pruning takes $O(\frac{d2^d n}{(t-1)^{d-1}} \log^{d-1} \frac{n}{(t-1)^{d-1}})$ time and space. The following theorem summarizes the results in the Euclidean plane.

Theorem 6.14 *One can construct a t-spanner of a set V of n vertices in $\mathcal{R}^d$ with degree $O(\frac{1}{(t-1)^{2d-2}})$ and weight $O(\frac{\sqrt{d} \cdot 2^d}{(\sqrt{t}-1)} \cdot \log n \cdot wt(MST(V)))$ in $O(\frac{n}{(t-1)^{d-1}} \log^{d-1} n)$ time and space.*

6.4 Concluding remarks

In this chapter we introduced a relaxed version of the w-Gap Property and proved that unless a non-constant number of edges of E in $\mathcal{R}^d$ are almost identical, that is, have very similar length, slope and position, then the total weight of the graph is bounded by $O(2^d \cdot (1 + 8/w) \cdot \log n \cdot wt(MST(V)))$. Compared to the original Gap Theorem our bound is larger by roughly a factor 2^d. It is natural to ask if this can be improved to only have a polynomial dependence on d, or even just a constant factor.

We used the Relaxed Gap Property to obtain a simple and efficient algorithm for constructing low weight t-spanners. However, we believe that the Extended Gap Theorem should be applicable to other types of geometric networks.

Part III

Assorted Problems

Chapter 7

Chips on Wafers, or Packing Rectangles into Grids

In the VLSI wafer industry it is nowadays possible that multiple projects share a single fabrication matrix (the wafer); this permits fabrication costs to be shared among the participants. No a priori constraints are placed on either the size of the chips nor on the aspect ratio of their side lengths (except the maximum size of the outer bounding box). After fabrication, in order to free the separate chips for delivery to each participant, they must be cut from the wafer. A diamond saw slices the wafer into single chips. However cuts can only be made all the way across the bounding box, i.e., all chips must be placed within a grid. A grid is a pattern of horizontal and vertical lines (not necessarily evenly spaced) forming rectangles in the plane. There are some practical constraints, for example, the distance between two parallel cuts cannot be infinitely small, since machines with a finite resolution must be programmed with each cut pattern. Although some of these constraints may simplify the problem we will not consider them in this chapter.

We will consider the theoretical aspects of the problem and we simplify it by assuming that the wafers are rectangular. In practice the wafers are usually circular, although rectangular wafers are produced and used in small scale [124, 144]. This application leads us to define grid packing as follows.

Definition 7.1 A set of rectangles $\mathcal{S}$ is said to be *grid packed* if there exists a rectangular grid such that every rectangle lies in the grid and there is at most one rectangle of $\mathcal{S}$ in each cell, as illustrated in Fig. 7.1. The grid is said to enclose $\mathcal{S}$ if it contains no empty rows or columns and the height/width of every row/column is minimized with respect to its included rectangles. The

area of a grid packing is the area of a minimal bounding box that contains all the rectangles in the grid packing.

There are several cases in the chip design industry where it is favorable to produce a good grid packing. Some practitioners have therefore asked for algorithms solving such problems [97]. In practice one is given a set of chips and one wants to minimize the number of wafers used. A naïve simulated-annealing rectangle placement [97] with n rectangles might be packed on n separate wafers, as a worst case. This means, fabricating n wafers, wasting all except one chip on each wafer. So the number of recovered chips per cut pattern should be maximized.

Note that the number of input rectangles is n and that the input rectangles are allowed to be rotated. The problems considered in this chapter are defined as follows.

Problem 5 [Minimum area grid packing (MAGP)] *Given a set $\mathcal{S}$ of n rectangles find a grid packing of $\mathcal{S}$ that minimizes the total area of the grid.*

We also consider several interesting variants of the problem, for example:

Problem 6 [Minimum area k-grid packing (MAkGP)] *Given a set $\mathcal{S}$ of n rectangles and an integer $k \leq n$ compute a minimum area grid packing containing at least k rectangles of $\mathcal{S}$.*

Problem 7 [Minimum area grid packing with bounded aspect ratio (MAGPAR)] *Given a set $\mathcal{S}$ of n rectangles and a real number $\mathcal{R}$, compute a minimum area grid packing whose bounding box aspect ratio is at most $\mathcal{R}$.*

Problem 8 [Maximum wafer packing (MWP)] *Given a set of n rectangles $\mathcal{S}$ and a rectangular region $\mathcal{A}$ compute a grid packing of $\mathcal{S}' \subseteq \mathcal{S}$ on $\mathcal{A}$ of maximal size.*

Problem 9 [Minimum number of wafers (MNWP)] *A plate is a prespecified rectangular region. Given a set of n rectangles $\mathcal{S}$ compute a grid packing of $\mathcal{S}$ onto a minimal number of plates.*

Weighted variants of the problem is also studied. In the weighted case each rectangle $r \in \mathcal{S}$ also has a weight, denoted $w(r)$, associated to it.

More problems related to wafer-packing can be found in [49]. A problem that is similar to grid packing is the tabular formatting problem [146, 161].

In the most basic tabular formatting problem, one is given a set of rectangular entries and the aim is to construct a table containing the entries in a given order for each row and column. Shin *et al.* [146] suggested three different objective functions that evaluate the quality of a tabular layout: minimal diameter, minimal area, and minimal white space. Therefore we also consider the following problem:

Problem 10 [Minimum diameter grid packing (MDGP)] *Given a set $\mathcal{S}$ of n rectangles find a grid packing of $\mathcal{S}$ that minimizes the diameter of the grid.*

A similar problem, with the exception that the rectangles cannot be rotated, was considered by Beach [20] under the name "Random Pack". Beach showed that Random Pack is strongly $\mathcal{NP}$-hard, and so it partly corresponds to our $\mathcal{NP}$-hardness result in Theorem 7.23.

Another problem similar in flavor to the grid packing problem is the classical 2-dimensional bin packing problem, where one is given a set of n rectangles and an unlimited number of identical rectangular bins and the objective is to allocate all the items to the minimum number of bins. The problem has a rich history and many variants have been considered, see for example [54, 55, 96, 120, 143].

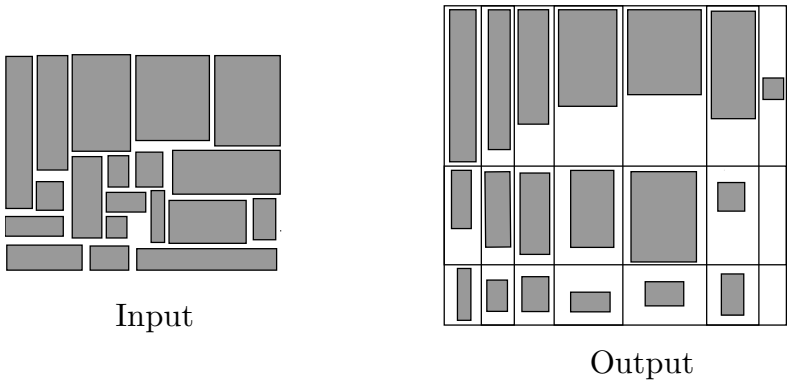

Figure 7.1: Input is a set of rectangles $\mathcal{S}$. Output a grid packing of $\mathcal{S}$

Our main result is a polynomial time approximation scheme (PTAS) for the MAGP-problem and some of its variants (Problems 2, 3 and 6). We also show how the algorithm can be modified to produce approximate solutions for Problems 4 and 5. Surprisingly, if the value of ε is a large enough constant the approximation algorithms will run in linear time.

The approximation algorithms all build upon the same ideas. The main idea is that for every possible grid $\mathcal{G}$ that encloses $\mathcal{S}$, there exists a grid $\mathcal{G}'$ that can be uniquely coded using only $\mathcal{O}(\log n)$ bits such that the area of $\mathcal{G}'$ is at most a factor $(1+\varepsilon)$ larger than the area of $\mathcal{G}$. Now, let $\mathcal{F}$ be the family of these grids that can be uniquely coded using $\mathcal{O}(\log n)$ bits. It trivially follows that there is only a polynomial number of grids in $\mathcal{F}$. Hence, every grid $\mathcal{G}'$ in $\mathcal{F}$ can be generated and tested. The test is performed by computing a maximal packing of $\mathcal{S}$ into $\mathcal{G}'$, which in turn is done by transforming the problem into an instance for the max-flow problem, i.e., given a directed graph with a capacity function for each edge, find the maximum flow through the graph. To obtain a PTAS one uses $\mathcal{O}(\log_{1+\varepsilon} n)$ bits for coding a grid in $\mathcal{F}$. In a similar way only $\log n/2$ bits are used to obtain a linear time approximation algorithm, however the approximation factor will be quite large in this case. More details about $\mathcal{F}$ are given in Section 7.2 together with two important properties. $\mathcal{F}$ is a member of the family of so-called (α, β, γ)-*restricted* grids, which are also described in Section 7.2. Then, in Section 7.3 we show how a grid is tested by reducing the problem to a min-cost max-flow problem. The main theorem is also stated in this section. In Section 7.4 we consider variants of the MAGP-problem, and finally, in Section 7.5 we show some hardness results. Our model of computation is the traditional algebraic computation tree model. In our algorithms we will for simplicity of writing use the notation $\lceil \mathcal{R} \rceil$ to denote the ceiling of a real number $\mathcal{R}$, but the time to compute it will be assumed to be $\mathcal{O}(\log_2 \mathcal{R})$.

