Approximation Algorithms for Network Design Problems

Inaugural-Dissertation
zur
Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von
Anna Schulze
aus Herford

Köln 2008

Berichterstatter: Prof. Dr. R. Schrader
PD Dr. H. Randerath

Tag der mündlichen Prüfung: 17. Oktober 2008

Zusammenfassung

Diese Arbeit beschäftigt sich mit Netzwerkdesign-Problemen. Gegeben ist eine Menge von Punkten in der Ebene. Gesucht wird nach einer Menge von Kanten mit minimaler Gesamtlänge, die alle Punkte miteinander verbindet. In dieser allgemeinen Formulierung ist das Problem als Steinerbaum-Problem bekannt. Wir betrachten in dieser Arbeit oktilineare Steinerbäume mit harten und weichen Blockaden. Ein oktilinearer Steinerbaum darf Kanten enthalten, die in horizontaler, vertikaler oder diagonaler Richtung verlaufen. Eine Blockade ist eine zusammenhängende Region in der Ebene, die durch ein einfaches Polygon berandet wird. Keine Kante eines oktilinearen Steinerbaums darf im Inneren einer harten Blockade liegen. Wenn wir einen Steinerbaum mit dem Inneren einer weichen Blockade schneiden, dann darf keine der sich daraus ergebenden Zusammenhangskomponenten länger als eine vorgegebene Länge sein. Wir führen polynomielle Approximationsschemata für das oktilineare Steinerbaum-Problem mit harten und weichen Blockaden ein. Für diese Probleme waren dies die ersten vorgestellten Approximationsschemata. Zusätzlich geben wir noch einen 2-Approximationsalgorithmus für das Problem mit weichen Blockaden an.

Danach beschäftigen wir uns mit euklidischen Gruppensteinerbäumen. Hierbei hat man anstatt einer festen Menge von Punkten für jeden Punkt eine Menge von möglichen Positionen. Diese möglichen Positionen werden zu Gruppen zusammengefasst. Wir betrachten den Fall, dass die Gruppen innerhalb disjunkter Regionen liegen, die alle die spezielle Eigenschaft der sogenannten α-Dicke erfüllen. Grob gesagt spezifiziert der Begriff α-Dicke die Form einer Region im Vergleich zum Kreis. Wir führen den ersten Approximationsalgorithmus für dieses Problem ein und erreichen eine Approximationsgüte von $(1 + \varepsilon)(9.093\alpha + 1)$.

Als letztes betrachten wir Manhattan-Netzwerke. Sie dürfen Kanten enthalten, die in horizontaler und vertikaler Richtung verlaufen. Im Vergleich zu Steinerbäumen enthalten sie zusätzlich einen kürzesten Weg zwischen je zwei Punkten. Wir führen drei neue Approximationsalgorithmen für das Manhattan-Netzwerk-Problem ein, den ersten mit Approximationsgüte 3 und zwei Algorithmen mit Güte 2. Für diese Algorithmen benötigen wir Algorithmen, die das Manhattan-Netzwerk-Problem für Treppen lösen. Wir geben zwei Algorithmen für dieses spezielle Problem an. Der erste löst Manhattan-Netzwerke für Treppen optimal, der zweite erreicht eine Approximationsgüte von 2. Ähnliche Ansätze wurden vorher schon diskutiert. Da wir eine etwas andere Definition von Treppen benutzen und wir zusätzlich spezielle Eigenschaften brauchen, die unsere Treppen erfüllen, haben wir diese Ansätze auf unseren Fall übertragen. Die 2-Approximationsalgorithmen für allgemeine Manhattan-Netzwerke erreichen die bisher beste bekannte Approximationsgüte.

Abstract

We consider different variants of network design problems. Given a set of points in the plane we search for a shortest interconnection of them. In this general formulation the problem is known as Steiner tree problem. We consider the special case of octilinear Steiner trees in the presence of hard and soft obstacles. In an octilinear Steiner tree the line segments connecting the points are allowed to run either in horizontal, vertical or diagonal direction. An obstacle is a connected region in the plane bounded by a simple polygon. No line segment of an octilinear Steiner tree is allowed to lie in the interior of a *hard* obstacle. If we intersect a Steiner tree with the interior of a *soft* obstacle, no connected component of the induced subtree is allowed to be longer than a given fixed length. We provide polynomial time approximation schemes for the octilinear Steiner tree problem in the presence of hard and soft obstacles. These were the first presented approximation schemes introduced for the problems. Additionally, we introduce a $(2 + \varepsilon)$-approximation algorithm for soft obstacles.

We then turn to Euclidean group Steiner trees. Instead of a set of fixed points we get for each point a set of potential locations (combined into groups) and we need to pick only one location of each group. The groups we consider lie inside disjoint regions fulfilling a special property so-called α-fatness. Roughly speaking, the term α-fat specifies the shape of the region in comparison to a disk. We give the first approximation algorithm for this problem and achieve an approximation ratio of $(1 + \varepsilon)(9.093\alpha + 1)$.

Last, we consider Manhattan networks. They are allowed to contain edges only in horizontal and vertical direction. In contrast to Steiner trees they contain a shortest path between each pair of points. We introduce insights into the structure of Manhattan networks, particularly in the context of so-called staircases. We give three new approximation algorithms for the Manhattan network problem, the first with approximation ratio 3 and two algorithms with ratio 2. To this end we introduce two algorithms for the Manhattan network problem of staircases. The first algorithm solves the problem to optimality the second yields a 2-approximation. Variants of both algorithms are already known in the literature. Since we use a slightly different definition of staircases and we need special properties of them, we adopt the algorithms to our situation. The 2-approximation algorithms achieve the best known approximation ratio of an algorithm for the Manhattan network problem known so far. Last we give an idea how we could possibly find an algorithm with better approximation ratio.

Danksagung

Als erstes möchte ich mich bei Prof. Dr. Schrader bedanken, dass er mir die Möglichkeit gab, bei ihm zu promovieren. Die Zusammenarbeit mit ihm hat mir immer sehr viel Freude bereitet. Hervorheben und bedanken möchte ich mich auch für die Möglichkeit, das Thema dieser Arbeit selbst zu bestimmen und mir dabei größtmögliche Freiheit einzuräumen. Weiterhin möchte ich sehr herzlich PD Dr. Randerath für die Übernahme des Zweitgutachtens danken.

Das erste Thema, mit dem ich mich beschäftigte, waren oktilineare Steinerbäume. Ich verdanke es Prof. Dr. Matthias Müller-Hannemann, dass er mir diese phaszinierende Welt zwischen Graphentheorie und algorithmischer Geometrie eröffnete. Vor allem möchte ich ihm aber dafür danken, dass er mir beibrachte, sich Probleme zu suchen und Fragen zu stellen. Von ihm habe ich auch gelernt, ein wenig einzuschätzen, was man selbst leisten kann und was nicht. Bedanken möchte ich mich auch bei Dr. Bernhard Fuchs, der mir und den Manhattan-Netzwerken keine Ruhe ließ. Ohne ihn und seine immer neuen Anstöße hätte ich die Manhattan-Netzwerke und die Probleme, die dort auftauchten schwer lösen können.

Weiterhin möchte ich mich auch sehr herzlich bei Prof. Dr. Schrader und Prof. Dr. Faigle bedanken, dass ich in ihrer Arbeitsgruppe mitarbeiten konnte. Sie ermöglichten mir, dass diese Arbeit entstehen konnte. Ein weiterer Dank gilt hierbei auch der gesamten Arbeitsgruppe. Vor allem erwähnen und danken möchte ich Birgit Engels und Daniel Herrmann. Birgit für die gesamte Zeit, die wir zusammen am ZAIK verbracht haben, Daniel vor allem für das unterstützende unter die Arme greifen in den letzten Wochen.

Meinen Eltern und meiner Familie möchte ich dafür danken, dass sie sich vollkommen unverantwortlich für diese Arbeit fühlen und ihnen hier an dieser Stelle kein Dank dafür gebürt. Sie vermittelten mir seit ich denken kann, dass ich für das, was ich tue, selbst verantwortlich bin.

Contents

Chapter 1

Introduction

1.1 Motivation

In this thesis, we consider variants of network design problems. Given a set of points (called *terminals*) in the plane we search for a shortest interconnection of these. This problem is one of the variants of the Steiner tree problem. See Figure 1.1 for different examples of Steiner trees. Historically, Fermat (1601–1665) was the first who considered a Steiner tree problem. He posed the following question: *"Given three points in the plane, find a fourth point such that the sum of its distances to the three given points is minimum."* Torricelli found a solution for Fermat's problem with compass and ruler before 1640. The generalization of the problem to n given points for which we search for a point which minimizes the sum of the distances to the n points was considered by many researchers. One of them was the mathematician Jacob Steiner (1796–1863). Courant and Robbins [CR41] referred to Steiner in their popular book *"What is Mathematics?"*, establishing the notion "Steiner tree problem". In 1934 Jarník and Kössler [JK34] were the first who investigated the problem to find a shortest interconnection of n given points.

Since then, a lot of research has been done in this field. There are numerous variants of the problem to interconnect a set of locations and they model several real-world problems as an algorithmic problem. In the last decades network design problems found applications in the design of integrated chips (VLSI design) and got an important push by it. One of the tasks in VLSI design is to connect components (circuits) placed on a chip in a most efficient way. Each circuit contains pins which enable the connections between the circuits. Several circuits are combined to a so-called net. The pins of the circuits belonging to the same net must be connected by wires. If we take the pins as terminals, a Steiner tree is a solution of such an interlinkage. The exact positions of the wires are established in the routing phase. To work towards a feasible routing, one objective is to minimize the length of the interlinkages of a net. This is modeled by classical

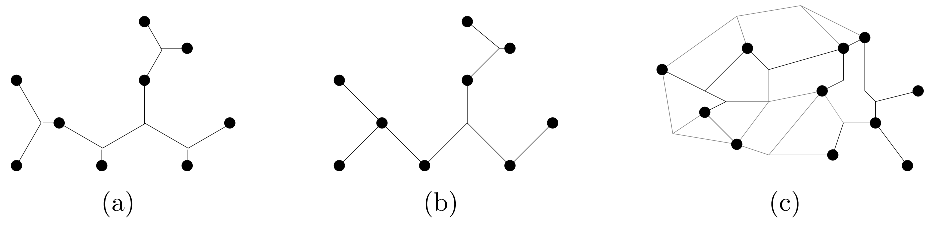

Figure 1.1: (a) A minimum Euclidean Steiner tree. (b) A minimum octilinear Steiner tree on the same terminal set as in (a). (c) A minimum Steiner tree in a network with the black dots being the terminals.

Steiner trees [CW05]. But in VLSI design many constraints have to be considered. One of them are preplaced macros or other circuits which must not be crossed by a wire. The corresponding model is a *Steiner tree with hard obstacles*. A hard obstacle prohibits wiring in the interior and therefore has to be avoided completely in the interior by the Steiner tree.

Due to the availability of several routing layers, most obstacles usually do not block wires completely. However, a large wire requires the insertion of buffers (or inverters) to amplify the signals sent along the wire, in such a way that no induced subtree without any buffer becomes too large. It is impossible to place a buffer or inverter on top of an obstacle. This requirement can be modeled by *soft obstacles*: A Steiner tree is allowed to cross obstacles; however, if we intersect the Steiner tree with some obstacle, no connected component of the induced subtree is allowed to be longer than a given fixed length [HAQG02].

Usually, the nets can be connected to several electrically equivalent pins. In a Steiner tree model these pins are grouped. The related Steiner tree problem is denoted as *group Steiner tree problem*. We search for a Steiner tree which covers at least one point of each group [ZR03].

The wire length affects significantly the power consumption and the time to spread the signal across the chip. Minimum Steiner trees minimize the total wire length. If we want to transmit signals between pairs of components on the chip fast, we search for a network containing a shortest path between each pair of points. This is modeled by *Manhattan networks* which impose this additional constraint.

Usually the wires are allowed to run in horizontal and vertical direction on the chip. A novel routing paradigm in VLSI design is octilinear routing, the so-called *X-architecture* [X], which has recently been introduced. In addition to vertical and horizontal wires, octilinear routing allows wiring in diagonal directions. The wires can be placed on a number of different routing layers. Each layer prefers one of the four allowed directions. To connect adjacent layers so-called *vias*

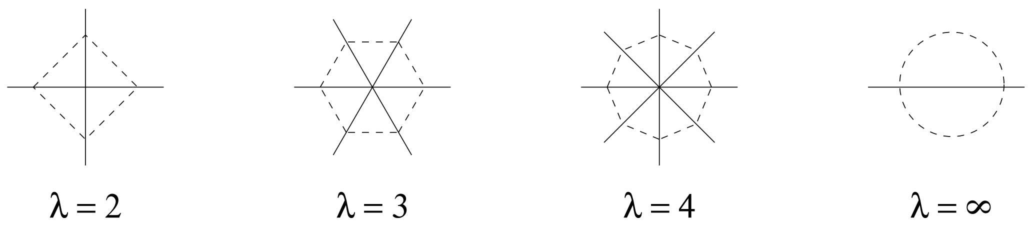

Figure 1.2: Valid orientations and the unit circles of different λ-geometries.

are used. Compared to traditional and state-of-the-art rectilinear (Manhattan) routing, the X-architecture promises clear advantages in wire length but also in via reduction. As a consequence a significant chip performance improvement and power reduction can be obtained (with estimations being in the range of 10% to 20% improvement) [Tei02, CCK$^+$03, PWZ04].

There are basically two different formulations of the Steiner tree problem. The *Steiner tree problem in networks* (also referred to as *Steiner tree problem in graphs*) gets as input a weighted graph and asks for connecting a specified subset of the vertices. The *geometric Steiner tree problem* is based on a set of points in the plane and a given metric. We search for line segments connecting the points with shortest total length with respect to the given metric. We can add new points, so-called Steiner points, to shorten the network. The Euclidean and the rectilinear Steiner tree problems are the most studied geometric problem variants. An instance of the rectilinear Steiner tree problem can be transformed into an instance of the Steiner tree problem in graphs where the constructed graph has quadratic size in the number of terminals. However, as well as the Steiner tree problem in graphs, the Euclidean and the rectilinear Steiner tree problem are NP-hard [Kar72], [GGJ77], [GJ77]. Therefore, besides exact algorithms approximation algorithms have to be considered.

The octilinear Steiner tree problem is another variant of the geometric Steiner tree problem. The line segments connecting the terminals are allowed to run either in horizontal, vertical and diagonal direction. It was shown to be NP-hard [MS05a, Sch05]. Octilinear Steiner trees can be seen in the context of more general routing architectures. They are obtained if a fixed set of uniformly oriented directions is allowed. For an integer parameter $\lambda \geq 2$, consecutive orientations are separated by a fixed angle of π/λ. A λ-*geometry* is a routing environment in which every line segment uses one of the given orientations. See Figure 1.2 for an example of valid orientations for different values of λ. Manhattan routing can then be seen as the special case $\lambda = 2$ and the X-architecture or octilinear routing as the case $\lambda = 4$.

We assume the reader is familiar with basic notions of combinatorial optimization and computational geometry. For a general overview about combinatorial optimization and definitions used in this thesis see the book of Korte and Vygen [KV07]. An introduction in the subject of computational geometry gives the book of de Berg et al. [dBCvKO08]. Comprehensive information about Steiner trees can be found in the book of Hwang et al. [HRW92] and the one of Prömel and Steger [PS02]. For more information about λ-Steiner trees see the surveys of Brazil, Thomas and Weng [BTW00] and the one of Brazil [Bra01].

1.2 Outline

In Chapter 2 we focus on octilinear Steiner trees (although most of our results can be generalized to arbitrary $\lambda \geq 2$). We consider the problem under the additional constraint of octilinear hard and rectangular soft obstacles. As the octilinear Steiner tree problem with hard or soft obstacles contains the ordinary octilinear Steiner tree problem as special case, both problems are also NP-hard [MS05b, Sch05].

We provide two polynomial time approximation schemes (PTAS) for the octilinear Steiner tree problem in the presence of hard octilinear and soft rectangular obstacles. To this end, we construct planar graphs of polynomial size which yield an approximation guarantee of $(1 + \varepsilon)$ of the octilinear Steiner tree problem with hard and soft obstacles, respectively. For rectangular soft obstacles we additionally introduce a $(2 + \varepsilon)$-approximation algorithm by constructing a path preserving path, i. e., a so-called planar spanner which contains for any pair of terminals a path that is at most $(1 + \varepsilon)$ times the length of the shortest path between them. To the best of our knowledge we presented the first polynomial time approximation schemes for octilinear Steiner trees with hard and soft obstacles. Quite recently, Müller-Hannemann and Tazari [MT07] published a PTAS for the λ-Steiner tree problem with soft obstacles that yields a better running time.

In Chapter 3 we consider the Euclidean group Steiner tree problem which is another variant of the geometric Steiner tree problem. We need to pick only one point of each group and search for a shortest interconnection of them. See Figure 1.3 for an example. The Euclidean group Steiner tree problem is a generalization of the ordinary Euclidean Steiner tree problem and therefore belongs also to the class of NP-hard problems [GGJ77]. The groups we consider have to lie inside disjoint regions each fulfilling the property of being α-fat. The term α-fat specifies the shape of the region in comparison to a disk. We will define α-fatness in Section 3.2 and give a $(1 + \epsilon)(9.093\alpha + 1)$-approximation algorithm for the Euclidean group Steiner tree problem where the groups lie inside regions being disjoint α-fat objects. This is the first result given for this special case of Euclidean group Steiner trees.

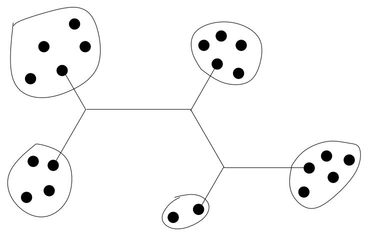

Figure 1.3: A minimum Euclidean group Steiner tree.

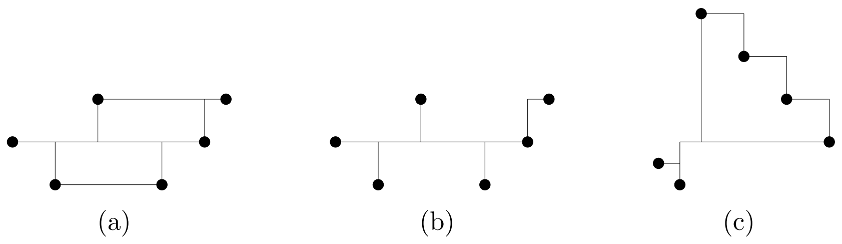

Figure 1.4: (a) A minimum Manhattan network. (b) A minimum rectilinear Steiner tree on the same terminal set as in (b). (c) A staircase.

In the last three chapters we consider so-called Manhattan networks which satisfy an additional constraint. In contrast to Steiner trees they must contain a *shortest* path between each pair of points. As suggested by the name these networks are observed in the Manhattan or λ-geometry for $\lambda = 2$ where one is allowed to use horizontal and vertical line segments only. See Figure 1.4 (a) and (b) for an example of a minimum Manhattan network in comparison to the minimum rectilinear Steiner tree. The problem is a novel field of research. It was introduced by Gudmundsson et al. [GLN01] in 2001. In contrast to the other problems we consider, the complexity status is still open. Our aim is to provide three different approximation algorithms for the Manhattan network problem in Chapter 6. To achieve this we examine the structure of Manhattan networks in Chapter 4. For this we use the notion of a *staircase* and point out how to split a Manhattan network problem into a set of subproblems of Manhattan network problems for staircases. Roughly speaking, a staircase is a set of points or line segments having the shape of a staircase. See Figure 1.4 (c) for an example of a staircase. Staircases were introduced by Gudmundsson et al. [GLN01] but the definition is not standardized. The advantage of staircases is that we can compute minimum Manhattan networks for them in polynomial time. In Chapter 5 we introduce an exact algorithm and a 2-approximation algorithm for Manhattan networks of staircases. Similar approaches are already discussed by Gudmundsson et al. [GLN01] and Benkert et al. [BWWS06]. But since we use a slightly different definition of staircases and we need in Chapter 6 special properties of them, we adopt the algorithms to our situation. With this at hand we present in Chapter 6 a 3-approximation algorithm for the ordinary Manhattan network problem with running time $O(n \log n)$ and two 2-approximation algorithms, the first with running time $O(n^3)$ and the second with running time $O(n \log n)$.