7.0.1 Approximability preserving simplifications

In most practical applications we believe that the side lengths of the input rectangles belongs to the interval $[1, n^c]$, for some constant c. Below we will also show that for most of the problems considered in this chapter the side lengths of the input rectangles can be scaled such that the scale lengths lie in the interval $[1, n^c]$ without affecting the approximation results by more than a factor $(1 + \varepsilon)$, for any constant $\varepsilon > 0$. Set $c' = c/2$.

Let h be the length of the longest side of any rectangle in $\mathcal{S}$ and, let w be the length of the longest short side of any rectangle in $\mathcal{S}$ Let $\mathcal{G}$ denote an optimal grid packing, and let H and W denote the height and width of $\mathcal{G}$. Note that the area of $\mathcal{G}$ is bounded from below by $(h \cdot w)$ and from above by $(h \cdot nw)$. We will have two cases, depending on the ratio h/w.

In the case when $h/w < n^{c'}$ we may assume that the width and height of each rectangle $r \in \mathcal{S}$ is in the interval $[1, n^{2c'}]$. Note that $w > n^{c'}$, since $h = n^{2c'}$. We argue that this assumption can be made in this case without

loss of generality with respect to our approximation results. Since the area is approximated we may assume that the shortest side of any input rectangle has length at least 1. If not, the side is expanded such that it has length 1. Note that the side lengths of any solution will expand by at most n which is at most a factor $(1 + n^{1-c'})$ larger than an optimal solution and therefore negligible for the approximation algorithms we consider in this chapter. Hence, we may assume that the sides of the input rectangles have length within the interval $[1, n^{2c'}]$.

In the case when $h/w \geq n^{c'}$, we will show that the orientation of the rectangles in $\mathcal{S}$ can be fixed, i.e., all rectangles are packed "standing". For simplicity we assume that $H \geq W$. Let L be the set of rectangles that are lying down in the optimal packing $\mathcal{G}$, and denote by h' the length of the longest long side of any rectangle in L. If L is empty then the claim holds, hence, we may assume that L contains at least one rectangle. We will have two subcases:

1. If $h' > h/n^2$ then the area of $\mathcal{G}$ is at least $h \times h' \geq h^2/n^2$ which is greater than $h \times w$, thus, we have a contradiction since $area(\mathcal{G}) \leq h \times w = h^2/n^{c'}$.

2. If $h' \leq h/n^2$ then we can place all the rectangles in L standing on top of the optimal grid. Obviously the width is the same as $\mathcal{G}$ and the height increases by at most $n \cdot h' < h/n$. Thus, the area of the new grid is bounded by $(H + h/n) \times W \leq (1 + \varepsilon)H \times W$.

Hence, in the case when $h/w \geq n^{c'}$ we may fix the orientation of the input rectangles (standing). Having fixed the orientation, we may without loss of generality create a new scale for horizontal lengths. In this new scale we can again define the longest width of a "standing" rectangle to be equal to n^c. In this way, using the same arguments as previously, we may assume that all sides of all rectangles have length in the interval $[1, n^c]$.

Similarly, we may assume that all rectangles have weight in the interval $[1, n^c]$. Let $\max_{r \in \mathcal{S}} w(r) = n^c$; it holds that all rectangles with weight less than 1 can be discarded since their combined weight sums up to at most n, which is at most n^{1-c} of the weight of an optimal solution and hence negligible. Note that the above simplification cannot be applied to Problem 2 and 4.

7.1 The approximation algorithm

The approximation algorithm is simple and straight-forward, therefore we here give the global structure. The two non-trivial steps, lines 11 an 12, will be described in detail in Sections 7.2 and 7.3 respectively. The last step, PACKINTOGRID, is obtained by slightly modifying the procedure TESTGRID. As input to the algorithm we will be given a set $\mathcal{S}$ of n rectangles and a real value $\varepsilon' > 0$.

Algorithm GRIDPACK($\mathcal{S}, \varepsilon'$)
1. c is calculated as defined in Section 1
2. $bestVal \leftarrow \infty$, $n \leftarrow |\mathcal{S}|$, $\alpha = \beta = \sqrt{1+\varepsilon'}$, $\gamma = \frac{1}{\sqrt{1+\varepsilon'}-1}$
3. **for** each $1 \leq i, j \leq \log_\alpha n^c$ **do**
4. $\quad \mathcal{S}_{i,j} \leftarrow \emptyset$
5. **for** each $r \in \mathcal{S}$ **do**
6. $\quad i \leftarrow \lceil \log_\alpha \text{ width}(r) \rceil$
7. $\quad j \leftarrow \lceil \log_\alpha \text{ height}(r) \rceil$
8. $\quad \mathcal{S}_{i,j} \leftarrow \mathcal{S}_{i,j} \cup \{r\}$
9. **end**
10. **for** $k \leftarrow 1$ **to** $n^{2c \cdot f(\alpha,\beta,\gamma)}$ **do**
11. $\quad \mathcal{G} \leftarrow$ GENERATEGRID$(\alpha, \beta, \gamma, k, n, c)$
12. $\quad val \leftarrow$ TESTGRID$(\mathcal{G}, \{\mathcal{S}\}_{1 \leq i,j \leq \log_\alpha n^c}, \alpha, \beta, \gamma)$
13. $\quad$ **if** $val < bestVal$ **then**
14. $\quad\quad bestVal \leftarrow val$ and $bestGrid \leftarrow \mathcal{G}$
15. **end**
16. **Output** PACKINTOGRID$(\mathcal{S}, bestGrid)$

The initialization is performed on lines 1 to 4. On lines 5–9, the rectangles are partitioned into groups in such a way that a rectangle $r \in \mathcal{S}$ belongs to $\mathcal{S}_{i,j}$ if and only if the width of r is between α^{i-1} and α^i, and the height of r is between α^{j-1} and α^j. Lines 1-9 obviously run in linear time. Next, a sequence of grids $\mathcal{G}$ are produced in a loop of lines 10-15. They are members of the family of so-called (α, β, γ)-restricted grids which is described in Section 7.2. (This family consists of $n^{(2c \cdot f(\alpha,\beta,\gamma))}$ grids, where $f(\alpha, \beta, \gamma) = (\log(\alpha\beta\gamma))/(\log \alpha \log \beta)$.)

The generated grid is tested and the weight of an approximate grid packing of $\mathcal{S}$ into the grid $\mathcal{G}$ is computed. If the grid packing is better than the previously tested grids then $\mathcal{G}$ is saved as the best grid tested so far. Finally, when all grids in $\mathcal{F}$ have been generated and tested a call to PACKINTOGRID performs a grid packing of $\mathcal{S}$ into the best grid found. This

procedure is a simple modification of the TestGrid-step and it is briefly described in Theorem 7.11 and Lemma 7.18.

7.2 The family $\mathcal{F}$ of (α, β, γ)-restricted grids

The aim of this section is to define the family $\mathcal{F}(\alpha, \beta, \gamma, n, c)$, or $\mathcal{F}$ for short, of (α, β, γ)-restricted grids and prove two properties about $\mathcal{F}$. However, before the properties can be stated we need the following definition. A grid G_1 is said to *include* a grid G_2 if every possible set of rectangles that can be grid packed into G_2 also can be grid packed into G_1. $\mathcal{F}$ has the following two properties.

1. For every grid G that encloses $\mathcal{S}$, and hence has at most n rows and n columns, there exists a grid $\mathcal{G} \in \mathcal{F}$ that includes G and whose width and height is at most a factor $\left(\frac{\alpha^2\beta\gamma}{\alpha\gamma-1}\right)$ times larger than the width and height of G, and
2. $\#\mathcal{F} \leq n^{(2c \cdot f(\alpha,\beta,\gamma))}$ where $f(\alpha, \beta, \gamma) = \frac{\log(\alpha\beta\gamma)}{\log\alpha \log\beta}$.

The definition of an (α, β, γ)-restricted grid is somewhat complicated, therefore we choose to describe this step by step.

A trivial observation is that two grid-packings are equivalent if the one can be transformed to the other by exchanging the order of rows and/or the columns. Hence we may assume that the columns are ordered with respect to decreasing width from left to right and that the rows are ordered with respect to decreasing height from top to bottom. This ordering will be assumed throughout this chapter.

7.2.1 Size of an (α, β, γ)-restricted grid

Consider an arbitrary grid G and let α be a real constant greater than 1. An *α-restricted* grid is a grid where the width and height of each cell in the grid is an integral power of α.

Observation 7.2 *For any grid G there exists an α-restricted grid $\mathcal{G}$ that includes G and whose width and height is at most a factor α longer than the width and height of G.*

Proof: The observation is straight-forward since the width and height of each cell of G can increase by at most a factor α to make the grid α-restricted. □

Let G be a α-restricted grid. If the number of columns/rows of each size β^i, for some integer i, then G is an (α, β)-restricted grid.

Observation 7.3 *For any α-restricted grid G there exists an (α, β)-restricted grid $\mathcal{G}$ that includes G and whose width and height is at most a factor β greater than the width and height of G.*

Proof: The observation is straight-forward since $\mathcal{G}$ can be obtained from G by increasing the number of columns/rows by at most a factor β, hence the observation follows. □

Note, as β^i may not be an integer, the more formal definition of (α, β)-restricted grid would be as above, but with β^i replaced by $\lfloor \beta^i \rfloor$. However, the above observation holds for both cases and for simplicity of writing β^i will be assumed to be an integer.

The columns/rows in an α-restricted grid of width/height α^i are said to have column/row *size* i. A grid G is said to be γ-*monotone* if the number of columns (rows) of size i is greater than or equal to $1/\gamma > 0$ times the number of columns (rows) of size $i + 1$ for every i.

Observation 7.4 *Let γ be a constant greater than or equal to 1. For any (α, β)-restricted grid G there exists a γ-monotone (α, β)-restricted grid $\mathcal{G}$ ((α, β, γ)-restricted grid for short) that includes G and whose width and height is at most a factor $(\gamma\alpha)/(\gamma\alpha - 1)$ greater than the width and height of G.*

Proof: Since the number of columns is decreasing with at most a factor γ and since the width of the columns decrease with a factor α for each size we obtain that the worst-case can be bounded by a geometric series, $\sum_{i=1}^{\infty} 1/(\gamma\alpha)^i < (\gamma\alpha)/(\gamma\alpha - 1)$. The observation follows. □

Putting together the observations immediately gives the following corollary.

Corollary 7.5 *For any grid G there exists an (α, β, γ)-restricted grid $\mathcal{G}$ that includes G and whose width and height is at most a factor $\left(\frac{\alpha^2\beta\gamma}{\alpha\gamma - 1}\right)$ greater than the width and height of G.*

Most often we do not need the actual grid, instead we are interested in the number of cells in the grid of a certain size. That is, the grid $\mathcal{G}$ is represented by a $\lceil \log_\alpha n^c \rceil \times \lceil \log_\alpha n^c \rceil$ integer matrix, where $\mathcal{G}[i, j]$ stores the number of cells in $\mathcal{G}$ of width α^i and, height α^j. We call this a matrix representation of a grid.

7.2.2 Size of the family $\mathcal{F}$ of (α, β, γ)-restricted grids

We are now ready to formally define the family of grids to be considered by the algorithm.

Definition 7.6 Consider $c > 0$ and $\alpha, \beta, \gamma > 1$ as well as a positive integer n. By $\mathcal{F}(\alpha, \beta, \gamma, n, c)$, or $\mathcal{F}$ for short, we denote the set of all (α, β, γ)-restricted grids where both the size of each row and the size of each column is upper bounded by $\log_\alpha n^c$.

Using Corollary 7.5 it is now straight-forward to see that Property 1 holds for $\mathcal{F}$. Next we show that the second property holds for $\mathcal{F}$, i.e., the number of grids that are members of $\mathcal{F}$ is at most $n^{(2c \cdot f(\alpha,\beta,\gamma))}$. It will be a constructive proof where it will be shown that every grid in $\mathcal{F}$ can be coded uniquely with $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^2)$ bits. Hence, producing all words of length $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^2)$ will also generate all members of $\mathcal{F}$.

Assume that we are given a member $f \in \mathcal{F}$ and that f has r_j rows of size j. Recall that r_j is an integral power of β. The idea of the scheme is as follows, see also algorithm CodingRows below. The bit string, denoted S, is built incrementally. Consider a generic step of the algorithm. Assume that the bit string, denoted S_{j+1} has been built for all the row sizes greater than j and that the number of rows of size $(j + 1)$ is r_{j+1}. Initially $S_{\log_\alpha n^c}$ is the empty string. Consider the row size j. We will have two cases, either $r_j \leq (r_{j+1}/\gamma)$ or $r_j > (r_{j+1}/\gamma)$. In the first case, add '1' to S_{j+1} to obtain S_j. In the latter case, when $r_j > (r_{j+1}/\gamma)$, add $(\log_\beta r_j - \max((\log_\beta r_{j+1} - \lceil \log_\beta \gamma \rceil), 0))$ zeros followed by a '1' to S_{j+1} to obtain S_j. Decrease the value of j and continue the process until $j = 0$, and hence, $S_0 = S$.

The algorithm above describes how to generate the rows, the coding for the columns is done in the same way. The following observation proves that property 2 holds for the family of (α, β, γ)-restricted grids.

Observation 7.7 *The length of S is* $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^2)$.

Proof: Consider the code for the rows. There are $\log_\alpha n^c$ different row sizes and, therefore, also at most $\lceil \log_\alpha n^c \rceil$ many '1':s. The total number of rows is at most n^2, hence the number of '0':s is bounded by $\log_\alpha n^c \cdot \log_\beta \gamma + \log_\beta n^2$. □

Algorithm CodingRows$(\alpha, \beta, \gamma, n, c)$
1. $size \leftarrow 2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^2)$

2. $S \leftarrow array[1..size]$, $r_{\lceil \log_\alpha n^c \rceil + 1} \leftarrow 0$, $index \leftarrow 1$
3. **for** each row size $j = \lceil \log_\alpha n^c \rceil$ **downto** 1 **do**
4. $\#Rows \leftarrow \log_\beta r_j - \max((\log_\beta r_{j+1} - \lceil \log_\beta \gamma \rceil), 0)$
5. **for** $k = 1$ **to** #Rows **do**
6. $S[index] \leftarrow$ '0'
7. $index \leftarrow index + 1$
8. **end**
9. $S[index] \leftarrow$ '1'
10. $index \leftarrow index + 1$
11. **end**
12. **while** $index \leq 2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^2)$ **do**
13. $S[index] \leftarrow$ '0'
14. $index \leftarrow index + 1$
15. **end**
16. **Output** S

7.2.3 GenerateGrid

On line 11 in algorithm GridPack the procedure GenerateGrid is called with the parameters α, β, γ and k, where k is a positive integer $\leq n^{(2c \cdot f(\alpha,\beta,\gamma))}$ and hence its binary representation, denoted $B(k)$, has length $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^2)$. Note that for simplicity we chose to generate all bit strings of length at most $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n^2)$ and then decide in the procedure GenerateGrid if it corresponds to an (α, β, γ)-restricted grid or not. Alternatively, one could generate only valid bit strings. We obtain the following corollary:

Corollary 7.8 *Given a bit string B of length $2(\log_\alpha n^c(1+\log_\beta \gamma)+\log_\beta n^2)$ one can in time $\mathcal{O}(\log^2 n)$ construct the unique matrix representation of the corresponding (α, β, γ)-restricted grid, or decide that there is no corresponding (α, β, γ)-restricted grid.*

Proof: It is easily seen that deciding if B corresponds to an (α, β, γ)-restricted grid can be done in linear time w.r.t. the length of B. If B corresponds to a (α, β, γ)-restricted grid the matrix representation of the grid can easily be computed in time $\mathcal{O}(\log_\alpha^2 n + \log_\beta^2 n)$ since the number of columns/rows of each size is obtained by traversing the string, hence the corollary follows. □

7.3 Testing an (α, β, γ)-restricted grid

In the previous section we showed a simple method to generate all possible (α, β, γ)-restricted grids. For the approximation algorithm, shown in Section 7.1, to be efficient we need a way to pack a maximal number of rectangles of $\mathcal{S}$ into the grid. As input we are given a matrix representation of an (α, β, γ)-restricted grid $\mathcal{G}$, and a set $\mathcal{S}$ of n rectangles partitioned into groups $\mathcal{S}_{i,j}$ depending on their width and height. Let $\mathcal{C}_{p,q}$ denote the number of cells in $\mathcal{G}$ that have width α^p and height α^q. We will give an exact algorithm for the problem by reformulating it as a max-flow problem. The problem could also be solved by reformulating it as a matching problem but in the next section we will show that the max-flow formulation can be extended to the weighted case. The max-flow problem is as follows. Given a directed graph $G(V, E)$ with capacity function $u(e)$ for each edge e in E. Find the maximum flow f through G.

The flow network $\mathcal{F}_{\mathcal{S},\mathcal{G}}$ corresponding to a grid $\mathcal{G}$ and a set of rectangles $\mathcal{S}$ contains four levels, numbered from top-to-bottom, and will be constructed level-by-level. In figure 7.3, the weighted variant of the network is shown. The number of nodes on level i, $1 \leq i \leq 4$, is denoted m_i.

Level 1. At the top level there is one node, the source node $\nu^{(1)}$.

Level 2. At level 2 there are $\log_\alpha^2 n^c$ nodes. A node $\nu_{i,j}^{(2)}$ at level 2 represents the group $\mathcal{S}_{i,j}$. For each node $\nu_{i,j}^{(2)}$, there is a directed edge from $\nu^{(1)}$ to $\nu_{i,j}^{(2)}$. The capacity of this edge is equal to the number of rectangles in $\mathcal{S}$ that belong to $\mathcal{S}_{i,j}$.

Level 3. At level 3 there are also $\log_\alpha^2 n^c$ nodes. A node $\nu_{p,q}^{(3)}$ on level 3 represents the set of cells in $\mathcal{C}_{p,q}$. For each node $\nu_{p,q}^{(3)}$ there is a directed edge from node $\nu_{i,j}^{(2)}$ to node $\nu_{p,q}^{(3)}$ if and only if $p \geq i$ and $q \geq j$ (or $q \geq i$ and $p \geq j$), i.e., if a rectangle in $\mathcal{S}_{i,j}$ can be packed into a cell in $C_{p,q}$. All edges from level 2 to level 3 have capacity n.

Level 4. The bottom level only contains one node, the sink $\nu^{(4)}$. For every node $\nu_{p,q}^{(3)}$ on level 3 there is a directed edge from $\nu_{p,q}^{(3)}$ to $\nu^{(4)}$. The capacity of this edge is equal to the number of cells in $\mathcal{G}$ that belongs to $\mathcal{C}_{p,q}$.