The best approximations published so far are a combinatorial 3-approximation algorithm in time $O(n \log n)$ presented by Benkert et al. [BWWS06], and an LP-based 2-approximation algorithm of Chepoi et al. [CNV08]. Kato et al. [KIA02] proposed a 2-approximation algorithm with running time $O(n^3)$, however the proof of the correctness seems to be incomplete [BWWS06, CNV08]. Seibert and Unger [SU05] presented an approximation algorithm and claimed that it yields a 1.5-approximation. As remarked by Chepoi et al. [CNV08] both the description of the algorithm and the performance guarantee are somewhat incomplete and not fully understandable. In Chapter 6 we show by a counterexample that an important intermediate step of the analysis is incorrect. Thus our 2-approximation algorithm with running time $O(n \log n)$ achieves the best known approximation ratio and running time so far. Last we give an idea how we could possibly find algorithms with better approximation ratios for the Manhattan network problem.

Chapter 2

Octilinear Steiner Trees With Obstacles

In this chapter, we consider the octilinear Steiner tree problem in the plane in the presence of octilinear hard and rectangular soft obstacles.

Definition 2.1. *For a set $P \subseteq \mathbb{R}^2$ of n points in the plane, an* octilinear Steiner tree *on P is a tree that interconnects the points in P such that every line segment runs either in horizontal, vertical or diagonal direction. A* minimum octilinear Steiner tree *is an octilinear Steiner tree of minimum total length.*

The octilinear Steiner tree problem is to find a minimum octilinear Steiner tree. The points of P are denoted as *terminals* and the line segments as *edges*. In the Steiner tree there can be additional points. If their degree exceeds two we refer to them as *Steiner points*. See Figure 2.1 for an example. For a set S of line segments (e. g., a Steiner tree or also a single line segment) we denote by $\ell(S)$ the total length of the line segments in S.

Octilinear Steiner trees model a novel routing paradigm in VLSI design, the so-called *X-architecture* [X] which promises clear advantages in wire length but also in via reduction. In VLSI design preplaced macros or other circuits are obstacles.

Definition 2.2. *An* octilinear obstacle *is a connected region in the plane bounded by a simple polygon such that all segments of the polygon lie either in horizontal, vertical or diagonal direction. If all boundary segments of an obstacle are rectilinear, we call such an obstacle a* rectilinear obstacle*.*

For a given set $\mathcal{O}$ of obstacles we require that the obstacles are disjoint, except for possibly a finite number of common points. By ∂O we denote the boundary of an obstacle O. In practice, obstacles can be assumed to be axis-parallel rectangles. An obstacle which prohibits wiring in the interior and therefore has to be avoided completely in the interior will be referred to as *hard obstacle*.

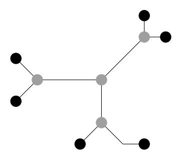

Figure 2.1: An octilinear Steiner tree with terminals (black dots) and Steiner points (grey dots).

Due to the availability of several routing layers, most obstacles usually do not block wires, but it is impossible to place a buffer (or inverter) on top of an obstacle. A large Steiner tree requires the insertion of buffers (or inverters) in such a way that no induced subtree without any buffer becomes too large. This application in VLSI design motivates and translates into our model of *soft obstacles*. In this case the Steiner tree is allowed to run over obstacles; however, if we intersect the Steiner tree with the interior of some obstacle, no connected component of the induced subtree may be longer than a given fixed length L.

The rectilinear and the Euclidean Steiner tree problem have been shown to be NP-hard in [GJ77] and [GGJ77], respectively. The octilinear Steiner tree problem is also NP-hard in the strong sense [MS05b]. The same holds for the octilinear Steiner tree problem with hard or soft obstacles, since it contains the octilinear Steiner tree problem without obstacles as a special case. Arora [Aro98] and Mitchell [Mit99] presented a polynomial time approximation scheme (PTAS) for the Euclidean Steiner tree problem that are applicable to the octilinear Steiner tree problem without obstacles. The running time of Arora's algorithm was improved by Rao and Smith [RS98] from $O(n(\frac{1}{\varepsilon}\log n)^{O(1/\varepsilon)})$ to $O(2^{\text{poly}(1/\varepsilon)}n + n\log n)$ using a so-called "banyan" graph. A *banyan* is a graph containing a $(1+\varepsilon)$-approximation of the Euclidean Steiner tree problem for any subset of the terminals and whose weight is at most a constant larger than the minimum spanning tree of the terminal set. Their graph can be constructed in time $O(n\log n)$ and has size $O(n)$. This is the best known running time so far. None of these algorithms are applicable to the case with obstacles since the so-called "patching lemma" fails to hold.

Most previous work on the octilinear Steiner tree problem considered the problem *without obstacles*. Exact approaches to the octilinear Steiner tree problem have been developed by Nielsen, Winter and Zachariasen [NWZ02] and Coulston [Cou03]. Nielsen et al. [NWZ02] report the exact solution to a large instance with 10000 terminals within two days of computation time. An exact algorithm for obstacle-avoiding Steiner trees in the Euclidean metric has been developed by Zachariasen and Winter [ZW99]. For the octilinear Steiner tree problem *without obstacles* heuristics have been proposed by Kahng et al. [KMZ03] and Zhu et al. [ZZJ+04].

For rectilinear Steiner tree problems, the most successful approaches are based on transformations to the related Steiner tree problem in graphs. Given a connected graph $G = (V, E)$, a length function ℓ, and a set of terminals $P \subseteq V$, a *Steiner tree* of G is a tree which contains all vertices of P and is a subgraph of G. A Steiner tree T is a *minimum Steiner tree* of G if the length of T is minimum among all Steiner trees. It has been shown to be APX-complete to find a minimum Steiner tree [BP89] and thus, no PTAS exists unless $P = NP$. The best available approximation guarantee for the Steiner problem in general graphs is $1 + \frac{\ln 3}{2} \approx 1.55$, obtained by Robins and Zelikovsky [RZ00]. The case of planar graphs has been shown to admit a PTAS by Borradaile et al. [BKMK07a]. The running time of their PTAS is $O(n \log n)$. Its constant has been improved to be singly exponential in $1/\varepsilon$ by Borradaile et al. [BKMK07b]. An implementation by Althaus, Polzin and Daneshmand [APD03] is the currently strongest available exact approach for both the Steiner tree problem in graphs and the rectilinear Steiner tree problem.

Given a finite point set P in the plane, the *Hanan grid* [Han66] is obtained by constructing a vertical and a horizontal line through each point of P. The importance of the Hanan grid lies in the fact that it contains a minimum rectilinear Steiner tree [Han66].

Du and Hwang [DH92] generalized the Hanan grid construction to λ-geometries. They define grids $G_k(P)$ recursively in the following way. For an instance with point set P, $G_0(P) = P$. The grid $G_1(P)$ is constructed by taking λ (infinite) lines with orientations $\pi/\lambda, 2\pi/\lambda, \ldots, (\lambda - 1)\pi/\lambda, \pi$ through each point of P. The k-th grid $G_k(P)$ for $k > 1$ is constructed from the $(k-1)$-th grid by adding for each intersection point x of lines in $G_{k-1}(P)$ additional lines through x with orientations $\pi/\lambda, 2\pi/\lambda, \ldots, (\lambda-1)\pi/\lambda, \pi$. See Figure 2.2 for an example of $G_1(P)$ and $G_2(P)$. Note that for $\lambda = 2$ (the rectilinear case) $G_1(P) = G_2(P) = \ldots$ holds. Lee and Shen [LS96] showed that for every instance of the Steiner tree problem in a λ-geometry with $\lambda \in \mathbb{N}_{\geq 2}$, there is a minimum λ-Steiner tree which is contained in $G_{n-2}(P)$. This result has been strengthened for octilinear Steiner trees by Lin and Xue [LX00]. They showed that a minimum octilinear Steiner tree is already contained in the grid $G_{(\lceil 2n/3 \rceil - 1)}(P)$. Unfortunately, the graph $G_k(P)$ has $\Omega(n^{2^k})$ vertices and edges. Hence, for non-constant k this approach requires an exponentially large graph.

It is therefore an interesting open question which approximation guarantee for the octilinear (or λ–) Steiner tree problem can be achieved if one works with a graph $G_k(P)$ for some fixed constant k. Some partial answers to this question are obvious. Since $G_1(P)$ contains a shortest path between any pair of terminals it also contains the solution obtained from the minimum spanning tree heuristic

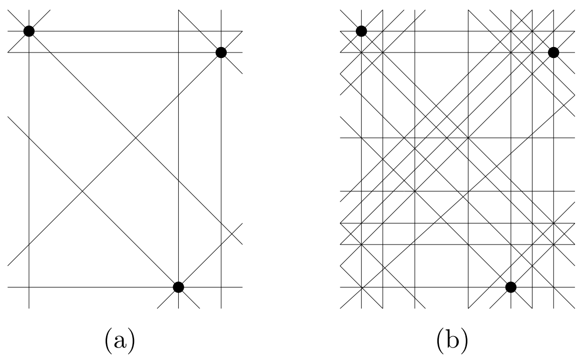

Figure 2.2: (a) The graph $G_1(P)$ for a set of three terminals. (b) The graph $G_2(P)$ for the same terminal set.

of Mehlhorn [Meh88] to approximate the minimum Steiner tree. Therefore, its performance guarantee cannot be worse than the Steiner ratio. The *Steiner ratio* is the smallest upper bound on the ratio between the length of a minimum spanning tree and the length of a minimum Steiner tree. The Steiner ratio in the octilinear case is $\frac{4}{2+\sqrt{2}}$ [Koh95, She97]. This implies that $G_1(P)$ contains a solution which is not more than about 17.15% above the minimum. In Section 2.2 we show how to modify $G_1(P)$ so that we can derive stronger approximation guarantees. For any $k \in \mathbb{N}$ we construct a planar graph of size $O(k^2n^2)$, which contains a $(1 + \frac{1}{k})$-approximation of an octilinear Steiner tree. In Section 2.2.1 we extend this approach to octilinear Steiner trees with octilinear hard obstacles. The basic idea is to refine $G_1(P)$ by superimposing a $k \times k$ grid structure. A similar reduction to the Steiner tree problem in graphs has earlier been used by Provan [Pro88] to approximate the Euclidean Steiner tree problem. Provan also uses a grid structure to define locations of potential Steiner points. However, all these potential Steiner points are pairwise connected (in the presence of obstacles only if they are visible from each other). Thus, this construction requires $O(k^4n^4)$ many edges and so is substantially larger than our construction.

By using the PTAS for the Steiner tree problem in planar graphs given by Borradaile et al. [BKMK07b], we also obtain a PTAS for the octilinear Steiner tree problem with or without hard obstacles.

Afterwards, we consider octilinear Steiner trees with soft rectangular obstacles. Müller-Hannemann and Peyer [MP03] showed that the rectilinear Steiner tree problem in the presence of soft obstacles can be 2-approximated in time $O(n^2 \log n)$, where n denotes the number of terminals plus the number of obstacle vertices. They also introduced a $(1.55+\varepsilon)$-approximation for rectangular soft obstacles. With the later presented algorithm of Borradaile et al. [BKMK07b] their construction leads to a PTAS for the rectilinear Steiner tree problem with rectangular soft obstacles. We generalize these results to the octilinear Steiner tree problem to admit first a $(2+\varepsilon)$-approximation and also a PTAS in the presence of soft rectangular obstacles.

To achieve a $(2+\varepsilon)$-approximation for octilinear Steiner trees with soft obstacles in Section 2.3 our aim is to construct a *path preserving graph*, i. e., a graph which contains a $(1+\varepsilon)$-approximate octilinear path between any pair of terminals. A $(1+\varepsilon)$-approximate path is a path of length at most $(1+\varepsilon)$ times the length of a shortest one. With respect to obstacles, the graph should only contain feasible paths and only feasible Steiner trees. (Note that for soft obstacles the latter does not follow from the feasibility of all paths.) These properties ensure that any approximation algorithm based on this graph for the Steiner tree problem will produce a feasible Steiner tree. In particular, we may use Mehlhorn's [Meh88] minimum spanning tree approximation which runs in time $O(m + n \log n)$ on a graph with n nodes and m edges. This approach yields a $(2+\varepsilon)$-approximation.

Heading for a good running time, our secondary goal is to construct small path preserving graphs. Shortest paths in the presence of polygonal obstacles have already been studied intensively. See the surveys of Mitchell [Mit00] and Lee et al. [LYW96]. In Section 2.3.1 we present a construction of small path preserving graphs that generalizes techniques in previous work of Wu et al. [WWSCW87] and Clarkson et al. [CKV87].

To achieve a $(1+\varepsilon)$-approximation in Section 2.4 we develop a different technique based on t-restricted Steiner trees. A Steiner tree is a *full* Steiner tree if all its terminals are leaves. Any Steiner tree can be decomposed into its full components. A *t-restricted Steiner tree* is a Steiner tree where all full components have at most t terminals. The boundary of each obstacle is discretized by auxiliary vertices with a distance of at most Δ between neighboring vertices. (Δ can be chosen so that we obtain a polynomial number of auxiliary vertices and still achieve the desired accuracy.) Inside obstacles, we approximate an optimal tree with the help of t-restricted Steiner trees for some constant t. We compute them by a linear programming approach. Each of these trees respects the length restriction L for the obstacle. Outside obstacles, a grid-like graph through the terminals and obstacle vertices similar to the one presented in Section 2.2, is refined by additional lines so that it contains a sufficiently close approximation.

Altogether, we construct a planar graph of size $O(n^5)$ containing a $(1+\varepsilon)$-approximation of the octilinear Steiner tree problem with rectangular soft obstacles. Hence, the PTAS for the Steiner tree problem in planar graphs implies also PTAS for this problem. These ideas will be made precise in Section 2.4.

Quite recently, Müller-Hannemann and Tazari [MT07] published a PTAS for the λ-Steiner tree problem with soft obstacles. They construct a planar graph of size $O(\frac{1}{\varepsilon^{11}} n \log^2 n)$ containing a $(1+\varepsilon)$-approximation of the octilinear Steiner tree problem with rectangular soft obstacles.

2.1 Facts for Octilinear Steiner Trees

In this section we recall some basic definitions and known facts about octilinear Steiner trees which will be used in the later analysis of our approach, see for example [LS96].

Property 2.3. *The number of Steiner points for a Steiner tree on n terminals is at most $n - 2$.*

Property 2.4. *The degree of any Steiner point of a Steiner tree is either three or four.*

Property 2.5. *There exists a minimum octilinear Steiner tree such that every degree-4 Steiner point is adjacent to four terminals which form a cross (i. e., all four angles around a degree-4 Steiner point are $\frac{\pi}{2}$).*

Property 2.6. *There exists a minimum octilinear Steiner tree such that the three angles around a degree-3 Steiner point are $\frac{\pi}{2}, \frac{3\pi}{4}, \frac{3\pi}{4}$ (in some order).*

A Steiner tree is a *full* Steiner tree if all its terminals are leaves. Any Steiner tree can be decomposed into its full components.

Property 2.7 ([BTW00])**.** *Given a set $P \subseteq \mathbb{R}^2$ of terminals in the plane such that every minimum Steiner tree is a full Steiner tree. There exists a minimum Steiner tree such that all but at most one edge are straight edges. The latter one may bend once.*

2.2 A PTAS for Hard Obstacles

In this section we show how to improve upon the approximation guarantee obtained for $G_1(P)$ for octilinear Steiner trees. To this end we construct a graph $G_1^k(P)$ which is parameterized by some constant $k \in \mathbb{N}$.

Recall from the introduction that the graph $G_1(P)$ is the graph induced by four lines (vertical, horizontal, and both main diagonals) through each terminal. The idea is to refine $G_1(P)$ by superimposing $O(k)$ additional lines. This is done as follows. Given a set P of terminals with $|P| = n$, let $B(P)$ denote the *bounding box* of this point set, that is, the smallest axis-parallel rectangle which includes all terminals. We subdivide each side of $B(P)$ equidistantly by k points into $k+1$ segments and add for each subdivision point additional lines in all four feasible orientations of the octilinear geometry. See Figure 2.3 for a small example. Since we have $O(n+k)$ lines in each feasible direction, we get $O((n+k)^2)$ intersection points of these lines. Hence, the induced graph $G_1^k(P)$ has $O((n+k)^2)$ vertices and edges. For the bounding box $B(P)$ with side lengths bb_x and bb_y, denote by $bb := \max\{bb_x, bb_y\}$ its maximum side length.

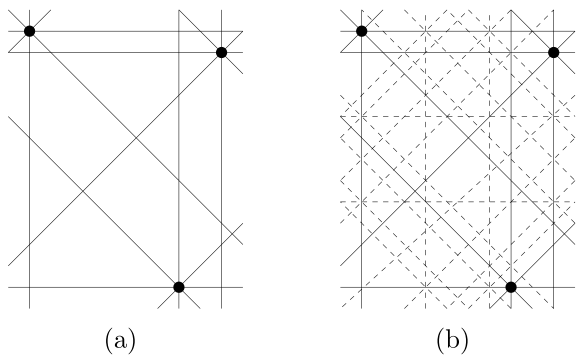
Figure 2.3: (a) Example of the graph $G_1(P)$ for a set of three terminals. (b) The refinement $G_1^k(P)$ with $k = 2$.

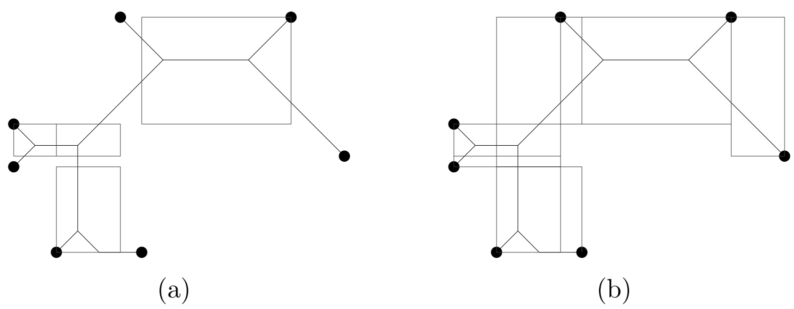
Figure 2.4: (a) Example of the covering of an octilinear Steiner tree by rectangles with edges in $G_1^k(P)$: We first cover the Steiner points. (b) Afterwards we add a smallest enclosing rectangle for each of the six remaining segments which have not yet been covered.

Next we define how to *cover a Steiner tree T by a set $\mathcal{R}$ of axis-parallel rectangles* as follows (the rectangles may overlap). For each Steiner point s of T, the set $\mathcal{R}$ contains a smallest rectangle including s with horizontal and vertical edges from $G_1^k(P)$. In the degenerate case that s lies on a vertex or an edge of $G_1^k(P)$ we add no rectangle. We also add a smallest enclosing rectangle for each point p where an edge of T bends. Degenerate cases are handled as Steiner points. For each straight-line segment of T not covered by previous rectangles we independently add to $\mathcal{R}$ a smallest enclosing rectangle bounded by vertical and horizontal edges from $G_1^k(P)$. Thus, we finally have the following partition of the Steiner tree: $T = \cup_{R \in \mathcal{R}}(T \cap R)$. See Figure 2.4 for an example.

For a given minimum Steiner tree T (fulfilling Properties 2.3–2.7), we construct an approximating Steiner tree T_{app} with edges in $G_1^k(P)$ as follows. For each rectangle $R \in \mathcal{R}$ let S_R be the set of intersection points of T with the boundary of R. We connect the point set S_R in the shortest possible way by (portions of) edges in $G_1^k(P)$, yielding a tree T_R. From the union of all these trees T_R we eliminate in a postprocessing step the longest edge of each cycle which may occur and all leaves and incident edges of the resulting tree which are not terminals. We thereby obtain our approximation T_{app}. The following technical lemma shows that we can bound for each rectangle R included in $\mathcal{R}$ the length $\ell(T_{app} \cap R)$ of $T_{app} \cap R$ in terms of the length $\ell(T \cap R)$ of $T \cap R$. The proof can be found in [MS07, Sch05].

Lemma 2.8. *For each $R \in \mathcal{R}$, the following bound holds:*

$$\ell(T_{app} \cap R) - \ell(T \cap R) \leq (4 - \sqrt{2})\frac{bb}{k+1}.$$

With this lemma at hand we can bound the length of a Steiner tree using only edges of $G_1^k(P)$.

Lemma 2.9. *The graph $G_1^k(P)$ contains an octilinear Steiner tree which is at most a factor of*

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

longer than the minimum one.

Proof. Let P be a set of points in the plane with $|P| = n$ and T_{opt} be some minimum octilinear Steiner tree for P. We have to show that there is some octilinear Steiner tree T_{app} within $G_1^k(P)$ which approximates T_{opt} sufficiently well.

We cover T_{opt} by a set $\mathcal{R}$ of axis-parallel rectangles as described above. Let us assume that T_{opt} is composed of $k' \geq 1$ full Steiner trees with $n_1, n_2, \ldots, n_k' \geq 2$ vertices each. Then $\sum_{i=1}^{k} n_i = n + k' - 1$. Each full component may have at most $s_i \leq n_i - 2$ Steiner points by Proposition 2.3. Hence, the total number of Steiner points satisfies $\sum_{i=1}^{k} s_i \leq n - k' - 1$. If m_i denotes the number of edges in the i-th full component, we have $m_i = n_i + s_i - 1$ for a total of $m = \sum_{i=1}^{k'} m_i \leq 2n - 2 - k'$ edges in T_{opt}.

The cover $\mathcal{R}$ of T_{opt} by rectangles contains at most one rectangle per Steiner point, one rectangle for each edge and at most two additional rectangles per bending edge (one for the bending point and one for the second part of the edge). By Property 2.7, we may assume that each full component has at most one bending edge. Thus

$$|\mathcal{R}| \leq \sum_{i=1}^{k'} s_i + \sum_{i=1}^{k'} m_i + 2k' \leq 3n - 3.$$

Next, we analyze the length $\ell(T_{app})$ of T_{app} in comparison to the optimal length $\ell(T_{opt})$. All edges of T_{opt} which are incident to a terminal are represented in $G_1^k(P)$.

Hence, for all corresponding rectangles $\ell(T_{opt} \cap \mathcal{R}) = \ell(T_{app} \cap \mathcal{R})$. Clearly, there are at least n edges incident to terminals. This implies that for at most $2n-3$ rectangles of $\mathcal{R}$ there will be a difference between $\ell(T_{app} \cap \mathcal{R})$ and $\ell(T_{opt} \cap \mathcal{R})$. Thus, we have

$$\begin{aligned}\frac{\ell(T_{app})}{\ell(T_{opt})} &= \frac{\ell(T_{opt}) + \sum_{R\in\mathcal{R}}(\ell(T_{app} \cap \mathcal{R}) - \ell(T_{opt} \cap \mathcal{R}))}{\ell(T_{opt})} \\ &\leq \frac{\ell(T_{opt}) + (2n-3)\cdot \max_{R\in\mathcal{R}}\{\ell(T_{app} \cap \mathcal{R}) - \ell(T_{opt} \cap \mathcal{R})\}}{\ell(T_{opt})}.\end{aligned}$$

Hence, it suffices to show that

$$\max_{R\in\mathcal{R}}\{\ell(T_{app} \cap \mathcal{R}) - \ell(T_{opt} \cap \mathcal{R})\} \leq (4-\sqrt{2})\cdot\frac{\ell(T_{opt})}{k+1}.$$

This relation follows from Lemma 2.8 and the observation that $\ell(T_{opt}) \geq bb$, since every Steiner tree must connect the terminals which define the bounding box $B(P)$. □

Theorem 2.10. *For a given set $P \subseteq \mathbb{R}^2$ of n terminals, and for every $\varepsilon > 0$ there exists a graph of size $O(\frac{n^2}{\varepsilon^2})$ which contains a $(1+\varepsilon)$-approximation of a minimum octilinear Steiner tree.*

Proof. The approximation guarantee follows directly from Lemma 2.9 if we choose $k := \frac{(4-\sqrt{2})2n}{\varepsilon}$. With such a choice of k, the graph has the claimed size. □

Trivially, we get the following corollary

Corollary 2.11. *Let α denote the approximation guarantee for an algorithm solving the Steiner tree problem in graphs. For a set $P \subseteq \mathbb{R}^2$ of terminals, and some $\varepsilon > 0$, there is an $(\alpha+\varepsilon)$-approximation of the octilinear Steiner tree problem.*

2.2.1 Extension to Obstacles.

Let us now work out the necessary modifications in the presence of obstacles. Let P be a set of points (terminals) in the plane and $\mathcal{O}$ be a set of octilinear (or rectilinear) obstacles. Denote by $V_{\mathcal{O}}$ the set of obstacle vertices. Let $n = |P| + |V_{\mathcal{O}}|$.

Analogously to the definition of $G_1(P)$, we now define a graph $G(P,\mathcal{O})$ which is induced by the set $\mathcal{L}$ of lines in all feasible directions in 4-geometry going through terminals or obstacle vertices.

For a given parameter k, we refine $G(P,\mathcal{O})$ by adding lines. For any two parallel lines in $\mathcal{L}$ which are neighbored (i.e., no third line with the same orientation lies between them) we add k additional lines with the same orientation between them and place them equidistantly. In total, we get $O(nk)$ lines.

From the resulting induced graph, we erase all vertices and their incident edges which lie strictly inside some obstacle. The latter guarantees that every Steiner tree in this graph corresponds to a tree in the plane which avoids all obstacles.

Theorem 2.12. *For a set $P \subseteq \mathbb{R}^2$ of terminals, a set $\mathcal{O}$ of octilinear hard obstacles with n terminals and obstacle vertices, and for every $\varepsilon > 0$ there is a graph of size $O(\frac{n^2}{\varepsilon^2})$ which contains a $(1+\varepsilon)$-approximation of a minimum octilinear Steiner tree with obstacles which have to be avoided.*

The proof of this theorem follows basically the same ideas as that for the case without obstacles. There is one essential difference, however. In the presence of obstacles, edges between terminals and/or Steiner points may be forced to bend several times. But if such an edge bends, then all but at most two of its straight segments will lie on $G(P, \mathcal{O})$. Since we have at most $2n-3$ edges (each contributing two segments and possibly one corner point), but n edges incident to terminals and $n-2$ Steiner points, a cover by rectangles of a minimum octilinear Steiner tree requires at most $3 \cdot (2n-3) - n + n - 2 = 6n - 11$ rectangles on which we have to find an approximative solution. This upper bound of $6n-11$ rectangles (instead of $2n-3$ without obstacles) suffices for an analogous result as in Lemma 2.9. Again we get the following corollary.

Corollary 2.13. *Let α denote the approximation guarantee for an algorithm solving the Steiner tree problem in graphs. For a set $P \subseteq \mathbb{R}^2$ of terminals, a set $\mathcal{O}$ of octilinear hard obstacles, and some $\varepsilon > 0$, there is an $(\alpha+\varepsilon)$-approximation of the octilinear Steiner tree problem with obstacles which have to be avoided.*

With the PTAS introduced by Borradaile et al. [BKMK07b] we get the following corollary:

Corollary 2.14. *Given set $P \subseteq \mathbb{R}^2$ of terminals, a set $\mathcal{O}$ of octilinear hard obstacles, and some $\varepsilon > 0$, there is a PTAS of the octilinear Steiner tree problem with obstacles which have to be avoided.*

2.3 A $(2+\varepsilon)$-Approximation for Soft Obstacles

For a set $\mathcal{O}$ of soft obstacles we introduce length restrictions for those portions of a tree T which cross obstacles. Namely, for a given parameter $L \in \mathbb{R}_0^+$ we require the following for each obstacle $O \in \mathcal{O}$ and for each strictly interior connected component T_O of $(T \cap O) \setminus \partial O$: the length $\ell(T_O)$ of such a component must not be longer than the given length restriction L. Note that the intersection of a minimum Steiner tree with an obstacle may consist of more than one connected component and that our length restriction applies individually for each connected component.

For ease of exposition, we restrict our presentation of soft obstacles to (axis-parallel) rectangular obstacles. Generalizations to rectilinear and octilinear soft obstacles are possible and do not change the asymptotic size of the resulting graphs.

2.3.1 Octilinear Shortest Paths Amidst Hard Obstacles

Before we actually consider soft obstacles we insert another construction for hard obstacles which we need afterwards. Thus, let P be a set of points (terminals) in the plane and $\mathcal{O}$ be a set of octilinear hard obstacles. Denote by $V_{\mathcal{O}}$ the set of obstacle vertices. Let $n = |P| + |V_{\mathcal{O}}|$. In this paragraph we will show how to construct shortest path preserving graphs.

As a first step we construct a path preserving graph based on visibility. Our construction may be viewed as a generalization of that of Wu et al. [WWSCW87], which was designed for rectilinear polygons and rectilinear paths. To simplify our discussion we add to our scene a bounding box containing all obstacles and all terminals. Clearly all desired paths will lie within this bounding box.

A *track tr* generated by a point t and an orientation is a line segment that starts at t and ends when it first hits an obstacle edge or the bounding box. The generated endpoints of tracks are called *track-induced Steiner points*. For each terminal t and each feasible orientation we construct a track in both directions from t. Similarly, we introduce tracks for each convex obstacle vertex v. More precisely, if $e_1 = (v_1, v)$ and $e_2 = (v, v_2)$ are polygon edges incident with v in clockwise order of the polygon, denote by r_1 the ray in direction from v_1 to v, and by r_2 the ray in direction from v_2 to v. We construct a track generated by v for all feasible directions which do neither go through the interior of the obstacle nor through the interior of the sector spanned by ray r_1 and r_2 in counter-clockwise order. See Figure 2.5 (a).

The intersections among all tracks and their endpoints are made the vertices of the track graph. The edges are the track segments between the intersections. The construction is completed by adding edges connecting two consecutive track-induced Steiner points or polygon vertices along the boundary of each obstacle. The length of an edge in the track graph is simply the octilinear distance between its endpoints. See Figure 2.5 (b) for a small example which illustrates this construction. The track graph consists of $O(n)$ many tracks which induce $O(n^2)$ many vertices and edges.

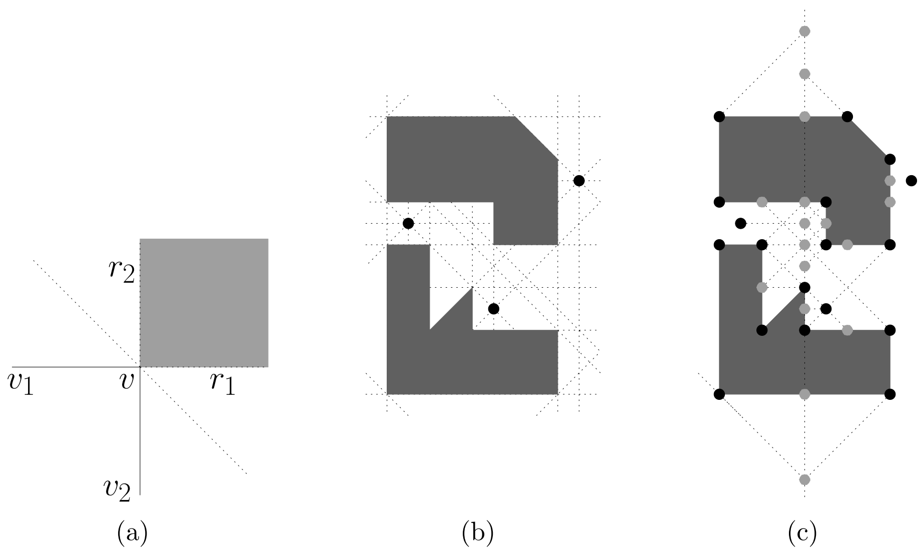

Figure 2.5: Illustration of the graph construction. (a) The tracks around a convex vertex v of some obstacle. There is no track inside the shaded area. (b) The track graph for an instance with three terminals (black dots) and two hard octilinear obstacles. (c) The first vertical cut line construction.

For rectilinear paths it is possible to restrict the track construction to so-called extreme edges [WWSCW87]. An obstacle edge is *extreme* if its two adjacent edges lie on the same side of the line containing the edge. Note that such a reduction is not possible for octilinear paths.

Sparser path-preserving graphs. To improve upon the quadratic space bound of the track graph we use an idea of Clarkson et al. [CKV87] and adapt their approach to the octilinear case. We construct a sparser path-preserving graph $G = (V, E)$ as follows. The vertex set is constructed in two rounds. In the first round, we create V_1 as the union of

1. the set of all terminals P,
2. the set of all obstacle vertices $V_{\mathcal{O}}$, and
3. the set of track-induced Steiner points for tracks induced by P and $V_{\mathcal{O}}$.

With respect to V_1 we create the set V_2 recursively by adding more Steiner points along vertical, horizontal and diagonal so-called *cut lines*. We explain the construction for vertical cut lines. A vertical cut line is placed at the x-coordinate of that point such that $\lfloor V_1/2 \rfloor$ points of V_1 have smaller x-coordinate than this point. Vertices in V_1 generate projection points on the line. Projections are performed in all feasible orientations so that we may get up to three projection points on the line for each vertex in V_1 (in the rectilinear setting, Clarkson et al. need to project only orthogonally onto the cut line). Two points are mutually *visible* to each other if the straight line segment between them contains no obstacle point in its interior. All those projection points on a cut line which are visible from some inducing point in V_1 are put into the vertex set V_2. Moreover, we add the intersection points of the cut line with obstacle vertices to V_2. The following edges are inserted into E. Two consecutive Steiner points on the cut line are connected by an edge if these points are visible to each other. We also add edges from each vertex in V_1 to its corresponding projection points.

This procedure is repeated recursively with the vertices respectively on the left and right sides of the cut line. The union of all these vertices yields $V = V_1 \cup V_2$. See Figure 2.5 for a vertical cut line on the highest level. There are $O(\log n)$ many levels of recursion, and in each level we will create $O(n)$ many vertices and edges. This gives in total $O(n \log n)$ vertices and edges. Finally, for each obstacle we have edges between consecutive vertices from V on its boundary.

The proof of the following theorem can be found in [MS05a, Sch05].

Theorem 2.15. *For any two vertices from P the constructed graph G contains a shortest octilinear path. The graph G has $O(n \log n)$ vertices and edges.*

2.3.2 Track Graph Construction

For soft obstacles, an analogous construction of the track graph is substantially more complicated than for hard obstacles. (This is in sharp contrast to the rectilinear case). We obtain the track graph by applying the following rules inductively. See Figure 2.6 for an illustration of each rule.

1. We generate track lines for all terminals and all feasible orientations. But in contrast to hard obstacles, a track does not end as soon as it hits an obstacle. It only ends at an obstacle if the intersection of the track line with the obstacle exceeds the given length restriction L. Hence, we distinguish between Steiner points which are endpoints of a track due to a length restriction, called *L-Steiner points*, and all other Steiner points generated as intersections of a track line and obstacles. The latter type of Steiner points will still be called *track-induced Steiner points.*

2. Similarly, we introduce track lines through all vertices of rectangular polygons. This yields $O(n)$ track lines and may cause $O(n^2)$ many track-induced Steiner points.

3. Additional tracks are needed to make shortcuts when an obstacle causes a deviation due to the length restriction L.

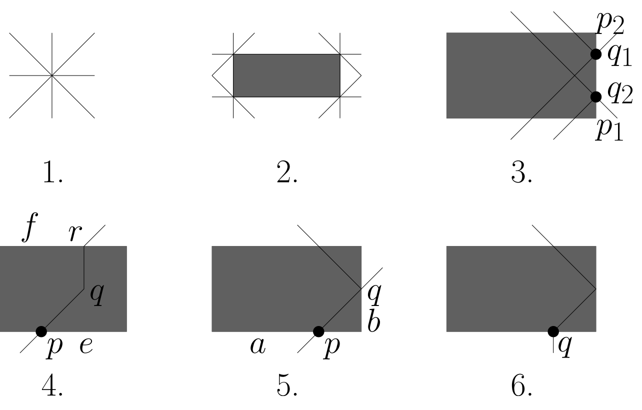

Figure 2.6: The different types of tracks for soft obstacles.

For each edge $e = (p_1, p_2)$ of an obstacle with length $\ell(e) > L/\sqrt{2}$ we do the following. At the points on e with distance $L/\sqrt{2}$ from the corners p_1 and p_2 we respectively generate tracks which have an angle of 45° and 135° with e and run through the obstacle but do not exceed the length restriction. For the track which has length L inside the obstacle we add also a track in the other direction (rotated by 180°) outside the obstacle. This yields another $O(n)$ track lines and $O(n^2)$ track-induced Steiner points.

4. Next suppose that a track tr ends at a point p of an edge e of some obstacle due to the length restriction and hits e with an angle of 45°. If the edge f which is opposite to e in such a rectangle has a distance not exceeding L from e, we let the track continue inside the rectangle up to a certain point q. At q the track bends by an angle of 135° and continues until it hits edge f, say at r. The point q is chosen in such a way that length of the two segments $\overline{pq}$ and $\overline{qr}$ together equals the length restriction L. Finally, at r a new track parallel to tr is created. Note that tracks generated for this item do not increase the asymptotic complexity.

5. Now consider the following situation. A track tr enters an obstacle O at some point p on edge a in an angle of 45° and leaves the obstacle at some point q on an edge b of O which is adjacent to a. Furthermore, we assume that the length of b exceeds L. Then we start a new track tr_2 at q which runs orthogonally to tr through the obstacle, provided that $\ell(tr_2 \cap O) < L$ (i.e., the intersection of tr_2 with O does not exceed L; if equality holds this track has already been inserted). As a track may cross many obstacles each of which potentially induces a new track of the just described kind, and newly generated tracks in turn may induce further tracks of this kind, we have to be careful not to generate infinitely many new tracks. Therefore, the generation process is done in rounds for each track generated in Items 1-4. In each round, we create a tree of new tracks, called *track tree*. The

root r of such a track tree is one of the tracks generated by Items 1-4. Every induced new track is made an immediate successor of its inducing track. A round ends if no new track is induced. To make each round finite, we add the following rule. Consider a fixed round and suppose that we have generated in step i of this round a track tr_i from a Steiner point on rectangle side e. If in a later step $j > i$ we would have to insert a further track tr_j from the very same rectangle side e due to Item 5 and this track would have track tr_i as a predecessor in the track tree, such a track is not necessary. This is because in such a scenario the generated tracks would form a full cycle around a rectangle, and clearly no cycle can be in a shortest path. Hence, our rule is not to generate a further track in such cases. By applying this rule, we have a finite number of tracks.

6. Suppose that a track tr ends at an obstacle O due to the length restriction and hits edge e of O orthogonally at some point q. Moreover, suppose q has a distance of less than $L/\sqrt{2}$ from some obstacle corner v on e. Then we add a segment and a new track to shortcut the way around O (the latter only if its intersection with O does not exceed L). See again Figure 2.6. We handle such tracks as in the previous item.