The following observation is straight-forward.

Observation 7.9 *The maximal grid packing of $\mathcal{S}$ into $\mathcal{G}$ has size k if and only if the max flow in the flow network is k.*

Proof: Every unit flow corresponds to a rectangle and since the flow is k it implies that every rectangle in $\mathcal{S}$ has been matched to a cell. It is easily seen that a rectangle can only be matched to a cell it can fit into, hence if the flow is k there exists a grid packing of size k of $\mathcal{S}$ into $\mathcal{G}$.

The 'only if' statement is obvious since if there exists a grid packing there must exist a k-matching between the rectangles and the cells, and if this is true there must be a maximal flow of value k. □

In 1998 Goldberg and Rao [74] presented an algorithm for the maximum flow problem with running time $\mathcal{O}(N^{2/3}M\log(N^2/M)\log U)$. If we apply their algorithm to the flow network we obtain the following lemma.

Lemma 7.10 *Given a matrix representation of an (α,β,γ)-restricted grid $\mathcal{G}$ and a set $\mathcal{S}$ of n rectangles partitioned into the groups $\mathcal{S}_{i,j}$ w.r.t. their width and height. (1) The size of an optimal packing of $\mathcal{S}$ in $\mathcal{G}$ can be computed in time $\mathcal{O}(\log^{19/3} n)$. (2) An optimal packing can be computed in time $\mathcal{O}(\log^{19/3} n + k)$, where k is the number of rectangles in the optimal grid packing of $\mathcal{S}$ in $\mathcal{G}$.*

Proof: The lemma follows by plugging in the different values for N, M and U in the above mentioned algorithm by Goldberg and Rao. We have that the number of nodes (N) is $\Theta(\log_\alpha^2 n^c)$, the number of edges ($M$) is $\Theta(\log_\alpha^4 n^c)$, and finally, the integral arc capacities (U) are in the in interval $[1..n]$. □

As a result we obtain the first grid packing theorem.

Theorem 7.11 *Algorithm* GridPack *is a $(1+\varepsilon)$-approximation algorithm for the MAGP-problem with running time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon/7})}\log^{19/3} n+n)$, where $f(\chi)=\frac{2c\log(\chi/(\sqrt{\chi}-1))}{\log^2\chi}$).*

Proof: Corollary 7.5 guarantees the stated approximation ratio, since it holds that $\left(\frac{\alpha^2\beta\gamma}{\alpha\gamma-1}\right)^2 = (1+\varepsilon/7)^3 < (1+\varepsilon)$. The time-complexity of the approximation algorithm is dominated by testing if the set of rectangles can be grid packed into a grid $\mathcal{G}$. The total time for this step is the number of tested grids times the time to test one grid, hence, $\mathcal{O}(n^{f(\chi)}\log^{19/3} n + n)$.

The final packing is easily computed from the max-flow and hence can be done in time linear with respect to the number of rectangles in the packing. □

Even though the expression for the running time in the above theorem looks somewhat complicated it is not hard to see that by choosing the value of ε appropriately we obtain that, algorithm GridPack is a PTAS for the MAGP-problem, and if ε is set to be a large constant GridPack produces a grid packing that is within a constant factor of the optimal in linear time.

7.4 Applications and extensions

The approximation algorithm presented above can be extended and generalized to variants of the basic grid packing problem. Below we show how to extend the algorithm to these variants.

By performing some small modifications to the procedure TestGrid we will obtain the following corollary from Theorem 7.11. Let $f(\chi) = \frac{2c\log(\chi/(\sqrt{\chi}-1))}{\log^2\chi})$ and let $g(\alpha,\beta,\gamma) = \left(\frac{\alpha^2\beta\gamma}{\alpha\gamma-1}\right)$.

Corollary 7.12 *Given an instance of the MNWP-problem, one can compute an $O(1)$-approximation in linear time provided that the optimal solution uses $O(1)$ wafers.*

Proof: Let μ be the number of wafers used in the optimal solution of the MNWP problem. The approximation algorithm becomes as follows. First, compute a $(1+\varepsilon)$-approximation for the MAGP-problem using the GridPack algorithm and an ε large enough for the running time to become linear. Let G be the corresponding grid of the solution produced. Next, create a second grid $\mathcal{Q}$, where each cell has the same size and shape as $\mathcal{A}$, and place $\mathcal{Q}$ on top of G. The number of cells in $\mathcal{Q}$ is minimized as to still cover all of G, where it is straightforward to see that $\mathcal{Q}$ will contain $\mathcal{O}(\mu^2)$ cells. Further, let q be an arbitrary cell in $\mathcal{Q}$ and let $G(q)$ be the subgrid of G whose cells lie entirely within q or is intersecting the right or bottom side of q. The subgrid $G(q)$ is now partitioned into four smaller grids, denoted $G_1(q),\ldots,G_4(q)$, as shown in Fig. 7.2. The first grid contains all cells of $G(q)$ that lie entirely within q. Grid $G_2(q)$ contains all cells of $G(q)$ intersecting the bottom right corner of q, and finally $G_3(q)$ and $G_4(q)$ are the grids induced by the cells in $G(q) \setminus (G_1(q) \cup G_2(q)$ intersecting the bottom and right side, respectively, of q. For each q we pack each smaller grid $G_1(q),\ldots,G_4(q)$ on a separate region $\mathcal{A}$. Obviously such a packing is possible for G_1, and also for $G_2(q),\ldots,G_4(q)$ as they each consist of a single row/column of width/height smaller than the width/height of $\mathcal{A}$. Thus, all of $\mathcal{S}$ may be packed on $\mathcal{O}(\mu^2)$ regions $\mathcal{A}$ and the corollary follows. □

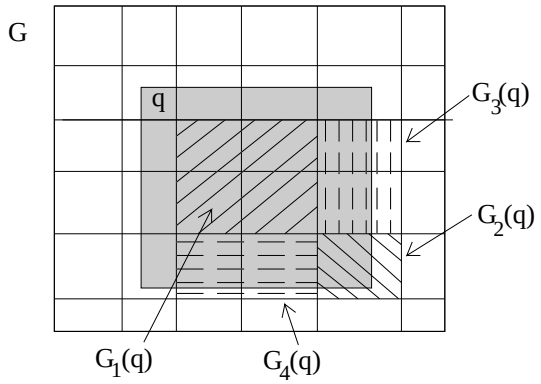

Figure 7.2: Illustration of the proof of Corollary 7.12

Note that the approximation factor for MNWP can be made smaller if we adapt the above method using an ϵ close to zero, but this would imply that the running time, although still polynomial, is not linear any more.

Corollary 7.13 *Given a set $\mathcal{S}$ of rectangles and a real positive value ε one can compute a $(1+\varepsilon)$-approximation of the MDGP-problem in time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon/3})} \cdot \log^{19/3} n + n)$.*

Proof: Consider an optimal grid G with height h and width w. There exists an (α, β, γ)-restricted grid $\mathcal{G}$ that includes G and whose height and width is at most a factor $g(\alpha, \beta, \gamma)$ greater than h and w respectively, hence its diameter is at most a factor $g(\alpha, \beta, \gamma)$ greater than the optimal. □

The proof of Corollary 7.13 can easily be modified to hold for Corollaries 7.14-7.16.

Let $\psi(\mathcal{S}, \mathcal{R})$ denote the area of minimum area grid packing of $\mathcal{S}$ with aspect ratio $\mathcal{R}$.

Corollary 7.14 *Given a set $\mathcal{S}$ of rectangles, a real number $\mathcal{R}$ and a real positive value ε one can compute a grid packing for the MAGPAR-problem of area $(1+\varepsilon) \cdot \psi(\mathcal{S}, \mathcal{R})$ and aspect ratio at most $(1+\varepsilon)\mathcal{R}$ in time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon/3})} \log^{19/3} n + n)$.*

Corollary 7.15 *Given a set $\mathcal{S}$ of n rectangles, an integer $k \leq n$ and a real positive value ε, where the height and width of each rectangle $r \in \mathcal{S}$ is in the range $[1, n^c]$, one can compute a $(1+\varepsilon)$-approximation of the MAkGP-problem in time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon/7})} \log^{19/3} n + n)$.*

Let $\kappa(\mathcal{S}, \mathcal{A})$ denote the cardinality of a maximum grid packing of $\mathcal{S}$ onto $\mathcal{A}$. Let $\mathcal{A}'$ be a rectangular region such that the sides of $\mathcal{A}'$ are a factor $(1+\varepsilon)$ longer that the sides of $\mathcal{A}$.

Corollary 7.16 *Given a set $\mathcal{S}$ of n rectangles, a rectangular region $\mathcal{A}$ and a real positive value ε, one can compute a packing for the MWP-problem of $\mathcal{S}$ onto A of size at least $\kappa(\mathcal{S}, \mathcal{A}')$ in time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon/3})} \log^{19/3} n + n)$.*

7.4.1 Extending the test algorithm

The problems considered below are weighted variants of the grid packing problem, hence every rectangle $r \in \mathcal{S}$ also has a weight $w(r)$. To be able to apply the above algorithm to the weighted case we need a way to pack a maximal number (weight) of rectangles in $\mathcal{S}$ into the grid, or at least approximate the weight of a maximal packing. As input we are given a matrix representation of a (α, β, γ)-restricted grid $\mathcal{G}$, and a set $\mathcal{S}$ of n rectangles partitioned into groups $\mathcal{S}_{i,j}$ depending on their width and height. We will give a τ-approximation algorithm for the problem by reformulating it as a min-cost max-flow problem, where $\tau > 1$ is a given parameter. Hence the call to TestGrid in algorithm GridPack will now include a fifth parameter τ.