This completes the construction of our track graph. For the proof of the following lemma see [MS05a, Sch05]:

Lemma 2.16. *The constructed track graph contains a shortest length-restricted path between each pair of terminals.*

2.3.3 Approximate Shortest Paths

The track graph as described in Section 2.3.2 above may have exponential size. With a smarter construction one can bound the size of the track graph by $O(n^3)$ (but the proof then becomes quite complicated) [Sch05]. We therefore prefer a simpler construction which uses approximate shortest paths. For any integer k, we obtain $(1+\frac{1}{k})$-approximate shortest paths. The idea is to leave out Items 5 and 6 of the track graph construction (which are responsible for the blow up in the graph size). Instead, we insert $k-1$ additional tracks for each corner of an obstacle. These tracks "cut off" the corner and are placed in distance $\frac{j \cdot L}{\sqrt{2} \cdot k}$ from the corner for $j = 1, \ldots, k-1$. See Figure 2.7.

This construction induces $O(kn)$ many new tracks which are responsible for $O(kn)$ new track-induced Steiner points per obstacle. Next we apply the same sparsification technique as we explain in Section 2.3.1 for hard obstacles and make sure that every path in our graph is feasible with respect to our length restriction.

We do this in two steps. In a first step, we regard all obstacles as hard obstacles and use the modified cut line approach on the set of original vertices,

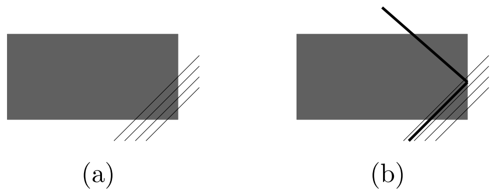

Figure 2.7: (a) The extra tracks inserted at a corner of some obstacle to approximate shortest paths. (b) Clearly, the "thick" path is only slightly longer than an approximation using one of the extra tracks.

terminals and all track-induced Steiner points. The overall number of Steiner points is $O(kn^2)$. Hence, the sparsification technique outside obstacles yields $O(kn^2 \log(kn))$ many vertices and edges.

In a second step, we add connections between vertices and Steiner points on the boundary of obstacles. In the previous discussion we observed that we may have $O(kn)$ many track-induced Steiner points lying on the boundary of an obstacle O. Locally these Steiner points can be regarded as terminals which have to be connected pairwise without violating the length bound L.

Lemma 2.17. *Let O be a rectilinear obstacle with t terminals on its boundary. Then we need $O(t^2)$ many edges for a graph which has (1) to represent shortest paths between any pair of terminals respecting the length restriction L, and (2) does not contain any path exceeding the length restriction L inside some obstacle.*

Proof. For every pair of terminals, we add an edge if and only if their octilinear distance is less or equal to L. □

Thus, we can now apply Lemma 2.17 with $t = O(kn)$ and get $O(k^2n^2)$ edges inside a single obstacle, for a total of $O(k^2n^3)$ edges inside all obstacles.
It is easy to see that shortest paths between terminals in this modified graph will be at most a factor of $(1 + \frac{1}{k})$ longer than shortest paths.

Lemma 2.18. *There is a graph for soft rectangular obstacles with $O(kn^2 \log(kn))$ many vertices and $O(k^2n^3)$ many edges which contains a $(1 + \frac{1}{k})$-approximative shortest path between any pair of terminals for any integer k. Moreover, all paths in this graph respect the length restriction L inside obstacles. The graph can be constructed in time proportional to its size.*

Proof. The size of the graph follows directly from our explanations given before. It is also clear that it can be constructed in the same time.

Thus it remains to prove the approximation guarantee. Denote the track graph according to Item 1-6 by G_1. By Lemma 2.16, the constructed track graph contains a shortest length-restricted path between each pair of terminals. Next,

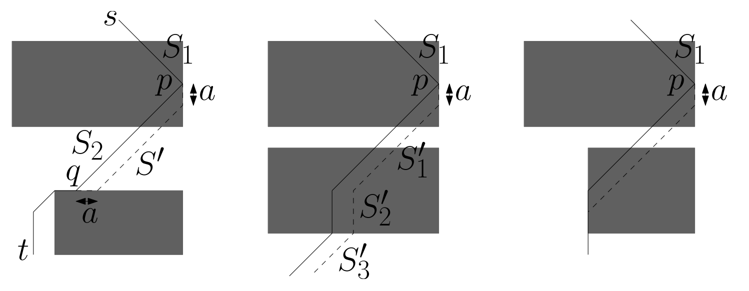

Figure 2.8: Approximation of a path in the track graph G_1. Parts of the original solid path not in G_1 are replaced by the dashed path which belongs to G_2.

we analyze the effect of inserting $k-1$ additional tracks for each corner instead of applying Item 5 and 6 of the track graph construction. Denote the track graph after this modification by G_2. For two arbitrary terminals s and t, consider a shortest length-restricted path $\mathcal{P}$ in G_1 from s to t. We may assume that this path $\mathcal{P}$ has the fewest number of bends among all shortest paths between s and t. We embed this path in G_2 as follows. All segments of $\mathcal{P}$ which are also in G_2 remain unchanged. All remaining segments are embedded successively. As long as the segments in G_2 are not connected, denote by S_1 the last inclusion-maximal segment of $\mathcal{P}$ (in the given orientation from s to t) which is also represented in G_2, and by S_2 the first inclusion-maximal segment not represented in G_2. Then, the common point of S_1 and S_2 must be a boundary point p of some obstacle O, and S_2 must lie on a track line introduced by Item 5 or 6. Let $\mathcal{P}_2$ denote the subpath of $\mathcal{P}$ which starts at p with segment S_2 and ends at a point q which is chosen as follows. The point q is the first point on $\mathcal{P}$ when traversing the path from p towards t where $\mathcal{P}$ enters an edge which also belongs to G_2 or where it bends once more and enters another track inserted by Item 5 or 6 in the track graph construction. (Clearly, such a q exists since the last bending point before arriving at t is always a candidate.) This subpath $\mathcal{P}_2$ is replaced by a sequence of track lines $S'_1, S'_2, \ldots, S'_t$ in G_2 which are parallel to and at most a distance of $a < \frac{L}{\sqrt{2}\cdot k}$ away from corresponding segments of $\mathcal{P}_2$. These segments are linked to the rest of $\mathcal{P}$ by at most two short segments of length a on the boundary of obstacles as shown in Figure 2.8. Note that the replacement exists, i.e., none of the necessary track lines $S'_1, \ldots, S'_t$ is stopped because of a length violation inside some obstacle. This is true since the intersection of some S'_i, $1 \le i \le t$, with some obstacle cannot exceed the length restriction L as otherwise either the corresponding segment of $\mathcal{P}$ would already have been infeasible or some other track line would be nearer to $\mathcal{P}$ contradicting the choice of our replacement (the right case in Figure 2.8).

Suppose, we have to apply such a modification m times. In each case, the original path goes through an obstacle O and the intersection of the path with the obstacle must be at least

$$\ell(\mathcal{P} \cap O) \geq \sqrt{2}L,$$

as otherwise no track for Item 5 or 6 would have been inserted. Hence, the length of $\mathcal{P}$ is lower bounded by $\sqrt{2}Lm$. The length of a is certainly smaller than the distance between two inserted additional track lines, hence $a \leq \frac{L}{\sqrt{2}\cdot k}$ and $\ell(S') \leq \ell(S_2)$ for each modified segment S'. Thus the modified path $\mathcal{P}'$ satisfies

$$\begin{aligned}\ell(\mathcal{P}') &\leq \ell(\mathcal{P}) + \frac{2Lm}{\sqrt{2}\cdot k}\\ &\leq \ell(\mathcal{P}) + \frac{\ell(\mathcal{P})}{k}\\ &\leq (1+\frac{1}{k})\cdot \ell(\mathcal{P}).\end{aligned}$$

We finally note that the sparsification technique applied to G_2 does not further change path lengths. In the same way as we proved the correctness for the sparsification technique for hard obstacles, one can show that the distance between any two track-induced Steiner points remains unchanged by the sparsification.

The final graph contains only paths which respect the length restriction inside obstacles since we deleted for each obstacle O the whole subgraph of G_2 with edges and vertices "inside" O, and replaced these subgraphs by length-feasible direct connections between points on the boundary. □

As the obtained graph contains only length-feasible paths, we can apply Mehlhorn's [Meh88] implementation of the minimum spanning tree heuristic to construct a Steiner tree. We finally obtain:

Theorem 2.19. *For any fixed $\varepsilon = 1/k$, we can find a $(2+\varepsilon)$-approximation of the octilinear Steiner tree problem with soft rectangular obstacles in time $O(\frac{1}{\varepsilon^2}n^3)$.*

We conclude by mentioning that our analysis is tight. It is possible to construct a class of instances for which our approximation algorithm asymptotically achieves a performance guarantee of 2 [MS05a, Sch05].

2.4 A PTAS for Soft Obstacles

We construct a graph of polynomial size which contains a $(1+\varepsilon)$-approximation for the octilinear Steiner tree problem with soft rectangular obstacles. Next we give a detailed description of this construction.

2.4.1 Graph Construction

The graph construction requires the following five steps:

Step 1: The very first step is to compute an axis-parallel square which contains an optimal Steiner tree. Everything outside such a square can then be safely ignored in the subsequent steps. For the analysis it is important that the side length b of this square can be bounded by a constant times the length of a minimum Steiner tree T_{opt}.

To achieve this goal, we can run the minimum spanning tree based approximation of Mehlhorn [Meh88]. Let us assume that this approximation yields a tree of length $\ell(T_{MST})$. Denote by $B(P)$ the bounding box of the given terminal set, that is, the smallest axis-parallel rectangle which includes all terminals. Let bb be the maximal side length of $B(P)$. Now we can define $b := bb + 2\ell(T_{MST})$. Clearly, an axis-parallel square B of side length b centered at the barycenter of $B(P)$ is large enough to contain a minimum Steiner tree. Since the minimum spanning tree yields a 2-approximation and $bb \leq \ell(T_{opt})$, we also have

$$b \leq 5 \cdot \ell(T_{opt}). \tag{2.1}$$

Note the difference to hard obstacles. For hard obstacles it suffices to consider the bounding box of all terminals because outside this box possible edges of a minimum Steiner tree run along the boundary edges of the obstacles (being in the constructed graph anyway).

Step 2: As in Section 2.2 we build a refinement of a Hanan-like grid graph restricted to the area of B. Again this refinement is parameterized by some parameter k (to be determined later). More specifically, we subdivide the boundary of the square B equidistantly with k points into $k + 1$ segments and add for each subdivision point additional lines in all four feasible orientations of the octilinear geometry.
To this set of lines we add lines through each terminal and each vertex of an obstacle in all feasible directions. Let G be the graph induced by intersections of these lines restricted to the area inside B (including the boundary of B).

Step 3: The resulting graph may allow subtrees inside obstacles which violate the length restriction L. Therefore, we delete all nodes and edges which lie strictly inside some obstacle.

Step 4: Let $t \in \mathbb{N}$ be another parameter which will be chosen as a constant depending on ε but independent from the given instance. For each obstacle O and for each subset S of at most t vertices on the boundary of O compute an optimal Steiner tree for S which respects the length restriction L inside O. We add each such Steiner tree to the current graph and identify common boundary

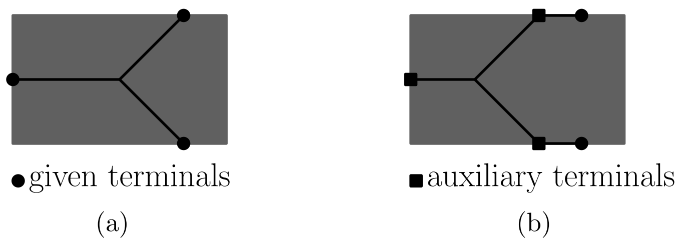

Figure 2.9: (a) The unrestricted optimal Steiner tree is shown. (b) The optimal solution subject to a length restriction.

vertices. Since t is a constant, there is only a polynomial number of these small Steiner tree instances and each of these trees can be computed in constant time as we will show later.

Step 5: Finally, we want that our graph contains a feasible almost shortest octilinear path between any pair of vertices on the boundary of obstacles. More precisely, we require that these paths approximate the true shortest paths by a factor of $1 + 1/(k+1)$. We can compute these paths and their lengths by the methods from Section 2.3 and add them to the graph.

On the resulting graph $G = G(k,t)$, parameterized by k and t, we can then solve the Steiner tree problem for the given terminal set P.

The parameter t will be chosen as a constant and the parameter $k = O(n)$. This immediately implies that the constructed graph has size $O(n^5)$. It remains to show that Step 4 can be done efficiently. Therefore, we have to show the following lemma.

Lemma 2.20. *Let S be a set of at most t terminals on the boundary of some rectangle O and L be some length restriction inside O. If t is a constant, then the octilinear Steiner tree problem for S with length restriction L can be solved in constant time.*

No that the *topology* of a Steiner tree merely refers to the graph structure, i.e., it includes the terminals and Steiner points as vertices and specifies the connections between these vertices as edges. However, the topology does not include the geometric embedding in the plane.

Proof of Lemma 2.20. For a given set S of terminals we first compute an optimal octilinear Steiner tree *without* considering the length restriction L. To this end, we simply enumerate over all possible tree topologies and finally take the shortest tree. Since every tree can be decomposed into its full components, we restrict our attention only to full trees. By Properties 2.3 to 2.6 the possible tree topologies are restricted. Their number is obviously finite. Brazil et al. [BTWZ02] have shown that, for each given topology one can construct a Steiner minimum tree (that means find an optimal embedding) in linear time in t.

If the optimum tree for S also satisfies the length restriction L, we are done. However, if this tree exceeds the length restriction and is therefore infeasible, we need some more work.

In such a case, the optimal feasible tree is composed by one or more full trees of *exactly length* L inside the obstacle O and some segments on its boundary which connect the full tree with the given terminals. See Figure 2.9 for a small example. The precise position of the full tree can be computed by linear programming as we will point out in the following. Again we restrict our attention to the case of a single full component inside O.

An octilinear (L, s)*-tree in a rectangular obstacle* O is a full Steiner tree of length L with s terminals which are located on the obstacle's boundary.

Assume that $S = \{t^{(1)}, t^{(2)}, \ldots, t^{(s)}\}$ are the given terminals, for $s \le t$. For each given terminal $t^{(i)}$, $1 \le i \le s$, we associate an auxiliary terminal $t'^{(i)}$ (this mapping is, in general, not injective. Two given terminals may be mapped to the same auxiliary terminal). These auxiliary terminals shall be the terminals of an (L, s)-tree. The coordinates of these auxiliary terminals have to be determined so as to minimize the segment lengths on the boundary. Let us fix the tree topology including the orientation of its edges of an (L, s)-tree. We also fix the counterclockwise order of given terminals and auxiliary terminals around the boundary of O and their assignment to the four rectangle sides of the obstacle O. Our objective is to minimize the overall length of the segments connecting the auxiliary terminals with the given terminals. This is a linear function in the unknown coordinates $t'^{(i)}_x, t'^{(i)}_y$ (for $1 \le i \le s$) subject to several linear side constraints. Assume that the origin of our coordinate system is the left bottom corner of the rectangle O. We require that

- for a rectangular obstacle of dimension $a \times b$, all vertical coordinates are in the range $[0, a]$ and all horizontal coordinates are in the range $[0, b]$.
- The length of the full Steiner tree is exactly L. The length of an (L, s)-tree T can be expressed as a function of the coordinates of its terminals and Steiner points. We obtain the linear equality

$$
\begin{aligned}
L &\equiv \sum_{e \in E(T)} \ell(e) \\
&= \sum_{\substack{e \in E(T) \\ \text{vertical}}} \ell(e) + \sum_{\substack{e \in E(T) \\ \text{horizontal}}} \ell(e) + \sum_{\substack{e \in E(T) \\ \text{diagonal}}} \ell(e) \\
&= \sum_{\substack{e=\{u,v\} \in E(T) \\ \text{vertical}}} |u_y - v_y| + \sum_{\substack{e=\{u,v\} \in E(T) \\ \text{horizontal}}} |u_x - v_x| \\
&\quad + \sqrt{2} \cdot \sum_{\substack{e=\{u,v\} \in E(T) \\ \text{diagonal}}} |u_y - v_y|.
\end{aligned}
$$

- All tree edges have nonnegative length. This gives one linear inequality for the coordinates of each edge.
- The distance between pairs of auxiliary terminals on opposite sides of the tree must be exactly the corresponding side length of the rectangle. For each such pair we obtain a linear equality.
- The given ordering of auxiliary terminals and terminals is not violated. This gives one or two additional linear inequalities per auxiliary terminal.

Thus, finding an optimal embedding of an (L, s)-tree for a fixed topology amounts to solving a linear programming problem of constant dimension. As this can be solved in constant time, our lemma follows by enumerating over all possible tree topologies. $\square$

2.4.2 Analysis of the Approximation

For the analysis, we fix some minimum Steiner tree T_{opt}. To bound the approximation achieved by our graph G, we partition T_{opt} into several parts which are analyzed independently. To this end we define how to *cover a Steiner tree T_{opt} by a set of axis-parallel rectangles.* This set $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ is obtained as follows. Denote by $\mathcal{R}_1$ the set of obstacles which include at least one Steiner point of T_{opt} in its interior. For each Steiner point s of T not covered by an obstacle, the set $\mathcal{R}_2$ contains a smallest rectangle including s with horizontal and vertical edges from G. In the degenerate case that s lies on a vertex or an edge of G we add no rectangle. We also add a smallest enclosing rectangle for each point p where an edge of T bends. Degenerate cases are handled as with Steiner points. For each straight-line segment of T not covered by previous rectangles we independently add to $\mathcal{R}_2$ a smallest enclosing rectangle bounded by vertical and horizontal edges from G. Thus, we finally have the following partition of the Steiner tree: $T_{opt} = \cup_{R \in \mathcal{R}} (T_{opt} \cap R)$.

The constructed graph G contains an approximative tree T_{app} which we obtain as follows. The general idea is to replace portions of the optimal tree by trees contained in G. From the union of all these trees we eliminate in a postprocessing step the longest edge of each cycle which may occur and all leaves and incident edges of the resulting tree which are not terminals.

Replacing portions covered by $\mathcal{R}_1$. Let T_R be some inclusion-maximal connected component of T_{opt} which lies strictly inside $R \in \mathcal{R}_1$ except for a finite set of points on the boundary of R. Denote by P_R this set of boundary points of T_R. In our graph G, the boundary of each obstacle within the square B has been discretized. For any point p on the boundary there is a point in the discretized set with distance at most

$$\Delta \leq \frac{b}{k+1} \leq \frac{5 \cdot \ell(T_{opt})}{k+1} \tag{2.2}$$

units from p. Denote by P'_R the set of vertices in G such that for every point in P_R there is one in P'_R with distance at most Δ. Then, an optimal Steiner tree T'_R for the set P'_R satisfies

$$\ell(T'_R) \leq \ell(T_R) + |P_R| \cdot \Delta.$$

If $|P'_R| > t$, the tree T'_R may not be contained in G. However, since we included in G optimal trees for any t-element subset of boundary vertices, we have an approximation by t-restricted Steiner trees available.

The exact approximation ratio of t-restricted Steiner trees under the octilinear metric and length restrictions has not yet been determined. However, this ratio cannot be worse than the ratio for t-restricted Steiner trees in graphs. For the latter, it is known that the ratio is $r_t = \frac{(r+1)2^r+\ell}{r2^r+\ell}$ for $t = 2^r + \ell$ [BD95]. Obviously, $r_t \geq 1$ is monotonously decreasing and converges to 1 for large t. Hence, we may choose t such that $r_t - 1 \leq \frac{\varepsilon}{2}$ for any given $\varepsilon > 0$.

For each $R \in \mathcal{R}_1$ we have

$$\ell(T_{app} \cap R) - \ell(T_{opt} \cap R) \leq (r_t - 1) \cdot \ell(T_{opt} \cap R) + r_t \cdot |P_R| \cdot \Delta.$$

Since $\sum_{R \in \mathcal{R}_1} |P_R| \leq 3n - 6$ (as we have at most $n-2$ rectangles in $\mathcal{R}_1$ and each Steiner point has 3 incident edges which may contribute to some P_R) and clearly $r_t \leq 2$, we obtain by (2.2)

$$\sum_{R \in \mathcal{R}_1} (\ell(T_{app} \cap R) - \ell(T_{opt} \cap R)) \leq \frac{\varepsilon}{2} \cdot \ell(T_{opt}) + \frac{10(3n-6)}{k+1} \cdot \ell(T_{opt}).$$

Replacing portions covered by $\mathcal{R}_2$. For rectangles in $R \in \mathcal{R}_2$ by Lemma 2.8 the following bound holds:

$$\ell(T_{app} \cap R) - \ell(T \cap R) \leq (4 - \sqrt{2}) \frac{b}{k+1}.$$

In the presence of (soft) obstacles, edges between terminals and/or Steiner points may be forced to bend several times. Hence, in general, an edge $e = (p, q)$ consists of a certain number of straight line segments, say $s_1, \dots, s_w$, and hits a number of obstacles. Let p_1 be the first common point of such an edge with some obstacle and p_2 be the last, respectively. Denote by p_1' and p_2' the nearest points in our graph G belonging to the same obstacle as p_1 and p_2, respectively. Since we have an almost shortest path between p_1' and p_2' in our graph (by step 5 of the graph construction), the path from p_1 to p_2 can be approximated by taking two short segments of length at most Δ on the boundary of the first and last obstacle plus this almost shortest path. Thus, this path is at most $2\Delta + \ell(T_{opt})/(k+1) \leq 11\ell(T_{opt})/(k+1)$ longer than the corresponding one in T_{opt}. The very first segment s_1 and the one or two last segments s_{w-1} and s_w plus the possible corner point between s_{w-1} and s_w are covered by up to four rectangles from $\mathcal{R}_2$. Hence by Lemma 2.8, the total error contributed by a single edge e is upper bounded by

$$4 \cdot (4 - \sqrt{2})\frac{b}{k+1} + \frac{11\ell(T_{opt})}{k+1} \leq \frac{71\ell(T_{opt})}{k+1}.$$

As there are at most $2n - 3$ edges in total, the overall error contributed by edges which are covered by rectangles in $\mathcal{R}_2$ is upper bounded by $\frac{71(2n-3)\ell(T_{opt})}{k+1}$. There are at most $n - 2$ Steiner points covered by rectangles in $\mathcal{R}_2$. These may contribute an additional error of $\frac{15(n-2)\ell(T_{opt})}{k+1}$.
Summing up, the total error can be bounded by

$$\begin{aligned} \ell(T_{app}) - \ell(T_{opt}) \leq\ & \frac{\varepsilon}{2}\ell(T_{opt}) + \frac{10(3n-6)}{k+1}\ell(T_{opt}) \\ & + \frac{71(2n-3)\ell(T_{opt})}{k+1} + \frac{15(n-2)\ell(T_{opt})}{k+1}, \end{aligned}$$

which simplifies to

$$\ell(T_{app}) - \ell(T_{opt}) \leq \frac{\varepsilon}{2}\ell(T_{opt}) + \frac{(187n - 303) \cdot \ell(T_{opt})}{k+1}.$$

If we choose $k := \lceil \frac{2\cdot(187n-303)}{\varepsilon} \rceil$, our graph contains a $(1 + \varepsilon)$-approximation and has polynomial size. Thus, we have shown the following theorem.

Theorem 2.21. *For a set $P \subseteq \mathbb{R}^2$ of terminals in the plane, a set $\mathcal{O}$ of rectangular soft obstacles with length restriction L with n terminals and obstacle vertices, and for every $\varepsilon > 0$ there is a graph of size $O(n^5)$ which contains a $(1 + \varepsilon)$-approximation of a minimum octilinear Steiner tree with length restriction L inside obstacles.*

In analogy to Section 2.2 we get the following corollary.

Corollary 2.22. *Let α denote the approximation guarantee for an algorithm solving the Steiner tree problem in graphs. For a set $P \subseteq \mathbb{R}^2$ of terminals in the plane, a set $\mathcal{O}$ of rectangular soft obstacles with length restriction L, and some $\varepsilon > 0$, there is an $(\alpha + \varepsilon)$-approximation of the octilinear Steiner tree problem with length restriction L inside obstacles.*

With the PTAS introduced by Borradaile et al. [BKMK07b] we get the following corollary:

Corollary 2.23. *For a set $P \subseteq \mathbb{R}^2$ of terminals in the plane, a set $\mathcal{O}$ of rectangular soft obstacles with length restriction L, and some $\varepsilon > 0$, there is a PTAS of the octilinear Steiner tree problem with length restriction L inside obstacles.*

2.5 Conclusion

In this chapter we studied approximation algorithms for the octilinear Steiner tree problem in the presence of hard octilinear and soft rectangular obstacles. For soft obstacles we constructed a graph which is guaranteed to contain paths at most $1 + \varepsilon$ so long as the shortest path between any pair of terminals (for arbitrary $\varepsilon > 0$). Such a graph is denoted as $(1 + \varepsilon)$-spanner. The construction leads to a $(2+\varepsilon)$-approximation by means of the minimum spanning tree heuristic of Mehlhorn [Meh88].

We introduced planar graphs for both the Steiner tree problem with hard octilinear and the one with soft rectangular obstacles (by two different constructions) which contain Steiner trees at most $1 + \varepsilon$ the length of minimum Steiner trees with hard and soft obstacles, respectively. By the polynomial time approximation scheme of Borradaile et al. [BKMK07b] we get polynomial time approximation schemes for both our problems. To the best of our knowledge these were the first polynomial time approximation scheme for octilinear Steiner trees with hard and soft obstacles. Quite recently, Müller-Hannemann and Tazari [MT07] published a PTAS for the λ-Steiner tree problem with soft obstacles with better running time.

For soft obstacles our asymptotically best approximation algorithm of the octilinear Steiner tree problem uses t-restricted Steiner trees. To achieve a polynomial running time, it was sufficient that the t-restricted Steiner ratio converges to 1 for large t. However, the convergence ratio for the Steiner ratio in general graphs (which we used) is very slow. We conjecture that the true convergence should be much faster. For comparison, we note that the t-restricted Steiner ratio in the rectilinear plane is $\frac{2t}{2t-1}$ for $t \geq 4$ [BR94, BDGW98]. Hence, it is an interesting open problem to find tighter or even exact bounds on the t-restricted Steiner ratio for octilinear Steiner trees.

As a major challenge it remains to find and analyze an approximation algorithm for the octilinear Steiner tree problem subject to obstacles which do not use (approximation) algorithms for the Steiner tree problem in graphs.

Chapter 3

Euclidean Group Steiner Trees

The group Steiner tree problem is a generalization of the ordinary Steiner tree problem. In the *Euclidean group Steiner tree problem* we are given a set of points P in the plane and a set of m connected regions. The regions contain the points of P. Each region or more precisely the points of P contained in the region are called *group* and the points in P are called terminals. We search for a network of minimum length which contains at least one terminal from every group.

Definition 3.1. *Let $P \subseteq \mathbb{R}^2$ be a set of points in the Euclidean plane and let $\{S_1, \ldots, S_m\} \subseteq P$ be m subsets of P. A* Euclidean group Steiner tree *is a Steiner tree, which contains at least one point of each set $\{S_1, \ldots, S_m\}$. A* minimum Euclidean group Steiner tree *is a Euclidean group Steiner tree of minimum length.*

The Euclidean group Steiner tree problem is to find a minimum Euclidean group Steiner tree. See Figure 3.1 for an example of a minimum Euclidean group Steiner tree.

The (group) Steiner tree problem can be also defined in graphs. Here we are given a graph G on n vertices with non-negative weights on the edges, and m subsets $\{S_1, \ldots, S_m\}$ of the vertices, and search for a subtree of G with minimum total length which contains at least one vertex of each of the m subsets.

As well as the standard Steiner trees group Steiner trees have applications in VLSI design. In the detailed routing phase the network can be connected to several equivalent ports. This is reflected by group Steiner trees. The equivalent ports are combined to groups and the Steiner tree have to connect only one terminal (or port) to the other groups.

Slavik [Sla97] presented a $3\rho/2$-approximation algorithm for the group Steiner tree problem under a fixed metric with ρ being the maximum number of nodes in a group. Garg et al. [GKR00] achieved a randomized $O(\log^3 n \log m)$-approximation

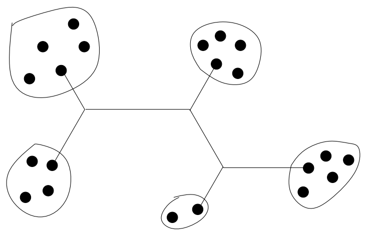

Figure 3.1: Minimum Euclidean group Steiner tree.

algorithm for the group Steiner tree problem in graphs which uses probabilistic tree embeddings together with randomized rounding.

If each set $\{S_1, \ldots, S_m\}$ contains exactly one point this problem is the ordinary Steiner tree problem. Since the Euclidean and also the Steiner tree problem in graphs are special cases of the Euclidean group Steiner tree and the Steiner tree problem in graphs, respectively, which are shown to be NP-hard [GGJ77, Kar72] the same holds for them.

Throughout this chapter, we consider an instance of the Euclidean group Steiner tree problem consisting of a finite set of points P and m subsets $S_1, \ldots, S_m$ of P. Each of the sets $S_1, \ldots, S_m$ is contained in a geometric region in the plane. A region is a connected and closed subset of the Euclidean plane. All these regions have to be disjoint. The objective is to find a minimum Steiner tree on a subset $P' \subseteq P$ such that $P' \cap S_i \neq \emptyset$ for $i = 1, \ldots, m$. For a set S of line segments (e. g., a Steiner tree or also a single line segment) we denote by $\ell(S)$ the total length of the line segments in S.

We restrict the regions to so called fat regions. The definition of fatness was introduced by Van der Stappen [vdS94] and also used by De Berg et al. [dBGK+05] and Elbassioni et al. [EFMS05] for the group traveling salesman problem.

Definition 3.2. *An region $O \subseteq \mathbb{R}^2$ is said to be an α-*fat *object if for any disk Θ which does not fully contain O and whose center lies in O, the area of the intersection of O and Θ is at least $1/\alpha$ times the area of Θ.*

As some examples, the plane $\mathbb{R}$ has fatness 1 and the half-plane has fatness 2. A disk is 4-fat. The *size* of an object is defined as the diameter of its smallest enclosing disk.

We give a $(1 + \epsilon)(9.093\alpha + 1)$-approximation for the regions being disjoint α-fat objects. The size of the objects can be varying.

3.1 Facts for Euclidean Steiner Trees

We need the following basic properties of minimum Euclidean Steiner trees. See for example the survey of Hwang, Richards and Winter [HRW92] for an overview.

Property 3.3. *Steiner points are incident with exactly three edges. The edges meet at angles of* $120°$.

Property 3.4. *Minimum Steiner trees for* n *terminals have at most* $n-2$ *Steiner points.*

Property 3.5. *No two edges of a minimum Steiner tree can meet at an angle less than* $120°$.

By the third property it follows that a terminal can be incident to at most three edges, which meet at angles of 120°.

3.2 A $(1+\epsilon)(9.093\alpha+1)$-Approximation of Euclidean group Steiner trees

In the following we give an algorithm for the Euclidean group Steiner tree problem where each region is α-fat which approximates the optimal tree by a factor of $(1+\varepsilon)(9.093\alpha+1)$. The algorithm we use for this approximation was introduced by Elbassioni et al. [EFMS05] for the Euclidean group traveling salesman problem. The traveling salesman problem asks for a shortest cycle containing each terminal exactly ones. This cycle is called tour. The Euclidean traveling salesman problem searches for a shortest tour of a set of points contained in m connected α-fat regions such that the tour contains at least one point in each of the m regions. They establish a $(1+\varepsilon)(9.1\alpha+1)$-approximation for the group traveling salesman problem. We transfer their ideas to the group Steiner tree problem to achieve the same approximation ratio. The main difference lies in the proof of the next lemma because a traveling salesman tour satisfies that every vertex is incident to exact two edges of the tour in contrast to a vertex or Steiner point of a minimum Steiner tree.

We use the following lemma to estimate the value of a shortest tree interconnecting a set of disjoint α-fat objects.

Lemma 3.6. *The length of a shortest tree connecting* c *disjoint* α*-fat objects in* $\mathbb{R}^2$ *is at least* $(c/\alpha-1)\pi\delta/4$*, where* δ *is the size of the smallest object.*

Proof. Let T be a tree that connects the c objects and let the center of a disk with diameter δ walk along this tree. See also Figure 3.2. We estimate the area A covered by the disk moving along the tree. At each point where the center of the

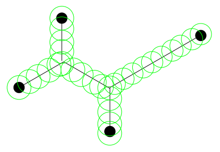

Figure 3.2: Covered area of a disk moving along T

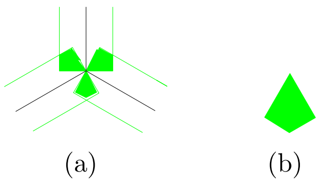

Figure 3.3: (a) Overlaps of the considered area (b) Three areas of a kite overlap

disk intersects the boundary of an object, the disk covers at least $1/\alpha$ fraction of it. That is, the total covered area is at least $A \geq c/\alpha \cdot \pi\delta^2/4$ of the disk.

To determine the upper bound, let t_1 be the overall number of terminals and t_2 and t_3 be the number of terminals of the tree with degree 2 and 3, respectively. We can roughly estimate the covered area by at most $\delta\ell(T)+(t_1-t_2-t_3)/2\cdot\pi\delta^2/4$ for $\ell(T)$ being the length of the tree T. But this overestimates the area at least at points of degree three by $3/16(\sqrt{3}+\sqrt{3}/3)\delta^2$. See Figure 3.3. The area of one kite is $1/16(\sqrt{3}+\sqrt{3}/3)\delta^2$. Therefore we can upper bound the covered area by

$$A \leq \delta\ell(T) + \frac{t_1-t_2-t_3}{2}\frac{\pi\delta^2}{4} - (t_1-t_2-2)\frac{3}{16}(\sqrt{3}+\frac{\sqrt{3}}{3})\delta^2.$$

By combining the upper and lower bound on the area we get

$$\begin{aligned}
& \frac{c}{\alpha}\frac{\pi\delta^2}{4} && \leq && \delta\ell(T) + \frac{t_1-t_2-t_3}{2}\frac{\pi\delta^2}{4} - (t_1-t_2-2)\frac{3}{16}(\sqrt{3}+\frac{\sqrt{3}}{3})\delta^2 \\
\Leftrightarrow\quad & \ell(T) && \geq && \frac{c}{\alpha}\frac{\pi\delta}{4} - \frac{t_1-t_2-t_3}{2}\frac{\pi\delta}{4} + (t_1-t_2-2)\frac{3}{16}(\sqrt{3}+\frac{\sqrt{3}}{3})\delta \\
& && = && \frac{c}{\alpha}\frac{\pi\delta}{4} + (\frac{3}{16}(\sqrt{3}+\frac{\sqrt{3}}{3}) - \frac{\pi}{8})(t_1-t_2)\delta + \frac{\pi\delta}{8}t_3 \\
& && && -\frac{3}{8}(\sqrt{3}+\frac{\sqrt{3}}{3})\delta \\
& && \overset{t_1-t_2\geq 2}{\geq} && \frac{c}{\alpha}\frac{\pi\delta}{4} + \frac{\pi\delta}{8}t_3 + (\frac{3}{8}(\sqrt{3}+\frac{\sqrt{3}}{3}) - \frac{\pi}{4})\delta - \frac{3}{8}(\sqrt{3}+\frac{\sqrt{3}}{3})\delta \\
& && \geq && (\frac{c}{\alpha}-1)\frac{\pi\delta}{4}.
\end{aligned}$$

□

Now we give a simple $(m - 1)$-approximation algorithm. See Algorithm 1. Let $SP(p_i, p_j)$ be a shortest path between two points p_i and p_j. The algorithm first choose an arbitrary point $p_1 \in S_1$. Then it determines for each set S_i, $1 \leq i \leq m$, the point $p_i \in S_i$ with minimum distance to p_1. Afterwards the algorithm constructs a tree be selecting the shortest paths between p_1 and the determined points.

Algorithm 1 $(m - 1)$-APPROXIMATION

```
Require: Sets $S_1, \dots, S_m \subseteq P$ contained in $\alpha$-fat disjoint regions.
1: Choose a point $p_1 \in S_1$.
2: Determine the points $p_i \in S_i$ $(i = 1, \dots, m)$ minimizing $\sum_{i=2}^{m} \ell(p_1, p_i)$.
3: Construct a tree $T := SP(p_1, p_2) \cup SP(p_1, p_3) \cup \ldots \cup SP(p_1, p_m)$.
4: return $T$.
```

Lemma 3.7. *The Algorithm is a $(m - 1)$-approximation for the group Steiner tree problem.*

Proof. Let T_{opt} be a minimum Steiner tree for the given instance. Each minimum Steiner tree contains paths from S_1 to S_i for all $i \in \{2, \dots, m\}$. We denote by $l_{opt}(S_i, S_j)$ the length of the path between S_i and S_j in the minimum Steiner tree. Therefore, $\sum_{i=2}^{m} \ell(p_1, p_i) \leq \sum_{i=2}^{m} l_{opt}(S_1, S_i) \leq (m - 1)\ell(T_{opt})$. □

Now we describe the algorithm to compute a $(1 + \varepsilon)(9.093\alpha + 1)$-approximation for the group Steiner tree problem. See Algorithm 2 DIAMETER for a detailed description. Let δ_i, $1 \leq i \leq m$ be the largest distance between any two points in S_i and $\ell(p, X) = \min_{x \in X} \ell(p, x)$ the minimum distance between a point p and a set of points X. The parameter δ_i, $1 \leq i \leq m$, is also called *diameter*. The algorithm first sorts the groups by increasing diameter and chooses an arbitrary point $p_1 \in S_1$. Afterwards, it determines for each set S_i, $1 \leq i \leq m$, the point with minimum distance to $\{p_1, \dots, p_{i-1}\}$. By using the approximation schemes of Arora [Aro98] and Mitchell [Mit99], in step 5 the algorithm constructs a $(1 + \varepsilon)$-approximation of a minimum Euclidean Steiner tree for the m selected points for an arbitrary $\varepsilon > 0$.

Theorem 3.8. *Algorithm 2* DIAMETER *is a $(1+\varepsilon)(9.093\alpha+1)$-approximation for the group Steiner tree problem with m groups contained in α-fat disjoint regions.*

Proof. We assume $m - 1 > 9.093\alpha + 1$ otherwise the algorithm outputs the solution of the $(m - 1)$-approximation. Denote the set of points chosen by the DIAMETER algorithm as $P' = \{p_1, \dots, p_m\}$ and let $p_i^* \in \{p_1, \dots, p_{i-1}\}$ be the point with minimum distance to p_i.