A min-cost max-flow problem in a directed graph where each edge e has a capacity $c(e)$ and a cost $d(e)$ is to find the maximum flow f with a minimum cost, where the cost of a flow f is $\sum_{e \in E} d(e) \cdot f(e)$.

The network that will be constructed next contains four levels, numbered from top-to-bottom, and will be constructed level-by-level. A rectangle r in $\mathcal{S}$ belongs to weight class W_k if and only if $w(r)$ is between τ^{k-1} and τ^k. The number of nodes on level i, $1 \leq i \leq 4$, is denoted m_i and we set $W = \sum_{r \in \mathcal{S}} w(r)$.

Level 1. At the top level there is one node, the source node $\nu^{(1)}$.

Level 2. At level 2 there are $\log_\alpha^2 n \cdot \log_\tau n$ nodes. A node $\nu^{(2)}_{i,j,k}$ at level 2 represents the class of rectangles in the group $\mathcal{S}_{i,j}$ and weight class W_k. For each node $\nu^{(2)}_{i,j,k}$, there is a directed edge from $\nu^{(1)}$ to $\nu^{(2)}$. The capacity of this edge is equal to the number of rectangles in $\mathcal{S}$ that belongs to $\mathcal{S}_{i,j}$ and weight class W_k. The cost per unit flow is $W - \tau^{k-1}$, i.e., a rectangle with greater weight is "cheaper" to send than a rectangle with low weight.

Level 3. At level 3 there are $\log_\alpha^2 n^c$ nodes, one for each set $\mathcal{C}_{p,q}$ of cells. Hence, a node $\nu_{p,q}^{(3)}$ on level 3 represents the set of cells in $\mathcal{C}_{p,q}$. For each node $\nu_{p,q}^{(3)}$ there is a directed edge from node $\nu_{k,i,j}^{(3)}$ to node $\nu_{p,q}^{(3)}$ if and only if $p \geq i$ and $q \geq j$, i.e., the rectangles in $\mathcal{S}_{i,j}$ can be packed into the cells in $C_{p,q}$. All the edges from level 2 to level 3 have capacity n and zero cost.

Level 4. The bottom level only contains one node, the sink $\nu^{(4)}$. For every node $\nu_{p,q}^{(3)}$ on level 3 there is a directed edge from $\nu_{p,q}^{(3)}$ to $\nu^{(4)}$ with cost 0 and capacity $|\mathcal{C}_{p,q}|$.

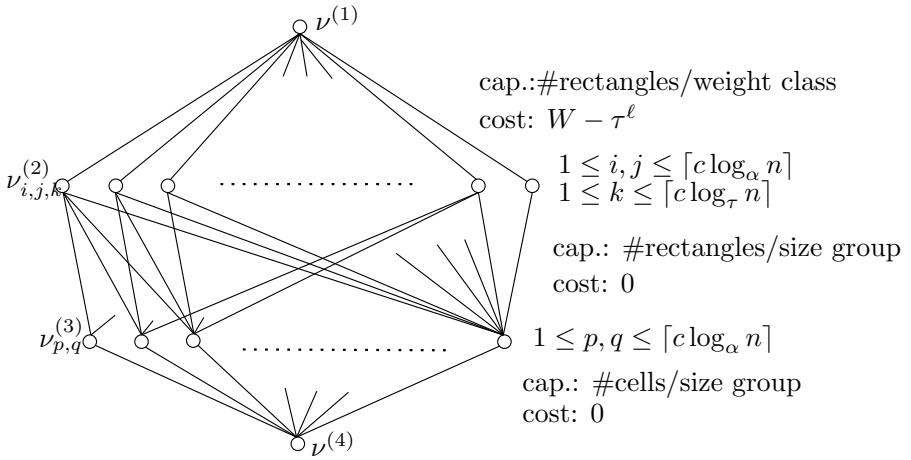

Figure 7.3: Illustrating the flow network given as input to the min-cost max-flow algorithm.

We need the following observation before we can state the main lemma.

Observation 7.17 *If the min-cost max-flow of $\mathcal{F}_{\mathcal{S},\mathcal{G}}$ is $(W - w)$ then the maximal grid packing of $\mathcal{S}$ into $\mathcal{G}$ has weight at most τw.*

Proof: The min-cost max-flow of cost $(W - w)$ is equivalent to a max cost max flow of cost w. It is not hard to see that if there exists a max flow of max cost w then there exists a maximum matching between the rectangles in $\mathcal{S}$ and the cells of $\mathcal{G}$ of total weight w. Since the weight of each rectangle is rounded down such that its weight is divisible by τ it holds that the weight of a maximal packing is at most τw. □

We summarize this section by stating the following lemma.

Lemma 7.18 *Given a matrix representation of an (α, β, γ)-restricted grid $\mathcal{G}$ and a set $\mathcal{S}$ of n rectangles partitioned into the groups $\mathcal{S}_{i,j}$ w.r.t. their width and height. (1) The cost of a τ-approximate packing of $\mathcal{S}$ onto G can be computed in time $\mathcal{O}(\log^9 n \log\log n)$. (2) A τ-approximate packing can be computed in time $\mathcal{O}(\log^9 n \cdot \log\log n + k)$, where k is the number of rectangles in the grid packing.*

Proof: According to Observation 7.17 the solution to the min-cost max-flow problem on $\mathcal{F}_{\mathcal{S},\mathcal{G}}$ will give an approximate solution to the maximal grid packing problem of $\mathcal{S}$ into $\mathcal{G}$ that is at most a factor τ of the optimal.

Let N be the number of vertices in the network; M, the number of edges; U, the maximum value of an edge capacity; and C, the maximum value of an edge cost. The values of the parameters in the above transformed instance can be bounded as follows: $N = \Theta(\log_\alpha^2 n \cdot \log_\tau n)$, $M = \Theta(\log_\alpha^4 n \cdot \log_\tau n)$, $U \leq n$ and, finally, $C = \mathcal{O}(n^{c+1})$. Plugging in the values in the algorithm by Ahuja *et al.* [4] with running time $\mathcal{O}(MN \log\log U \log NC)$ gives that the max-flow min-cost problem can be solved in time $\mathcal{O}(\log^9 n \cdot \log\log n)$.

Finally, if we are not satisfied with an estimated cost but actually want a packing then we just pair up the rectangles with the matching cells. This is trivially done in time linear with respect to the number of rectangles in the packing. □

The minimum-cost circulation problem has a rich history and the interested reader is encouraged to read [5]. In 1989, Goldberg and Tarjan [75] showed a simple algorithm that solves the minimum-cost circulation problem in time $\mathcal{O}(N^2M^3 \log N)$. There are also other types of algorithms that use capacity scaling, for example the algorithm by Edmonds and Karp [58] with a running time of $\mathcal{O}((M \log U)(M + N \log N))$.

7.4.2 Weighted variants

In this section we consider two weighted variants of the grid packing problem, i.e., every rectangle $r \in \mathcal{S}$ also has a weight $w(r)$. Both results follow from Corollaries 7.5-7.8 and Lemma 7.18. Recall that $f(\chi) = \frac{2c\log(\chi/(\sqrt{\chi}-1))}{\log^2 \chi}$.

Problem 11 [Minimum area weighted grid packing(MAwGP)] *Given a set of n rectangles and a real number w compute a minimum area grid packing containing a set of rectangles of $\mathcal{S}$ of total weight at least w.*

Theorem 7.19 *Given a set $\mathcal{S}$ of n rectangles and a real value $\varepsilon > 0$, one can compute a grid packing containing rectangles of total weight at least*

w whose area is at most $(1+\varepsilon)$ greater than the area of a minimum area (τw)-grid packing in time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon/7})} \cdot \log^9 n \cdot \log\log n + n)$.

Problem 12 [Maximum weight wafer packing (MwWP)] *Given a set of rectangles $\mathcal{S}$ and a rectangular region $\mathcal{A}$ compute a grid packing of $\mathcal{S}' \subseteq \mathcal{S}$ onto $\mathcal{A}$ of maximum weight.*

Let $\kappa(\mathcal{S}, \mathcal{A})$ denote the weight of a maximum weight grid packing of $\mathcal{S}$ onto $\mathcal{A}$. Let $\mathcal{A}'$ be a rectangular region such that the sides of $\mathcal{A}$ is $(1+\varepsilon)$ longer than the sides of $\mathcal{A}'$.

Theorem 7.20 *Given a set $\mathcal{S}$ of n rectangles, a rectangular region $\mathcal{A}$ and a real value $\varepsilon > 0$, one can compute a grid packing of $\mathcal{S}$ onto $\mathcal{A}$ whose weight is at least $\kappa(\mathcal{S}, \mathcal{A}')/\tau$ in time $\mathcal{O}(n^{f(1+\varepsilon/7)} \cdot \log^9 n \cdot \log\log n + n)$.*

7.5 Hardness results

In this section we show that several of the problems considered are $\mathcal{NP}$-hard. We start with an interesting observation:

Observation 7.21 *There exists a set $\mathcal{S}$ of n rectangles such that every optimal grid packing of $\mathcal{S}$ on a wafer (rectangular region) $\mathcal{A}$ induces a grid with $\Omega(n^2)$ cells.*

Proof: Let $\mathcal{A}$ be a square with side length L, and $\mathcal{S}$ consists of one very large square, with side length $L - \varepsilon$ and $n-1$ thin long squares with side length $L - \varepsilon$ and $2\varepsilon/(n-1)$. The entire set $\mathcal{S}$ can only be packed onto $\mathcal{A}$ in one single way which induces $\Omega(n^2)$ cells. □

Theorem 7.22 *MNWP cannot be approximated within a factor of $3/2 - \varepsilon$ for any $\varepsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$.*

Proof: An instance for the BinPacking is a finite set of items U, a size $s(u) \in Z^+$ for each $u \in U$ and a positive integer capacity B. A solution to the problem is a partition of U into disjoint sets $U_1, \ldots, U_m$ such that the sum of the items in each U_i is B or less.

The transformation is straight-forward. For each $u \in U$ produce a rectangle r of height $s(u)$ and width 1. Finally, set the width of a wafer to be 1, and the height to be B. Solve the Minimum number of wafer-problem. The

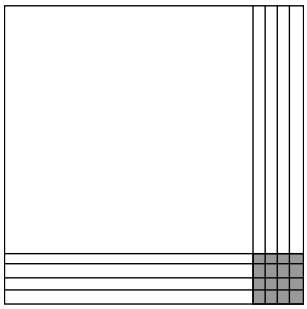

Figure 7.4: There are $\Omega(n^2)$ shaded squares which all are empty.

solution is also a solution for the BinPacking-problem and since this cannot be approximated within a factor of $3/2 - \varepsilon$ for any $\varepsilon > 0$ in polynomial time [**?**] the same inapproximability result holds for MNWP. □

Theorem 7.23 *The MWP-problem is $\mathcal{NP}$-hard.*

Proof: Let MWP′ be the decision version of MWP: given a set of rectangles $\mathcal{S}$, a rectangular plate $\mathcal{A}$ and an integer $\mathcal{L}$ can we grid pack at least $\mathcal{L}$ rectangles of $\mathcal{S}$ onto $\mathcal{A}$?

In order to show the reduction we need the decision version of the Partition problem: Given integers $a = \{a_1 \leq \ldots \leq a_n\}$, does there exist a subset $P \subseteq I = \{1, 2, \ldots, n\}$ such that $\sum_{j \in P} a_j = \sum_{j \in I \setminus P} a_j$? This problem is $\mathcal{NP}$-complete [131].

We show that the Partition-problem reduces in polynomial time to the MWP′ problem. Hence, given a Partition instance $(a_1, \ldots, a_n)$, we create a MWP′ instance $(\mathcal{S}, \mathcal{A}, \mathcal{L})$ such that $\mathcal{L} = n + 1$, as shown in Fig. 7.4, with some minor differences. That is, $\mathcal{A}$ is created as a square with sides $w + \delta$, where $\delta = \sum_{i=1}^{n} a_i/2$ and w is arbitrarily chosen larger than the diagonal of a $\delta \times \delta$ square ($w > \sqrt{2}\delta$). The set $\mathcal{S}$ is created to contain one large square with sides w, and n rectangles, one for each integer a_i, with sides w and a_i. Since we create $n + 1$ rectangles in total, it is clear that we have a polynomial time reduction.

This reduction is valid if we can show that all rectangles of $\mathcal{S}$ can be grid packed on exactly one plate $\mathcal{A}$, if, and only if, the original Partition instance is a yes instance. In order to be able to pack all of $\mathcal{S}$ on $\mathcal{A}$ the following grid is needed. First we need a column and a row with width and height, respectively, w. These define a cell in which the large square may

be packed. As for the remaining rectangles it is clear that no such rectangle can be packed in the $\delta \times \delta$ lower right corner, which is formed when the large square is packed. This follows since these rectangles have a side which is longer than the diagonal of this corner ($w > \sqrt{2}\delta$). This means that we need either exactly one column, or exactly one row, with width, respectively height, a_i, for each rectangle corresponding to an integer a_i. Note that each such column or row defines a cell with sides w and a_i, in which the corresponding rectangle can be packed. Since $\delta = \sum_{i=1}^{n} a_i/2$ it is clear that all these columns and rows can be created, and thus all of $\mathcal{S}$ packed on $\mathcal{A}$, if, and only if, the original PARTITION instance is a 'yes'-instance. □

Theorem 7.24 *The MAGPAR-problem is $\mathcal{NP}$-hard.*

Proof: The proof is almost identical to the proof of Theorem 7.23. □

7.6 Final comments

The approximation algorithm can be generalized to d dimensions, such that the resulting grid packing has sides that are at most $(1 + \varepsilon)$ times longer than the length of the sides of an optimal grid packing. The running time is $\mathcal{O}(dn \log n)$ for sorting the d-dimensional rectangles into bins. The size of the coding becomes $\mathcal{O}(d \log n)$, which implies that the number of grids is $n^{dc \cdot f(\alpha,\beta,\gamma)}$. The flow network will have $\mathcal{O}(\log^{d+1} n)$ nodes and $\mathcal{O}(\log^{2d+1} n)$ arcs, which implies that the running time of the max-flow algorithm will be $\mathcal{O}(\log^{3d+3} n)$.

7.7 Acknowledgements

We thank Esther and Glenn Jennings for introducing us to the problem. We would also like to thank Bernd Gärtner for pointing us to "The table layout problem" [11, 20, 146, 161].

Finally, we thank the anonymous referees of CGTA (Computational Geometry, Theory and Applications) for pointing out some errors of an older journal version on which this chapter was based, as well as helping us to improve the presentation of our results.

Bibliography

[1] *GNU Triangulated Surface Library*.

[2] M. Ali Abam, M. de Berg, M. Farshi, and J. Gudmundsson. Region-fault tolerant geometric spanners. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, 2007.

[3] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66(1):207–243, 2003.

[4] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. Finding minimum-cost flows by double scaling. *Mathematical Programming*, 53(3):243–266, 1992.

[5] R. K. Ahuja, T. L. Magnanti, , and J. B. Orlin. *Algorithms for cutting sheets of metal*. Prentice Hall, New Jersey, 1993.

[6] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

[7] G. Al-Naymat, S. Chawla, and J. Gudmundsson. Dimensionality reduction for long duration and complex spatio-temporal queries. In *Proceedings of the 22nd ACM Symposium on Applied Computing*, pages 393–397. ACM, 2007.

[8] N. Alon and Y. Azar. On-line Steiner trees in the Euclidean plane. *Discrete & Computational Geometry*, 10:113–121, 1993.

[9] G. Aloupis, P. Bose, and P. Morin. Reconfiguring triangulations with edge flips and point moves. *Algorithmica*, 47(4):367–378, 2007.

[10] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[11] R. J. Anderson and S. Sobti. The table layout problem. In *proceedings of Symposium on Computational Geometry*, pages 115–123, 1999.

[12] M. Andersson, J. Gudmundsson, P. Laube, and T. Wolle. Reporting leaders and followers among trajectories of moving point objects. Technical Report PA006075, National ICT Australia, 2006. `http://www.nicta.com.au`, Extended abstract in *Proceedings of the 22nd ACM Symposium on Applied Computing*, pages 3–7. ACM, 2007.

[13] K. Andreev and H. Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[14] S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of the 4th European Symposium on Algorithms*, volume 1136 of *Lecture Notes in Computer Science*, pages 514–528, 1996.

[15] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55:197–218, 1994.

[16] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 489–498, 1995.

[17] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45(6):891–923, 1998.

[18] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17:33–54, 1997.

[19] S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006.

[20] R.J. Beach. *Setting tables and illustrations with style*. PhD thesis, Dept. of computer science, University of Waterloo, Canada, 1985.

[21] R. I. Becker, S. R. Schach, and Y. Perl. A shifting algorithm for min-max tree partitioning. *Journal of the Association for Computing Machinery*, 29(1):58–67, 1982.

[22] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, volume 1776 of *Lecture Notes In Computer Science*, pages 88–94, London, UK, 2000. Springer-Verlag.

[23] M. Benkert, J.Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. In *Proceedings of the 14th European Symposium on Algorithms (ESA 2006)*, volume 4168 of *Lecture Notes in Computer Science*, pages 660–671. Springer, 2006.

[24] P. Bose, J. Czyzowicz, P. Morin, J. Gao, and D. Wood. Simultaneous diagonal flips in plane triangulations. *Journal of Graph Theory*, 54(4):307–330, 2007.

[25] P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry - Theory and Applications*, 28:11–18, 2004.

[26] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.

[27] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15(3):211–228, 2006.

[28] S. M. C. Cavalcanti and F. F. Knowlton. Evaluation of physical and behavioral traits of llamas associated with aggressiveness toward sheep-threatening canids. *Applied Animal Behaviour Science*, 61(2):143–158, 1998.

[29] B. Chandra. Constructing sparse spanners for most graphs in higher dimensions. *Information Processing Letters*, 51(6):289–294, 1994.

[30] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *International Journal of Computational Geometry and Applications*, 5:124–144, 1995.

[31] B. Chandra, H. J. Karloff, and C. A. Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28(6):1998–2029, 1999.

[32] D. Z. Chen. On the all-pairs Euclidean short path problem. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 292–301, 1995.

[33] D. Z. Chen, K. S. Klenk, and H.-Y. T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM Journal on Computing*, 29(4):1223–1246, 2000.