Let T_{opt} be a minimum Euclidean group Steiner tree for the instance and assume a closed walk along this tree. See Figure 3.4 for an illustration. Choose an

Algorithm 2 DIAMETER

```
Require: Sets $S_1, \ldots, S_m \subseteq P$ contained in $\alpha$-fat disjoint regions.
1: Sort the groups by increasing diameter $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_m$.
2: Choose a point $p_1 \in S_1$.
3: for all $i = 2, \ldots, m$ do
4:     Determine the point $p_i \in S_i$ minimizing $\ell(p_i, \{p_1, \ldots, p_{i-1}\})$.
5: Construct a $(1+\varepsilon)$-approximation $T$ of a minimum Steiner tree on the set of
   the $m$ chosen points.
6: return the minimum of $T$ and the output of Algorithm 1.
```

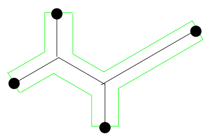

Figure 3.4: A walk along the tree

orientation of this walk. The rough procedure of the proof is to choose a number $c \in \{1, \ldots, m\}$ and to evaluate the length of the subtree of T_{opt} to connect c groups along the walk.

We define T_i, $1 \leq i \leq m$ to be the subtree of T_{opt} that connects c sets starting at group S_i and following the oriented walk.

We chose $c = \lceil \alpha(4/\pi+1) \rceil$. It holds $1 \leq c \leq m$ by assumption $m-1 > 9.093\alpha+1$. For $i \in \{1, \ldots, m\}$ let $S_{h(i)}$ be the set with smallest diameter among those c sets connected by T_i. By Lemma 3.6 and the choice of c one can estimate the length of T_i by

$$\ell(T_i) \geq (\frac{c}{\alpha} - 1)\frac{\pi\delta_{h(i)}}{4} \geq \delta_{h(i)}. \tag{3.1}$$

Since T_i starts at S_i, δ_i and the sets are sorted by increasing diameter we have $1 \leq h(i) \leq i$. We have to consider two cases.
If $h(i) = i$, that is S_i has smallest diameter, then by (3.1) it holds

$$\ell(T_i) \geq \delta_i. \tag{3.2}$$

If $h(i) < i$, the distance between any point in S_i and $p_{h(i)}$ is at least $\ell(p_i, p_i^*)$ which was the minimum distance between p_i and a point of $\{p_1, \ldots, p_{i-1}\}$ and p_i was chosen by the algorithm. Furthermore, the distance between any point of S_i and any point in $S_{h(i)}$ is at least $\ell(p_i, p_i^*) - \delta_{h(i)}$. And therefore, since T_i connects S_i with $S_{h(i)}$ we have $\ell(T_i) \geq \ell(p_i, p_i^*) - \delta_{h(i)}$. Together with (3.1) we get

$$\ell(T_i) \geq \max\{\delta_{h(i)}, \ell(p_i, p_i^*) - \delta_{h(i)}\} \geq \frac{\ell(p_i, p_i^*)}{2}. \tag{3.3}$$

We construct a tree for the set of points $P' = \{p_1, \dots, p_m\}$ chosen by the algorithm. Since the algorithm finds a Steiner tree approximating a minimum by a factor of $(1+\varepsilon)$, the algorithm gets an approximation at least as good as our construction.

We partition the set of points P' into two subsets fulfilling the two distinguished cases. That is, let H be the set of indices i for which $\ell(T_i) \geq \delta_i$ and $\bar{H} = \{1, \dots, m\} \setminus H$ the rest of the indices. Let T_H be the $(1+\varepsilon)$-approximation for the points $\{p_i \,|\, i \in H\}$. We get

$$\ell(T_H) \leq (1+\varepsilon)(\ell(T_{opt}) + \sum_{i \in H} \delta_i) \overset{(3.2)}{\leq} (1+\varepsilon)(\ell(T_{opt}) + \sum_{i \in H} \ell(T_i)). \tag{3.4}$$

To get a graph connecting also the points with indices in $\bar{H}$ we add for each $i \in \bar{H}$ the shortest path between p_i and p_i^* to T_H. By an inductive argument we can see that the tree is connected because $p_1 \in H$ and for any $i \in \bar{H}$ we know $p_i^* = p_j$ for some $j < i$. The total length of the so constructed graph, called T_{app} is

$$\begin{aligned}
\ell(T_{app}) &= \ell(T_H) + \sum_{i \in \bar{H}} \ell(p_i, p_i^*) \\
&\overset{(3.3),(3.4)}{\leq} (1+\varepsilon)(\ell(T_{opt}) + \sum_{i \in H} \ell(T_i)) + \sum_{i \in \bar{H}} 2\ell(T_i) \\
&\leq (1+\varepsilon)(\ell(T_{opt}) + 2\sum_{i=1}^{m} \ell(T_i)).
\end{aligned}$$

If we take the sum over all T_i then every edge is counted $2(c-1)$ times, therefore we get $2(c-1)\ell(T_{opt}) = \sum_{i=1}^{m} \ell(T_i)$ and with this

$$\ell(T_{app}) \leq (1+\varepsilon)(1 + 4(c-1))\ell(T_{opt}) < (1+\varepsilon)(9.093\alpha + 1)\ell(T_{opt}).$$

□

3.3 Conclusion

In this chapter we gave a $(1+\epsilon)(9.093\alpha + 1)$-approximation algorithm for the Euclidean group Steiner tree problem with groups lying inside disjoint α-fat regions. This was the first approximation algorithm for this problem. We applied the idea of Elbassioni et al. [EFMS05] for the Euclidean group traveling salesman problem to our problem. To this end, we have to consider the difference between a traveling salesman tour and a Steiner tree. More precisely, a traveling salesman tour fulfills the condition that each edge of such a tour is incident to exactly two terminals. This condition does not hold for Steiner trees. A Euclidean Steiner tree can contain Steiner points of degree three. This has to be taken into account for the bound of the tree length connecting a set of α-fat objects.

Elbassioni et al. [EFMS05] generalize their result for the Euclidean group traveling salesman problem to higher dimensions. The same can be done for the Euclidean group Steiner tree problem. Furthermore, they deal with intersecting objects and give a $O(1)$-approximation algorithm for this problem. We think that this result can be also adapted to Euclidean group Steiner trees.

It would be desirable to design an approximation algorithm with constant ratio and independent of the parameter α.

Chapter 4

Insights into the Structure of Manhattan networks

Connecting a given set of points with minimum total length is a key problem in VLSI design. In the routing process a set of components, e. g., transistors or gates, have to be connected by wires. The wires are allowed to run in two perpendicular directions. The wire length significantly affects the power consumption and the time to spread the signal across the chip. Minimum Steiner trees minimize the total wire length. In Chapter 2 and 3 we considered two different types of Steiner trees which are also related to VLSI-design. Manhattan networks impose an additional constraint. In contrast to Steiner trees they must contain a *shortest* path between each pair of points. This reflects the requirement to transmit signals between pairs of components on the chip fast. Manhattan networks are also defined in the *rectilinear metric* where one is allowed to use only horizontal and vertical lines.

Another application of Manhattan networks is described by Lam et al. [PLA02], who use a variant of the Manhattan network problem to align gene sequences. They ask only for certain node pairs to be connected by shortest paths and solve the problem by a modification of an algorithm of Gudmundsson et al. [GLN01].

Manhattan networks fit in the concept of spanners. Given a set P of points in the plane, a given metric, and a number $t \in \mathbb{R}$ with $t \geq 1$, a network is a t-spanner for P under the given metric, if for each pair of points $p, q \in P$, there exists a (p, q)-path in the network of length at most t times the distance between p and q under the appropriate norm. A Manhattan network is a 1-spanner in the rectilinear metric. Spanners are commonly known and first introduced for the Euclidean metric by Chew [Che89]. They are studied extensively, see for instance the survey of Eppstein [Epp00] or the book of Narasimhan and Smid [NS07]. Note that the Euclidean 1-spanner is the trivial complete graph.

It is unknown whether it is NP-hard to construct a minimum Manhattan network. Furthermore, it is not known whether a polynomial time approximation scheme exists. Gudmundsson et al. [GLN01] introduced an 8-approximation algorithm with running time $O(n \log n)$ and a 4-approximation algorithm running in time $O(n^3)$. Kato et al. [KIA02] proposed a 2-approximation algorithm with running time $O(n^3)$, however the proof of the correctness seems to be incomplete [BWWS06]. Chepoi et al. [CNV08] presented a 2-approximation algorithm based on LP-rounding. Their LP consists of $O(n^3)$ variables and constraints. In his PhD thesis Nouioua [Nou05] gave a 2-approximation algorithm that runs in $O(n \log n)$. Based on the primal-dual method, in contrast to the approach of Chepoi et al. the algorithm avoids to solve an LP explicitly; yet the proof relies on LP/IP methods. The thesis is written in French and unpublished until now. Benkert et al. [BWWS06] gave a 3-approximation algorithm with running time $O(n \log n)$. Seibert and Unger [SU05] presented an approximation algorithm and claimed that it yields a 1.5-approximation. As remarked by Chepoi et al. [CNV08] both the description of the algorithm and the performance guarantee are somewhat incomplete and not fully understandable. We show by a counterexample that an important intermediate step is incorrect, see Section 6.1.

Our aim is to present three new approximation algorithms in Chapter 6. Before we give insights in the structure of Manhattan networks in the next section, we define the problem and discuss first ideas to solve the problem. Afterwards in Section 4.2 we give important definitions we require later on. Particularly, we define staircases and prove some characteristics of them. In Section 4.3 we point out how to find the staircases of a given set of points. Afterwards, in Section 4.4 we introduce a further important term, called staircase boundary. With this term at hand, we give an overall scheme to compute Manhattan networks. In Section 4.5 we present an algorithm to compute staircase boundaries for staircases. Last, in Section 4.6 we give some insights about staircases affecting each other.

4.1 Towards Approximation Algorithms

In this section we define the Manhattan network problem and discuss some easy ideas to solve the Manhattan network problem.

A *rectilinear path* is a path consisting only of horizontal and vertical line segments.

Definition 4.1. *For a set $P \subseteq \mathbb{R}^2$ of n points in the plane, a* Manhattan network *on P contains a rectilinear shortest path between each pair of points in P. A* minimum Manhattan network *on P is a Manhattan network of minimum total length.*

The Manhattan network problem is to find a minimum Manhattan network. See Figure 4.1 (a) and (b) for examples of Manhattan networks in contrast to a rectilinear Steiner tree in Figure 4.1 (c).

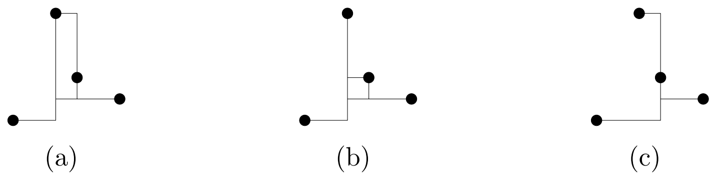

Figure 4.1: (a) A Manhattan network. (b) A minimum Manhattan network. (c) A rectilinear Steiner tree.

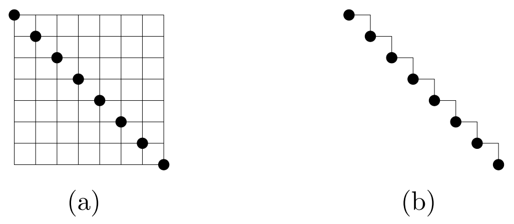

Figure 4.2: (a) The Hanan grid. (b) A minimum Manhattan network.

Our goal is to find algorithms solving the Manhattan network problem. Since the complexity status of the problem is still unknown we search after approximation algorithms. A first heuristic to get a Manhattan network is to take the whole *Hanan grid*. Recall the definition of the Hanan grid.

Definition 4.2. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane, the* Hanan grid *contains the boundary of the smallest rectangle $\mathcal{B}$ containing all points of P and all horizontal and vertical lines inside $\mathcal{B}$ touching a point of P.*

See Figure 4.2 (a) for the Hanan grid of a set of points. The whole Hanan grid of the example in Figure 4.2 has length $\Omega(n^2)$. A minimum Manhattan network for this instance has length $O(n)$ (see Figure 4.2 (b)). Thus we get the following lemma.

Lemma 4.3. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane, the Hanan grid is no constant factor approximation for the Manhattan network problem.*

The next idea to construct an approximation algorithm is that we do not select the whole Hanan grid but choose a set of point pairs such that it suffices to compute shortest paths for these point pairs to get a Manhattan network. We denote by p_x the x- and by p_y the y-coordinate of a point p. Each pair of points p and q spans a unique axis-parallel rectangle $R(p, q)$ with p and q as corners, the *enclosing rectangle* for p and q. See Figure 4.3. For a rectangle $R(p, q)$ we denote the points p and q as the points defining the rectangle $R(p, q)$.

Call $R(p, q)$ *critical* if it does not contain another point of P. To get shortest paths between all pairs of points, it suffices to consider critical rectangles only: If

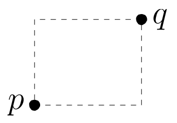

Figure 4.3: A critical rectangle.

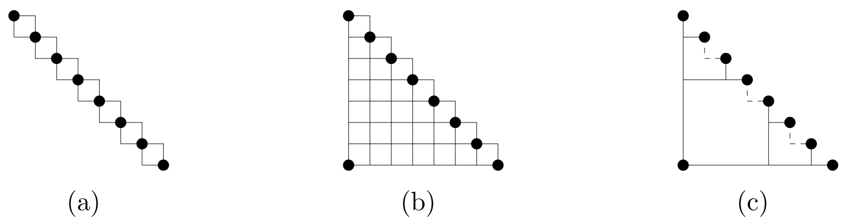

Figure 4.4: Taking the critical rectangles.

a rectangle $R(p, q)$ contains a further point $r \in P$ then we only need to consider the two rectangles $R(p, r)$ and $R(r, q)$ and to identify shortest paths between p and r and between r and q to get a shortest path between p and q. That is, to compute a Manhattan network it suffices to compute shortest path between all pairs of points forming a critical rectangle. Note that a minimum Manhattan network needs to contain line segments of length at least $|p_x - q_x|$ and $|p_y - q_y|$ in the corresponding dimension for a critical rectangle $R(p, q)$.

This leads to a second idea to compute a Manhattan network: We take all boundaries of critical rectangles. Obviously, we get a Manhattan network. See Figure 4.4 (a) for the resulting network of the point set already considered in Figure 4.2. For this instance the heuristic delivers a network with length twice the length of a minimum Manhattan network. But if we add just one more point we get many more critical rectangles to be considered. See Figure 4.4 (b). The network of the example has length $\Omega(n^2)$. A minimum Manhattan network for this instance is basically a binary tree and has length $O(n \log n)$ (see Figure 4.4 (c)). Again we get no constant factor approximation.

Lemma 4.4. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane, the set of all boundaries of critical rectangles is no constant factor approximation for the minimum Manhattan network problem.*

Thus, one has to think about more sophisticated algorithms to solve the Manhattan network problem. Almost all approximation algorithms as well as our for minimum Manhattan networks use staircases. Roughly speaking, a staircase is a set of points or line segments having the shape of a staircase. We define them in the next section. Afterwards we elaborate some important insights into the Manhattan network problem which we use in the next chapters.

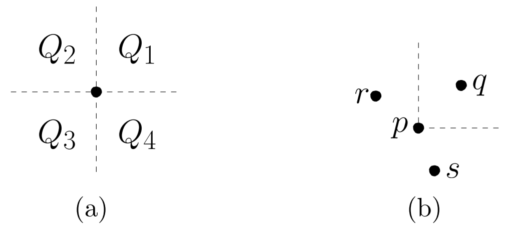

Figure 4.5: (a) The quadrants of p. (b) Global and local neighborhood.

4.2 Definitions

In this section we give further definitions we require later on.

For a point $p \in \mathbb{R}^2$ we denote by $Q_1(p), \ldots, Q_4(p)$ the four open quadrants of p. See also Figure 4.5 (a).

We call two points $p, q \in P$ (w. l. o. g. $p_x \leq q_x$ and $p_y \leq q_y$) *x-neighboring* if there is no further point r with $p_x < r_x < q_x$ and *y-neighboring* if there is no further point r with $p_y < r_y < q_y$.

For two points of P having the same x- or y-coordinate certainly a minimum Manhattan network as well as an approximation contains the direct line connecting these two points. Generally, the problem becomes easier. Since we would have to do some rather technically distinctions concerning this case, in the following we assume the coordinates of all points to be different.

Sometimes, we consider local neighborhood. More precisely, for a point $p \in P$ we consider the x- or y-neighboring point in one of its quadrants. In Figure 4.5 (b), the x-neighboring point of p in $Q_1(p)$ is the point q, whereas the globally x-neighboring points of p are the points r and s. We observe global neighborhood if not stated otherwise .

Almost all approximation algorithms as well as our for minimum Manhattan networks use staircases, but the definition of a staircase is not standardized. To get a clearer definition we define only one of four symmetric cases of a staircase shown in Figure 4.6. We will define the staircase type as shown in Figure 4.6 (a).

Definition 4.5. *Let $P \subseteq \mathbb{R}^2$ a set of points in the plane. For each point $p \in P$ the x-base point b^x is the x-neighboring point in $Q_3(p)$. The y-base point b^y is the y-neighboring point in $Q_3(p)$. Two points belong to the same* staircase *if they have the same base points.*

The definition of the x- and y-base points define an equivalence relationship between the points of P. The definition of a staircase can then be seen as an equivalence class.

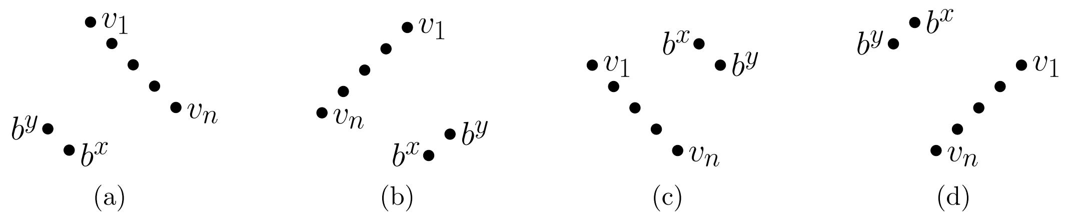

Figure 4.6: The four different cases of staircases.

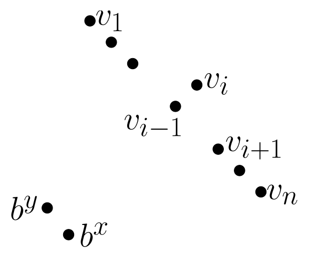

Figure 4.7: Proof of Lemma 4.6.

We also consider sequences with only one point p as a staircase sequence if none of the two base points of p belongs to a common staircase sequence with p. That is, for a staircase sequence $(v_1, \ldots, v_n)$ with base points b^x and b^y of type as in Figure 4.6 (a), formally the point v_2 is a base point for v_1 of a staircase of type as in Figure 4.6 (b). But they belong to the same staircase sequence with base points b^x and b^y and therefore do not form an own staircase. We denote the points v_1 and v_n of a staircase sequence $(v_1, \ldots, v_n)$ as the *outer points* of the sequence and the remaining points $\{v_2, \ldots, v_{n-1}\}$ as the *inner points*.

In the following we assume the sequence points $(v_1, \ldots, v_n)$ are sorted by increasing x-coordinate. The next lemma displays the typical structure of a staircase as depicted in Figure 4.6 (a).

Lemma 4.6. *The staircase sequence $(v_1, \ldots, v_n)$ of a staircase forms a sequence monotone increasing in x- and monotone decreasing in y-direction.*

Proof. Assume to the contrary that the sequence $(v_1, \ldots, v_n)$ is not monotone. That is, there exists a point v_i, $1 \le i \le n$, with $v_{i_y} > v_{i-1_y}$. See Figure 4.7. But then v_{i-1} is one of the base points of v_i contradicting that v_i has the same base points as v_{i-1}. $\square$

To make the definition of a staircase clearer, we point out important properties of staircases which clarify their structure. First, we define relevant regions of staircases.