[34] S.-W. Cheng, T.K. Dey, and S.-H. Poon. Hierarchy of surface models and irreducible triangulations. *Computational Geometry: Theory and Applications*, 27(2):135–150, 2004.

[35] Y.-J. Chiang and J. S. B. Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 215–224, 1999.

[36] N. R. Chrisman. Beyond the Snapshot: Changing the Approach to Change, Error, and Process. In M. J. Egenhofer and R. G. Golledge, editors, *Spatial and Temporal Reasoning in Geographic Information Systems*, pages 85–93. Oxford University Press, Oxford, UK, 1998.

[37] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal of Computing*, 28:210–236, 1998.

[38] L. Conradt and T. J. Roper. Group decision-making in animals. *Nature*, 421(6919):155–158, 2003.

[39] A. Czumaj and H. Zhao. Fault-tolerant geometric spanners. *Discrete and Computional Geometry*, 32(2):207–230, 2004.

[40] G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proceedings of the 9th Annual ACM Symposium on Computational Geometry*, pages 53–62, 1993.

[41] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7:297–315, 1997.

[42] G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 215–222, 1995.

[43] M. D'Auria, M. Nanni, and D. Pedreschi. Time-focused density-based clustering of trajectories of moving objects. In *Proceedings of the Workshop on Mining Spatio-temporal Data (MSTD-2005)*, Porto, 2005.

[44] M. de Berg and K. Dobrindt. On levels of detail in terrains. In *Proceedings of the 11th annual Symposium on Computational Geometry*, pages 426–427. ACM Press, 1995.

[45] M. de Berg, J. Gudmundsson, M. Katz, C. Levcopoulos, M. Overmars, and F. van der Stappen. Constant factor approximation algorithms for TSPN with fat objects. In *Proc. 10th Anual European Symposium on Algorithms*, pages 187–199, 2002.

[46] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[47] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959.

[48] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[49] D. H.-C. Du, I. Lin, and K. C. Chang. On wafer-packing problems. *IEEE Transactions on Computers*, 42(11):1382–1388, 1993.

[50] C. Du Mouza and P. Rigaux. Mobility patterns. *GeoInformatica*, 9(4):297–319, 2005.

[51] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. In *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 38–46, 2001.

[52] B. Dumont, A. Boissy, C. Achard, A. M. Sibbald, and H. W. Erhard. Consistency of animal order in spontaneous group movements allows the measurement of leadership in a group of grazing heifers. *Applied Animal Behaviour Science*, 95(1-2):55–66, 2005.

[53] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Planarity-preserving clustering and embedding for large planar graphs. *Computational Geometry: Theory and Applications*, 24(2):95–114, 2003.

[54] H. Dyckhoff. Typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[55] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and packing. *Annotated Bibliographies in Combinatorial Optimization, eds: M. Dell'Amico, F. Maffioli and S. Martello Eds*, pages 393–412, 1997.

[56] J. A. Dykes and D. M. Mountain. Seeking structure in records of spatio-temporal behaviour: visualization issues, efforts and application. *Computational Statistics and Data Analysis*, 43(4):581–603, 2003.

[57] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[58] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[59] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers, Amsterdam, 2000.

[60] D. Eppstein and D. Hart. An efficient algorithm for shortest paths in vertical and horizontal segments. In *In Proc. 5th Workshop on Algorithms and Data Structures*, pages 234–247, 1997.

[61] D. Eppstein and D. Hart. Shortest paths in an arrangement with k line orientations. In *In Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 310–316, 1999.

[62] J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete & Computational Geometry*, 13:41–57, 1995.

[63] M. Erwig and R. H. Güting. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3):269–296, 1999.

[64] M. Farshi and J. Gudmundsson. Experimental study of geometric t-spanners. In *Proceedings of the 13th European Symposium on Algorithms*, volume 3669 of *Lecture Notes in Computer Science*, pages 556–567, 2005.

[65] M. Farshi and J. Gudmundsson. Experimental study of geometric t-spanners: a running time comparison. In *Proceedings of 6th Workshop on Experimental Algorithms*, volume 4525, pages 270–284, 2007.

[66] L. De Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Proceedings of the 8th conference on Visualization*, pages 103–110. IEEE Computer Society Press, 1997.

[67] A. U. Frank. Socio-Economic Units: Their Life and Motion. In A. U. Frank, J. Raper, and J. P. Cheylan, editors, *Life and motion of socio-economic units*, volume 8 of *GISDATA*, pages 21–34. Taylor & Francis, London, 2001.

[68] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[69] G. N. Fredrickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[70] A. Gajentaan and M. H. Overmars. n^2-hard problems in computational geometry. Technical Report 1993-15, Department of Coumputer Science, Utrecht University, The Netherlands, 1993.

[71] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company, San Fransisco, 1979.

[72] M. Garland. Multiresolution modeling: Survey & future opportunities. In *Eurographics'99 – State of the Art Reports*, pages 111–131, 1999.

[73] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31:209–216, 1997.

[74] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, 1998.

[75] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by cancelling negative cycles. *Journal of the ACM*, 36(4):873–886, 1989.

[76] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. Manuscript, July 2007.

[77] J. Gudmundsson and C. Levcopoulos. A fast approximation algorithm for TSP with neighborhoods. *Nordic Journal of Computing*, 6:469–488, 1999.

[78] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479–1500, 2002.

[79] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graphs. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 828–837, 2002.

[80] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles revisited. In *Proceedings of the 13th International Symposium on Algorithms and Computation*, volume 2518 of *Lecture Notes in Computer Science*, pages 357–368. Springer-Verlag, 2002.

[81] J. Gudmundsson, G. Narasimhan, and M. Smid. Fast pruning of geometric spanners. In *Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, 2005.

[82] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *Proceedings of the 14th ACM Symposium on Advances in GIS*, pages 35–42, 2006.

[83] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal sets. *GeoInformatica*, 11(2):195–215, 2007.

[84] Joachim Gudmundsson and Christian Knauer. Dilation and detour in geometric networks. In Teofilo Gonzalez, editor, *Handbook on approximation algorithms and metaheuristics*. Chapman & Hall/CRC, Amsterdam, 2007.

[85] S. Gueron and S. A. Levin. Self-Organization of Front Patterns in Large Wildebeest Herds. *Journal of Theoretical Biology*, 165(4):541–552, 1994.

[86] S. Gumhold, P. Borodin, and R. Klein. Intersection free simplification. *International Journal of Shape Modeling*, 9(2):155–176, 2003.

[87] R. Güting, M. H. Boehlen, M. Erwig, C. S. Jensen, N. Lorentzos, E. Nardelli, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems (TODS)*, 2520(1):1–42, 2000.

[88] R. Güting, M. H. Boehlen, M. Erwig, C. S. Jensen, N. Lorentzos, E. Nardelli, M. Schneider, and J. R. R. Viqueira. Spatio-temporal

Models and Languages: An Approach Based on Data Types. In M. Koubarakis, T. Sellis, A. U. Frank, S. Grumbach, R. H. Gueting, C. S. Jensen, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H. J. Schek, M. Scholl, B. Theodoulidis, and N. Tryfona, editors, *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, volume 2520 of *LNCS*, pages 117–176. Springer, Berlin, 2003.

[89] R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, 2005.

[90] N. Guttmann-Beck and R. Hassin. Approximation algorithms for min-max tree partition. *Journal of Algorithms*, 24(2):266–286, 1997.

[91] H. Hoppe. Progressive meshes. *Computer Graphics*, 30:99–108, 1996.

[92] I. A. R. Hulbert. GPS and its Use in Animal Telemetry: The next five Years. In A. M. Sibbald and I. J. Gordon, editors, *Proceedings of the Conference on Tracking Animals with GPS*, pages 51–60, Aberdeen, UK, 2001. Macaulay Insitute.

[93] Y. Inada and K. Kawachi. Order and flexibility in the motion of fish schools. *Journal of theoretical Biology*, 214(3):371–387, 2002.

[94] Y. Ishikawa, Y. Tsukamoto, and H. Kitagawa. Extracting mobility statistics from indexed spatio-temporal datasets. In *Proc. of the 2nd Workshop on Spatio-Temporal Database Management (STDBM)*, pages 9–16, 2004.

[95] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[96] K. Jansen and G. Zhang. On rectangle packing: maximizing benefits. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 204–213, 2004.

[97] G. Jennings. Personal communication. 2002.

[98] T. Kapler, R. Harper, and W. Wright. Correlating events with tracked movement in time and space: A geotime case study, 2005. Presented at the 2005 Intelligence Analysis Conference, Washington, DC.

[99] D. Karger, C. Stein, and J. Wein. *Scheduling algorithms, In Handbook on Algorithms and Theory of Computation*. CRC Press, edited by M. J. Atallah, 1998.

[100] J. M. Keil. Approximating the complete Euclidean graph. In *Proceedings of the 1st Scandinavian Workshop on Algorithmic Theory*, volume 382 of *Lecture Notes in Computer Science*, pages 208–213, 1988.

[101] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.

[102] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[103] D.E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, 1976.

[104] G. Kollios, S. Sclaroff, and M. Betke. Motion mining: discovering spatio-temporal patterns in databases of human motion. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[105] M. Koubarakis, Y. Theodoridis, and T. Sellis. Spatio-Temporal Databases in the Years Ahead. In M. Koubarakis, T. Sellis, A. U. Frank, S. Grumbach, R. H. Gueting, C. S. Jensen, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H. J. Schek, M. Scholl, B. Theodoulidis, and N. Tryfona, editors, *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, volume 2520 of *LNCS*, pages 345–347. Springer, Berlin, 2003.