Definition 4.7. *Let $(v_1, \ldots, v_n)$ be the sequence of a staircase with base points b^x and b^y. A* sequence rectangle *is a rectangle $R(v_i, v_{i+1})$, $1 \le i < n$, defined by*

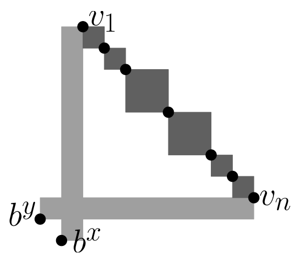

Figure 4.8: The dark shaded rectangles are the sequence rectangles. The remaining rectangles are the base rectangles.

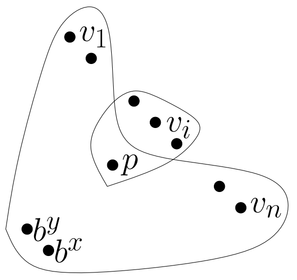

Figure 4.9: Proof of Lemma 4.8.

*two consecutive sequence points v_i and v_{i+1}. The x-*base rectangle *is the rectangle $R(v_1, b^x)$ defined by the first sequence point and the x-base point b^x. The y-*base rectangle *is the rectangle $R(v_n, b^y)$ defined by the last sequence point and the y-base point b^y.*

See Figure 4.8 for a diagram of the rectangles. Note, that the base rectangles overlap.

Now we will prove some properties of base and sequence rectangles.

Lemma 4.8. *Each sequence point forms a critical rectangle with both its base points.*

Proof. Let $(v_1, \ldots, v_n)$ be a staircase sequence with base points b^x and b^y. Assume to the contrary that a sequence point v_i, $1 \leq i \leq n$, does not form a critical rectangle with one of its base points. That is, the rectangle contains at least one further point $p \in P$. Then, the x- as well as the y-neighboring point in $Q_3(v_i)$ is p and not as required b^x and b^y, respectively. The staircase would split into at least two staircases (see Figure 4.9). □

We get the following corollary:

Corollary 4.9. *Each base rectangle $R(p, q)$ contains no further input point.*

After proving that sequence points form critical rectangles with their base points, we show that also sequence rectangles are critical.

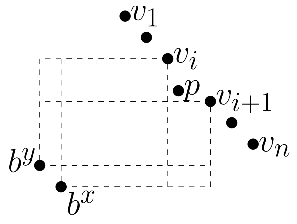

Figure 4.10: Proof of Lemma 4.10.

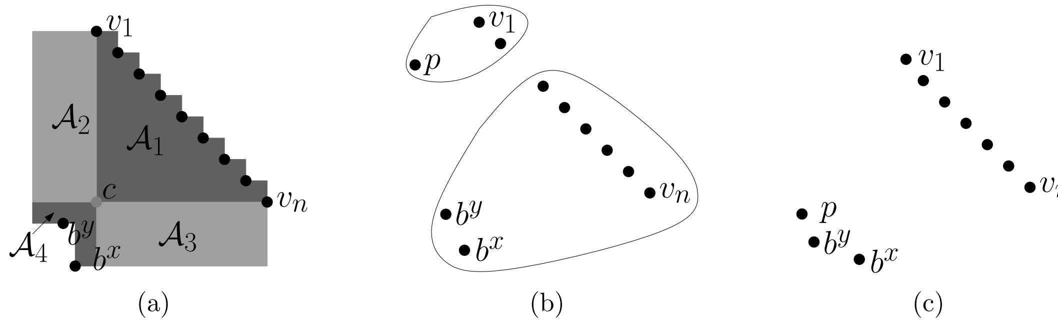

Figure 4.11: The areas $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ and $\mathcal{A}_4$ of a substaircase are empty.

Lemma 4.10. *For a staircase sequence $(v_1, \ldots, v_n)$ of a staircase, a sequence rectangle $R(v_i, v_{i+1})$, $1 \leq i < n$, contains besides v_i and v_{i+1} no further point of P.*

Proof. Assume to the contrary that a sequence rectangle $R(v_i, v_{i+1})$, $1 \leq i < n$, contains a further point $p \in P$, not being part of the sequence. If $R(v_i, v_{i+1})$ contains more points, let p be the point with smallest y-coordinate. See Figure 4.10.

By Lemma 4.8 the rectangles $R(v_i, b^x), R(v_i, b^y), R(v_{i+1}, b^x)$ and $R(v_{i+1}, b^y)$ do not contain a further point of P. Since p have smallest y-coordinate among all points in $R(v_i, v_{i+1})$ except v_{i+1}, the x- and y-neighboring points in $Q_3(p)$ are b^x and b^y, respectively. Thus p also pertain to the staircase sequence $(v_1, \ldots, v_n)$ in contradiction to the assumption. □

By Lemma 4.8 and Lemma 4.10 the area $\mathcal{A}_1 = R(v_1, v_2) \cup R(v_2, v_3) \cup \ldots \cup R(v_{n-1}, v_n) \cup R(v_2, c) \cup R(v_3, c) \cup \ldots \cup R(v_{n-1}, c)$ drawn in Figure 4.11 (a) contains besides the staircase points no further points of P. Also the areas $\mathcal{A}_2 = \{p \in \mathbb{R}^2 \,|\, p_x \leq v_{1_x} \text{ and } v_{n_y} \leq p_y \leq v_{1_y}\}$, $\mathcal{A}_3 = \{p \in \mathbb{R}^2 \,|\, p_y \leq v_{n_y} \text{ and } v_{1_x} \leq p_x \leq v_{n_x}\}$ and $\mathcal{A}_4 = \{p \in \mathbb{R}^2 \,|\, p_y \leq v_{n_y}, p_x \leq v_{1_x} \text{ and } (p_y \geq b^y_y \vee p_x \geq b^x_x)\}$ are empty. If one of the areas $\mathcal{A}_2$ or $\mathcal{A}_3$ contains a point $p \in P$, the staircase would split into at least two staircases because the x- and y-neighboring point in the third quadrant of at least one sequence point would be p and not b^x and b^y (see Figure 4.11 (b)). If a point $p \in P$ lies in the area $\mathcal{A}_4$, either b^x or b^y would not be base point but p (see Figure 4.11 (c)).

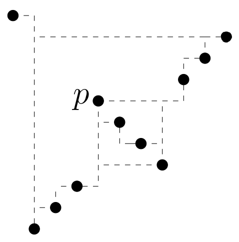

Figure 4.12: Proof of Lemma 4.11.

As depicted in Figure 4.6 there are four different types of staircases. However, a point cannot belong to four different staircases where each staircase sequence contains at least three points as we will prove by the following lemma.

Lemma 4.11. *Each point $p \in P$ belongs to at most three different staircase sequences each of them containing at least three points. If a point p belongs to three different staircase sequences then p is outer point of two them.*

Proof. Let p be a point of P. Trivially p can belong to at most one staircase of the same type because we identify for each staircase type exactly one x- and one y-base point. Assume a point $p \in P$ belongs to four different staircases. That is, p is part of two staircases S_1 and S_2 of type as in Figure 4.6 (a) and (c), respectively. Consider the two other staircase types (Figure 4.6 (b) and (d)) for which the sequence points proceed from bottom-left to up-right. First assume p is part of such a staircase sequence with base points lying top-left of the sequence points (Figure 4.6 (d)). See Figure 4.12. The point p cannot be an inner point of the sequences of S_1 and S_2 because then the top-left neighboring sequence point of p would be one of the top-left base points and thus does not form a staircase with sequence containing at least three points. Last p is also part of a staircase with base point lying bottom-right. By a similar argument as above the bottom-right neighboring sequence point of p is one of the two base points and thus these two points do not form a fourth staircase. Thus, we get that a point p can belong to at most three staircase sequences, each of them containing at least three points. If p belongs to three staircases (each of them containing at least three points), p is outer point of at least two sequences of them. □

After introducing the notion of staircases and pointing out some characteristics of them, in the next section we show how we can get all staircases.

4.3 Finding Staircases

In this section we describe how to find the staircases for a given set P of points in the plane. Our method is similar to that of Gudmundsson et al. [GLN01], but we use a slightly different definition of staircases. Our definition differs from

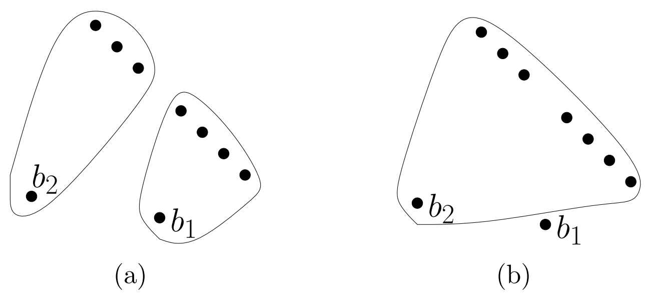

Figure 4.13: (a) Two staircases according to our definition. (b) Staircase according to Gudmundsson et al. [GLN01]

Gudmundsson's et al. definition that the points in Figure 4.13 according to our definition constitute two staircases because the point b_1 splits the sequence up into two sequences whereas Gudmundsson treat the sequence to belong only to one staircase with base point b_2.

As mentioned earlier there are four symmetric cases of staircases as shown in Figure 4.6. We describe how to find them only for the case (a). The other cases can be handled similar.

To find the staircases, for each point $p \in P$ we identify the x- and y-neighboring point in $Q_3(p)$, respectively. These points are by the definition of a staircase the two base points of p. For the two base points we keep the sequence point p in mind. After considering each point $p \in P$, we look out the perspective of the base points. For two points $\{b^x, b^y\}$ the points kept in mind are the sequence points of the staircase with base points b^x and b^y. See Algorithm 3 for a detailed description.

Algorithm 3 Finding Staircases

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
1: Set $BP = \emptyset$.
2: for each $p \in P$ considered from left to right do
3:     Let $b^x \in P$ be the $x$-neighboring point in $Q_3(p)$.
4:     Let $b^y \in P$ be the $y$-neighboring point in $Q_3(p)$.
5:     Set $SC(\{b^x, b^y\}) = SC(\{b^x, b^y\}) \cup \{p\}$.
6:     Set $BP = BP \cup \{b^x, b^y\}$.
   return $BP$ and $SC(\{b^x, b^y\})$ for each point pair $\{b^x, b^y\} \in BP$.
```

It is obvious that the algorithm finds all staircases because for each pair of base points $\{b^x, b^y\} \in BP$ the set $SC(\{b^x, b^y\})$ contains all appropriate sequence points.

The number of staircases for a set of n points can be upper bounded by $O(n)$ because each point can belong to at most 4 different staircase sequences and can also be base point to at most 4 different staircases. Thus, since a sweep over a point set requires sorting, we get the following running time for the algorithm:

Lemma 4.12. *The running time of Algorithm 3* Finding Staircases *for a set P of n points is $O(n \log n)$.*

Without sorting we achieve a linear running time.

Corollary 4.13. *The running time of Algorithm 3* Finding Staircases *for a set P of n sorted points is $O(n)$.*

4.4 Splitting into Staircases

Our algorithms for general Manhattan network problems which we will present in Chapter 6 act with akin strategies. They partition the global Manhattan network problem into disjoint local Manhattan network problems for staircases by inserting line segments which separate the staircases of each other. This general approach to partition the problem into a set of Manhattan network problems for staircases is used by all combinatorial approaches (see for example [BWWS06], [GLN01] or [KIA02]). For this, we compute a set of line segments containing a so-called (extended) staircase boundary for each staircase. These staircase boundaries partition our problem into a set of subproblems asking after Manhattan networks for staircases. Given such a partition we can appoint the minimum Manhattan network for each staircase in polynomial time as we show in Section 5.1. A 2-approximation of the Manhattan network problem for staircases is introduced in Section 5.2. Our algorithms presented in Chapter 6 guess in different ways the splitting into such local easy solvable subproblems.

In this section we define the notion of an (extended) staircase boundary and prove that it suffices to compute Manhattan networks for the staircases (given the boundaries) to get a Manhattan network for the whole instance. This provides a basis for our algorithms to compute a Manhattan network for a set of points in the plane which we present in Chapter 6.

Definition 4.14. *A* staircase boundary *for a sequence $(v_1, \ldots, v_n)$ of a staircase with base points b^x and b^y is a set of line segments defined in the following way: For any consecutive sequence points v_i and v_{i+1}, $1 \leq i < n$, the staircase boundary contains exactly one shortest (v_i, v_{i+1})-path. Furthermore, the staircase boundary contains exactly one shortest (v_1, b^x)- and one shortest (v_n, b^y)-path.*

The boundary of a staircase is not unique. See Figure 4.14 for different examples of staircase boundaries for the same point set.

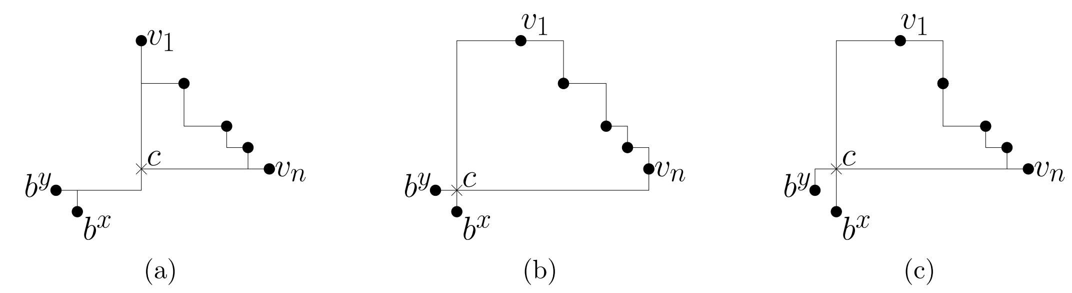

Figure 4.14: Different staircase boundaries for the same staircase.

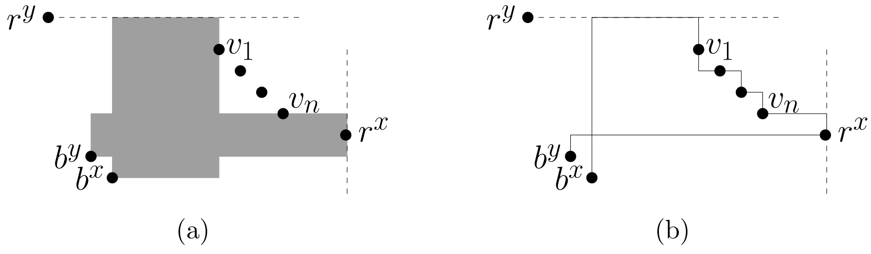

Figure 4.15: (a) Extended base rectangles. (b) Extended staircase boundary

Note that the shortest paths required for a staircase boundary lie inside the sequence and base rectangles. More precisely, for each sequence and each base rectangle exactly one shortest path between the defining points of the rectangle is required.

As used later, we want to define a variant of a staircase boundary which we call *extended staircase boundary.* The only difference in the definition for an extended staircase boundary is that we do not require a shortest path between the outer points and the base points. Nevertheless, there has to be a path lying in a certain area. To define this area we expand the base rectangles to so-called *extended base rectangles.*

Definition 4.15. *Let $(v_1, \ldots, v_n)$ be a staircase sequence with base points b^x and b^y. Let r^y be the y-neighboring point in $Q_2(v_1)$. If there exist no points in $Q_2(v_1)$, let $r^y = v_1$. Let r^x be the x-neighboring point in $Q_4(v_n)$. If there exist no points in $Q_4(v_n)$, let $r^x = v_n$. The* extended x-base rectangle *is the rectangle $R((v_{1_x}, r^y_y), b^x)$ defined by the point (v_{1_x}, r^y_y) and the x-base point b^x. The* extended y-base rectangle *is the rectangle $R((r^x_x, v_{n_y}), b^y)$ defined by the y-base point b^y and the point (r^x_x, v_{n_y}).*

See Figure 4.15 (a) for an example of extended base rectangles. Now, we can define the extended staircase boundary.

Definition 4.16. *An* extended staircase boundary *for a sequence $(v_1, \ldots, v_n)$ of a staircase with base points b^x and b^y is a set of line segments defined in the following* *way: For any consecutive sequence points v_i and v_{i+1}, $1 \leq i < n$, the extended staircase boundary contains exactly one shortest (v_i, v_{i+1})-path. Furthermore, the extended staircase boundary contains exactly one (v_1, b^x)-path inside the extended x-base rectangle and one (v_n, b^y)-path inside the extended y-base rectangle.*

See Figure 4.15 (b) for an example of an extended staircase boundary.

Since the region surrounded by the staircase boundary plays an important role, we want to name it by the following definition.

Definition 4.17. *Given a staircase S and an (extended) staircase boundary B. The* (extended) staircase area *of S, w. l. o. g. B, is the interior of B.*

If we choose the staircase boundary such that the staircase area is smallest, we call the area *smallest staircase area*. Note that the smallest staircase area of a staircase is unique. If we orient the (v_1, b^x)-path from v_1 to b^x and the (v_n, b^y)-path from v_n to b^y, the first intersection point of these two paths is called *cross point c* (see also Figure 4.14).

A Manhattan network requires for each point pair a shortest path between them. Thus, each minimum Manhattan network contains for each staircase a staircase boundary. Up to now we said how to define a boundary for a single staircase. Now we seek out the dependencies between the boundaries of different staircases. More precisely, a line segment can belong to two staircase boundaries. This occurs if sequence and/or base rectangles overlap. First, we point out that shortest paths required by a Manhattan network between the sequence and the base points run only through the smallest staircase area and the base rectangles.

Lemma 4.18. *Let $(v_1, \dots, v_n)$ be the sequence of a staircase with base points b^x and b^y. For each v_i, $1 \leq i \leq n$, the shortest paths between v_i and the base points b^x and b^y run solely through the region defined by the smallest staircase area and the base rectangles.*

Proof. Any shortest path between a sequence point v_i, $1 \leq i \leq n$, and one of the base points b^x and b^y lies completely inside the critical rectangles defined by v_i, b^x and b^y, respectively. The union of all these critical rectangles is exactly the region defined by the smallest staircase area and the base rectangles. $\square$

Now, it is easy to see that a staircase boundary separates the Manhattan network problem for the staircase from the rest of the problem.

Lemma 4.19. *Let B be the staircase boundary or extended staircase boundary of a staircase with sequence $(v_1, \dots, v_n)$ and base points b^x and b^y. The minimum Manhattan network for the staircase given B lies completely inside the staircase area defined by B.*

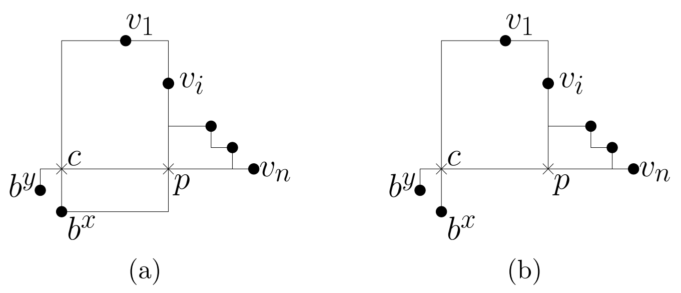

Figure 4.16: Proof of Lemma 4.19.

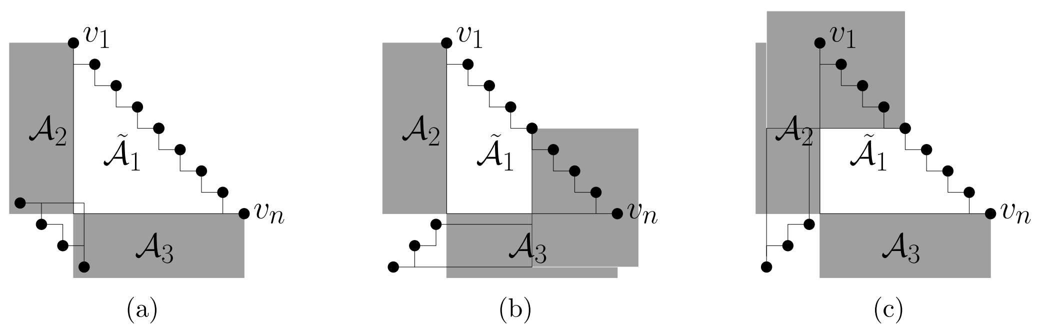

Figure 4.17: Proof of Lemma 4.20.

Proof. By Lemma 4.18 we must consider only the shortest paths running through the (extended) base rectangles. More precisely, we have to argue that no shortest path in the base rectangles takes course outside the staircase area. Assume there exists such a path from a point v_i, $1 \leq i \leq n$, with parts of the path lying outside the staircase area. See Figure 4.16. However, this path crosses the boundary segments at a point p, that is the path crosses the boundary segment either between v_1 and b^x or between v_n and b^y. Furthermore this intersection appear before the cross point c. But then we also have a shortest path to b^x and b^y using the shortest (v_i, p)-, (p, c)-, (c, b^x)- and (c, b^y)-path, respectively. We get a shorter Manhattan network if we delete all segments running outside the staircase area, in contrast to the minimality. □

Thus, if we search for a Manhattan network given the boundary segments of the staircases we have to consider for each staircase only the staircase area to achieve shortest paths. Now, we show that staircase areas do not overlap. First, we prove that smallest staircase areas are disjoint.

Lemma 4.20. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane, the interiors of the smallest staircase areas are disjoint.*

Proof. W. l. o. g. assume $(v_1, \ldots, v_n)$ be a staircase sequence belonging to a staircase as in Figure 4.17. As mentioned earlier the areas $\mathcal{A}_2$ and $\mathcal{A}_3$ are empty.

Assume the smallest staircase area $\tilde{\mathcal{A}}_1$ (which is somewhat smaller than $\mathcal{A}_1$) overlaps with another smallest staircase area of a staircase S'. There are four cases of the possible type of S' (see Figure 4.6). It is immediately reasonable that S' cannot be of the same type as the staircase S because then an overlapping cannot occur. Thus, consider the remaining three cases. For this purpose see Figure 4.17. We see that overlapping forces that at least one of the grey shaded areas $\mathcal{A}_2$ and $\mathcal{A}_3$ of the two staircases or the area $\tilde{\mathcal{A}}_1$ which have to be empty, contains points of P. Thus, the areas cannot overlap. $\square$

By Lemma 4.18 and Lemma 4.20 the only regions where paths between sequence and base points of different staircases can lie are the base and sequence rectangles. However, if we add line segments such that we achieve for each staircase a boundary we have associated each region of such a rectangle to exactly one staircase area.

Theorem 4.21. *Let $P \subseteq \mathbb{R}^2$ be a set of points in the plane. If $\mathcal{B}$ is a set of line segments such that for each staircase, $\mathcal{B}$ contains a staircase boundary or extended staircase boundary then each area surrounded by segments of $\mathcal{B}$ belong to exactly one staircase area.*

Proof. By Lemma 4.18 the Manhattan network of a staircase lies inside the smallest area and the (extended) base rectangles. By Lemma 4.19 since $\mathcal{B}$ contains an (extended) staircase boundary for each staircase the Manhattan network lies completely inside the staircase area. Even though a sequence or base rectangle can belong to two staircase areas if we insert a boundary we get a demarcation into two disjoint regions belonging to different staircase areas. $\square$

After showing that Manhattan networks of staircases are independently, we now show that it suffices to compute shortest paths between each point pair which belongs to the same staircase to get a Manhattan network for the complete instance.

Theorem 4.22. *Let $P \subseteq \mathbb{R}^2$ be a set of points in the plane and let $\mathcal{S}$ be the set of staircases of P. A network containing Manhattan networks for each staircase of $\mathcal{S}$ is a Manhattan network of P.*

Proof. Let MN be a network containing a Manhattan network for each staircase in $\mathcal{S}$ and let $p, q \in P$ two points. We show, that there exists a shortest path in MN between p and q proving that MN is a Manhattan network of P. We distinguish two cases.

Case 1: The points p and q belong to the same staircase.
Since MN contains a Manhattan network for each staircase in $\mathcal{S}$, there exists a shortest path between p and q in MN.

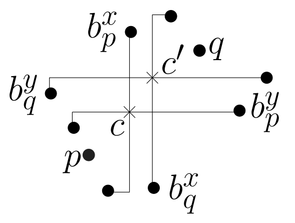

Figure 4.18: Proof of Theorem 4.22.

Case 2: The points p and q do not belong to the same staircase.
W. l. o. g. assume $p_x \leq q_x$ and $p_y \leq q_y$. See also Figure 4.18. Consider the staircase of the type depicted in Figure 4.6 (c) with p as a sequence point. Let b_p^x be the x-base point of this staircase and b_p^y the y-base point. Further on, consider the staircase of the type depicted in Figure 4.6 (a) with q as a sequence point. Analogously, let b_q^x be the x-base point of this staircase and b_p^y the y-base point. There are shortest paths in the Manhattan networks for staircases between the outer points of the staircases and their base points. That is, the only missing parts of a shortest (p, q)-path are shortest paths between p and q and the appropriate cross points which are also contained in Manhattan networks for staircases. See also Figure 4.18 with cross points c and c'.

Altogether this proves the theorem. □

Now we can specify the general strategy of our algorithms for the Manhattan network problem. Given a set of points in the plane, in a first phase we partition the set into the staircases and fix for each staircase the (extended) staircase boundary. After that, in a second phase we compute for each staircase a Manhattan network. See Algorithm 4 for a detailed description.

Algorithm 4 Manhattan Network

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
 Phase I:
1: Partition the problem into staircases and fix the (extended) staircase boundaries.
 Phase II:
2: for each staircase $S$ do
3:     Compute a Manhattan network for $S$ given the staircase boundary.
4: return the computed networks including the staircase boundaries.
```