[106] J. Krause and G. D. Ruxton. *Living in Groups*. Oxford Series in Ecology and Evolution. Oxford University Press, New York, NY, 2002.

[107] D. Krznaric and C. Levcopoulos. Computing hierarchies of clusters from the Euclidean minimum spanning tree in linear time. In *In Proc. 15th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 443–455, 1995.

[108] D. Krznaric and C. Levcopoulos. Optimal algorithms for complete linkage clustering in d dimensions. *Theoretical Computer Science*, 286(1):139–149, 2002.

[109] M. P. Kwan. Interactive geovisualization of activity-travel patterns using three dimensional geographical information systems: A methodological exploration with a large data set. *Transportation Research Part C*, 8(1–6):185–203, 2000.

[110] R. F. Lachlan, L. Crooks, and K. N. Laland. Who follows whom? Shoaling preferences and social learning of foraging information in guppies. *Animal Behaviour*, 56(1):181–190, 1998.

[111] P. Laube and S. Imfeld. Analyzing relative motion within groups of trackable moving point objects. In M. J. Egenhofer and D. M. Mark, editors, *Geographic Information Science 2002*, volume 2478 of *Lecture Notes in Computer Science*, pages 132–144, Berlin, 2002. Springer.

[112] P. Laube, S. Imfeld, and R. Weibel. Discovering relative motion patterns in groups of moving point objects. *International Journal of Geographical Information Science*, 19(6):639–668, 2005.

[113] P. Laube and R.S. Purves. An approach to evaluating motion pattern detection techniques in spatio-temporal data. *Computers, Environment and Urban Systems*, 30(3):347–374, 2006.

[114] P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In P. F. Fisher, editor, *Developments in Spatial Data Handling: Proceedings of the 11th International Symposium on Spatial Data Handling*, pages 201–214, Berlin, 2004. Springer.

[115] J. B. Leca, N. Gunst, B. Thierry, and O. Petit. Distributed leadership in semifree-ranging white-faced capuchin monkeys. *Animal Behaviour*, 66:1045–1052, 2003.

[116] H.-P. Lenhof, J. S. Salowe, and D. E. Wrege. New methods to mix shortest-path and minum spanning trees. Manuscript, 1993.

[117] C. Levcopoulos and A. Lingas. There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. *Algorithmica*, 8:251–256, 1992.

[118] C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32:144–156, 2002.

[119] Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, Seattle, WA, USA, 2004. ACM Press.

[120] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:379–396, 2002.

[121] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *Proceedings of the 10th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, pages 236–245. ACM, 2004.

[122] D. M. Mark. Geospatial Lifelines. In *Integrating Spatial and Temporal Databases*, volume 98471. Dagstuhl Seminars, 1998.

[123] C. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proc. 11th Annual ACM Symposium on Computational Geometry*, pages 360–369, 1995.

[124] Nikko Materials. *Compound Semiconductor Wafers*. www.nikkomaterials.com/compound.htm, April 2004.

[125] H. J. Miller and J. Han. Geographic data mining and knowledge discovery: An Overview. In H. J. Miller and J. Han, editors, *Geographic data mining and knowledge discovery*, pages 3–32. Taylor & Francis, London, UK, 2001.

[126] J. S. B. Mitchell. *Shortest paths and networks*. CRC Press LLC, 1997.

[127] J. S. B. Mitchell. A fast approximation algorithm for TSP with neighborhoodguillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[128] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *Proceedings of the 2005 ACM SIGMOD Conference on Management of Data*, pages 634–645, 2005.

[129] G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[130] R. T. Ng. Detecting outliers from large datasets. In H. J. Miller and J. Han, editors, *Geographic data mining and knowledge discovery*, pages 218–235. Taylor & Francis, London, UK, 2001.

[131] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, NY, 1998.

[132] R. O. Peterson, A. K. Jacobs, T. D. Drummer, L. D. Mech, and D. W. Smith. Leadership behavior in relation to dominande and reproductive status in gray wolves, *canis lupus*. *Canadian Journal of Zoology*, 80(8):1405–1412, 2002.

[133] F. Porikli. Trajectory Distance Metric Using Hidden Markov Model based Representation. In *Proceedings of the 6th IEEE European Conference on Computer Vision, Workshop on PETS*, Prague, 2004.

[134] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[135] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. In *Seventh international conference on Information and knowledge management*, pages 251–258, Bethesda, Maryland, United States, 1998. ACM Press.

[136] R. Raman. Recent results on the single-source shortest paths problem. *SIGACT News*, 28:81–87, 1997.

[137] S. A. Rands, G. Cowlishaw, R. A. Pettifor, J. M. Rowcliffe, and R. A. Johnstone. Spontaneous emergence of leaders and followers in foraging pairs. *Nature*, 423(6938):432–434, 2003.

[138] J. Raper. The Dimensions of GIScience, 2002. Keynote speech of GIScience 2002.

[139] J. H. Reif and S. Sen. *Parallel computational geometry: An approach using randomization, In Handbook of Computational Geometry*. Elsevier Science Publishers B.V. North-Holland, edited by J.-R. Sack and J. Urrutia, 2000.

[140] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, volume 21, pages 25–34. ACM Press, 1987.

[141] J. F. Roddick, K. Hornsby, and M. Spiliopoulou. An Updated Bibliography of Temporal, Spatial, and Spatio-temporal Data Mining Research. In J. F. Roddick and K. Hornsby, editors, *Temporal, spatial*

and spatio-temporal data mining, TSDM 2000, volume 2007 of *Lecture Notes in Artificial Intelligence*, pages 147–163, Berlin, 2001. Springer.

[142] J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry and Applications*, 1(2):99–107, 1991.

[143] I. Schiermeyer. Reverse-fit: a 2-optimal algorithm for packing rectangles. In *In Proc. 2nd Annual European Symposium on Algorithms*, pages 290–299. Lecture Notes in Computer Science 855, Springer-Verlag, 1994.

[144] Virginia semiconductors. www.virginiasemi.com/thinfilm.cfm, April 2004.

[145] B. Shaleooi. *Algoritmer för plåtskärning (*Eng. transl. *Algorithms for cutting sheets of metal).* LUNDFD6/NFCS-5189/1–44/2001, Master thesis, Department of Computer Science, Lund University, Sweden, 2001.

[146] K.-H. Shin, K. Kobayashi, and A. Suzuki. Tafel musik: Formatting algorithm of tables. *Mathematical and Computer Modelling*, 26(1):97–112, 1997.

[147] T. Shirabe. Correlation analysis of discrete motions. In *Proceedings of the Fourth International Conference on Geographic Information Science, GIScience 2006*, volume 4197 of *Lecture Notes In Computer Science*, pages 370–382, Berlin, 2006. Springer-Verlag.

[148] G. Sinha and D. M. Mark. Measuring similarity between geospatial lifelines in studies of environmental health. *Journal of Geographical Systems*, 7(1):115–136, 2005.

[149] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *13th International Conference on Data Engineering (ICDE13)*, 1997.

[150] M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science Publishers, Amsterdam, 2000.

[151] C. S. Smyth. Mining mobile trajectories. In H. J. Miller and J. Han, editors, *Geographic data mining and knowledge discovery*, pages 337–361. Taylor & Francis, London, UK, 2001.

[152] J. Soares. Approximating Euclidean distances by small degree graphs. *Discrete & Computational Geometry*, 11:213–233, 1994.

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

[154] James J. Thomas and Kristin A. Cook. A visual analytics agenda. *IEEE Computer Graphics and Applications*, 26(1):10–13, 2006.

[155] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999.

[156] M. Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms*, 35(2):189–201, 2000.

[157] M. Thorup and U. Zwick. Approximate distance oracles. In *In Proc. 33rd ACM Symposium on Theory of Computing*, pages 183–192, 2001.

[158] S. Tisue and U. Wilensky. NetLogo: A Simple Environment for Modeling Complexity. In *International Conference on Complex Systems*, Boston, 2004.

[159] P. M. Vaidya. A sparse graph almost as good as the complete graph on points in K dimensions. *Discrete & Computational Geometry*, 6:369–381, 1991.

[160] F. Verhein and S. Chawla. Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA)*, volume 3882 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2006.

[161] X. Wang and D. Wood. Tabular formatting problems. In *In Proc. of the 3rd International Workshop on Principles of Document Processing*, pages 181–191. Lecture Notes in Computer Science 1293, Springer-Verlag, 1996.

[162] U. Wilensky. NetLogo Flocking model, 1998. `http://ccl.northwestern.edu/netlogo/models/Flocking`.

[163] U. Wilensky. NetLogo (and NetLogo User Manual), 1999. `http://ccl.northwestern.edu/netlogo`.

[164] O. Wolfson and E. Mena. Applications of Moving Objects Databases. In Y. Manolopoulos, A. Papadopoulos, and M. Vassilakopoulos, editors, *Spatial Databases: Technologies, Techniques and Trends*. Idea group Co., 2004.

[165] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In M. Rafanelli and M. Jarke, editors, *10th International Conference on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998*, pages 111–131. IEEE Computer Society, 1998.

[166] J. C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of the 7th conference on Visualization*, pages 327–334. IEEE Computer Society Press, 1996.

[167] X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proc. of the 21st International Conference on Data Engineering (ICDE 2005)*, pages 643–654, 2005.