By Theorem 4.21 and Theorem 4.22 the algorithm outputs a Manhattan network.

Theorem 4.23. *For a set $P \subseteq \mathbb{R}^2$ of points Algorithm 4* Manhattan Network *computes a Manhattan network for P.*

In Section 5.1 we show how to compute a minimum Manhattan network for a staircase in polynomial time. As a consequence, if we would know the right boundary for each staircase (i. e., the boundary chosen by a minimum Manhattan network), we could compute a minimum Manhattan network for the whole instance in polynomial time. Thus, the crucial point to find a minimum Manhattan network is to fix the right staircase boundaries. We get different approximation algorithms by varying methods to fix the staircase boundaries and to compute the Manhattan networks for the staircases. In the next chapter we will go into detail how to compute Manhattan networks for staircases. Afterwards, in Chapter 6 we present our different approximation algorithms for the Manhattan network problem. But before, we want to introduce a method to compute staircase boundaries which we use in Chapter 6 for all but one algorithm.

4.5 Computing Boundaries

There are several ways to partition the problem into staircases by fixing the boundaries. In this section we present a method which uses at most the length of a minimum Manhattan network inside the sequence and base rectangles. This construction is used by the algorithms introduced in Section 6.4 and 6.5. In Section 6.3 we use another kind of boundaries which consume at most three times the length of a minimum Manhattan network inside the sequence and base rectangles.

The choice of the staircase boundaries is the crucial point to get an optimal solution for a Manhattan network problem. See Figure 4.19 for an instance with two different chosen staircase boundaries. In this example there are two staircases, first the one with sequence $(p_1, \dots, p_7)$ and base point b and second the one with sequence $(q_1, \dots, q_5)$ and base point p_4. We see that the choice of the boundary for the first staircase affects the staircase area of the second staircase. In Figure 4.19 (b) we see that the two boundaries share segments. Generally, if we want to try all possibilities this leads to an exponential algorithm. To get a polynomial algorithm we have to guess a boundary. Our algorithm computes for each staircase a staircase boundary by considering neighboring points.

Obviously for two points $p, q \in P$ forming a critical rectangle a minimum Manhattan network contains line segments of length at least $|p_x - q_x|$ and $|p_y - q_y|$ in the respective dimension inside $R(p, q)$. Segments inside $R(p, q)$ of length $|p_x - q_x|$ covering all x-coordinates are denoted as *x-connection*. Segments inside $R(p, q)$ of length $|p_y - q_y|$ covering all y-coordinates are denoted as *y-connection*. See also Figure 4.20 (a) and (b).

By a *line segment l* we denote a horizontal or vertical line which is touched by a perpendicular line only at the endpoints of l. That is, a line is partitioned into

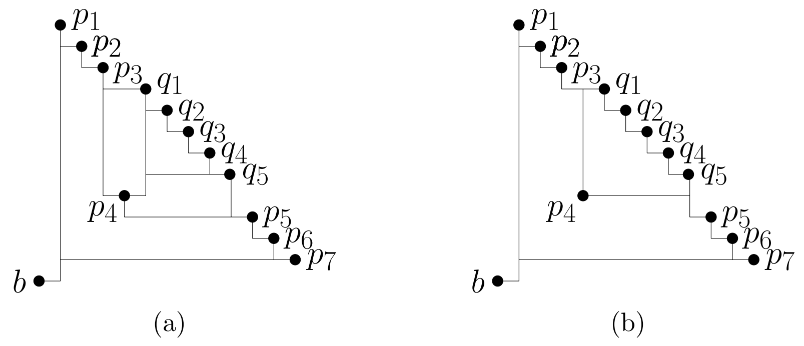

Figure 4.19: Nested staircases.

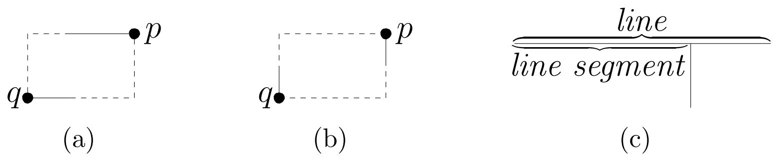

Figure 4.20: (a) An x-connection between the points p and q. (b) A y-connection between the points p and q. (b) Line and line segment.

several line segments if a perpendicular line segment touch or cross the line. See Figure 4.20 (c).

For an illustration of the following two definitions see again Figure 4.20 (a) and (b).

Definition 4.24. *An x-*covering *H of a set $P \subseteq \mathbb{R}^2$ of points in the plane is a set of x-connections for the points of P which we obtain in the following manner:*

1. *Let $H = \emptyset$.*
2. *Sweep over the points of P bottom-up. Let p be the current point and q be the previously processed point (i. e., p is the upper y-neighbor of q).*
3. *Add to H the portion of $[(q_x, p_y), p]$ that is complementary to $H \cap [q, (p_x, q_y)]$. (See Figure 4.20 (a).)*

Similarly, we define a *y-covering*:

Definition 4.25. *A y-*covering *V of a set $P \subseteq \mathbb{R}^2$ of points in the plane is a set of y-connections for the points of P which we obtain in the following manner:*

1. *Set $V = \emptyset$.*
2. *Sweep over the points of P from left to right. Let p be the current point and q be the previously processed point (i. e., p is the right x-neighbor of q).*

3. *Add to V the portion of $[(p_x, q_y), p]$ that is complementary to $H \cap [q, (q_x, p_y)]$. (See Figure 4.20 (b).)*

To get the one's best information about the required length of an x- and y-covering we define the area defined by neighboring points.

Definition 4.26. *The* neighboring point area $\mathcal{N}$ *of a set $P \subseteq \mathbb{R}^2$ of points in the plane is the union of all rectangles defined by neighboring points. That is,*

$$\mathcal{N} = \bigcup_{p,q \in P \text{ x- or y-neighboring}} R(p,q).$$

The total length of H and V is required in any Manhattan network.

Lemma 4.27. *The total length of $H \cup V$ is at most the length of a minimum Manhattan network inside the neighboring point area $\mathcal{N}$.*

Proof. Consider an x-covering H. We sweep over the points of P from bottom to top. If for two consecutive points there is not yet an x-connection inserted to the x-covering, then we insert it. We know that in any case in the minimum Manhattan network there must be a shortest path and therefore an x-connection between these two points. So we can insert an x-connection to the x-covering without inserting a segment of length larger than a segment between these points in the minimum Manhattan network. Since the two points are y-neighboring the considered rectangle is inside the neighboring point area $\mathcal{N}$.

The same holds for the y-covering in comparison to the vertical segments of a minimum Manhattan network. □

Our algorithm computes first an x- and y-covering H and V. See Algorithm 5 Compute Boundaries for a detailed description. Some of the lines of $H \cup V$ inside critical rectangles do not yet contribute to shortest paths of neighboring points. One end of such a line is neither a point of P nor incident to a perpendicular line. We shift these lines inside the appropriate rectangles to get a connection of the two defining points. See Steps 2 through 7. See Figure 4.21 and 4.22 for an example of such a covering and moving step. After these steps we have shortest paths between x- or y-neighboring points except for very special configurations which we handle in step 9.

To analyze the algorithm it is essential to keep in mind the next two facts.

Fact 4.28. *For two x-neighboring points $p, q \in P$ the set V constructed by step 1 of Algorithm 5* Compute Boundaries *contains a y-connection between p and q comprising of at most two lines. For two y-neighboring points $p, q \in P$ the set H constructed by step 1 of Algorithm 5* Compute Boundaries *contains an x-connection between p and q comprising of at most two lines.*

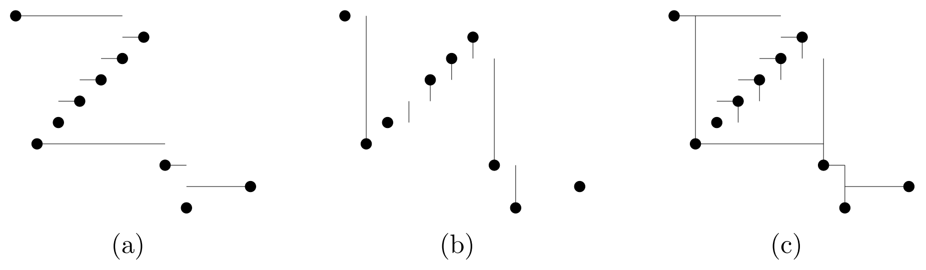

Figure 4.21: (a) Segments inserted by the x-covering. (b) Segments inserted by the y-covering. (c) Segments of the x- and y-covering together.

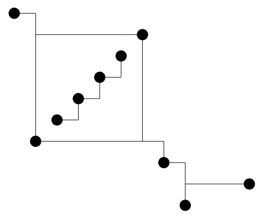

Figure 4.22: Network after the moving steps.

By Definition 4.24 of an x-covering at least one endpoint of a line segment of H is a point p of P. See Figure 4.23.

Fact 4.29. *Let $[p, p']$ be a line in H with $p \in P$ and $p' \notin P$. Then p' has an x-coordinate of a point below p' either y-neighboring to p or y-neighboring to a y-neighboring point of p.*

In almost the same matter, if an endpoint p' of a line segment of V is not in P and the other endpoint is a point $p \in P$ then the point p' has y-coordinate of a point on the left of p either x-neighboring to p or x-neighboring to an x-neighboring point of p.

Our aim is to prove that the algorithm computes for each staircase an (extended) staircase boundary. For this purpose we consider line segments inserted by step 1 together with the segments inserted by step 9 and show that by these line segments necessary shortest paths are established.

Note that for two x-neighboring points p and q we add vertical line segments and for two y-neighboring points we add horizontal line segments. Nevertheless, in the rectangle defined by two x-neighboring points lies also an x-connection and in the rectangle defined by y-neighboring points lies also a y-connection.

Lemma 4.30. *Let MN be the set of line segments computed by Algorithm 5* Compute Boundaries. *For two x-neighboring points $p, q \in P$ the set MN contains at least one x-connection inside $R(p, q)$. Each x-connection in $R(p, q)$*

Algorithm 5 Compute Boundaries

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
 1: Let $H$ be an $x$-covering and $V$ be a $y$-covering.
 2: for each line segment $l_h \in H$ considered from top to bottom do
 3:     if one endpoint of $l_h$ is neither a point of $P$ nor a point of a line segment of $V$ then
 4:         Move $l_h$ downwards until it hits either a point of $P$ or an endpoint of a line segment of $V$.
 5: for each line segment $l_v \in V$ considered from right to left do
 6:     if one endpoint of $l_v$ is neither a point of $P$ nor a point of a line segment of $H$ then
 7:         Move $l_v$ to the left until it hits either a point of $P$ or an endpoint of a line segment of $H$.
 8: Set $MN = H \cup V$.
 9: for all $x$- or $y$-neighboring points $p, q \in P$ do
10:     if there is no shortest $(p, q)$-path in $MN$ then
11:         Add to $MN$ the shortest line segment needed to establish a shortest $(p, q)$-path.
12: return $MN$.
```

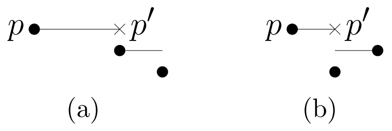

Figure 4.23: An example for Fact 4.29.

consists of exactly one segment of length $|p_x - q_x|$. For two y-neighboring points $p, q \in P$ the set MN contains at least one y-connection inside $R(p, q)$. Each y-connection in $R(p, q)$ consists of exactly one segment of length $|p_y - q_y|$.

Proof. W. l. o. g. let $p, q \in P$ are x-neighboring and let $p_x \leq q_x$. There exists at least one x-connection inside $R(p, q)$ because either p and q are also y-neighboring or there exists at least one point r with $p_y < r_y < q_y$ establishing an x-connection. Assume there is an x-connection for two y-neighboring points $p', q' \in P$ with $R(p', q')$ intersecting the rectangle $R(p, q)$ and consisting of two line segments inside $R(p, q)$. Then by Fact 4.29 there exists a point r with $p_x < r_x < q_x$. But then p and q would not be x-neighboring. □

Now we prove if the y-connection for two x-neighboring points $p, q \in P$ in $R(p, q)$ consists of two line segments then p and q are connected by a shortest path after step 7.

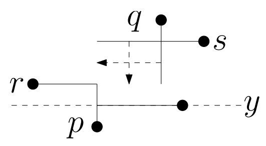

Figure 4.24: Proof of Lemma 4.31.

Lemma 4.31. *Let MN be the set of line segments after step 8 of Algorithm 5* Compute Boundaries*. If for two x-neighboring points $p, q \in P$ the y-connection consists of two line segments or for two y-neighboring points $p, q \in P$ the x-connection consists of two line segments then MN contains a shortest (p, q)-path.*

Proof. W. l. o. g. let $p, q \in P$ are x-neighboring and let $p_x \leq q_x$ and $p_y \leq q_y$. Assume p and q are not connected by a shortest path. Let p' be the other endpoint of the line segment incident to p of the y-connection and q' the endpoint of the line segment incident to q. See Figure 4.24. By Fact 4.29 there exists a point r in $Q_2(p)$ with y-coordinate $r_y = p'_y = q'_y$. This point r induces an x-connection inside a rectangle $R(r, s)$ for a point s either being q or a point $s \in Q_4(q)$. By Lemma 4.30 the x-connection consists inside $R(p, q)$ of exactly one line segment. Furthermore the x-connection inside $R(p, q)$ lies at the height of s. Either we move this line segment down to p' or the line segment $[(q_x, s_y), q']$ to the left to p' to establish a shortest (p, q)-path. This moving is done through steps 2 to 7 of Algorithm 5. □

With this at hand we now show that we get the desired extended staircase boundaries. First we prove that two neighboring sequence points are connected by a shortest path.

Lemma 4.32. *Let MN be the set of line segments computed by Algorithm 5* Compute Boundaries*. For each staircase sequence the set of line segments in MN contains a shortest path for consecutive staircase sequence points.*

Proof. Let $(v_1, \ldots, v_n)$ be a staircase sequence. Let v_i and v_{i+1}, $1 \leq i < n$, be two consecutive staircase sequence points with $v_{i_x} \leq v_{{i+1}_x}$ and $v_{i_y} \geq v_{{i+1}_y}$ with base points bottom-left (of type as in Figure 4.6 (a)). If v_i and v_{i+1} are x- or y-neighboring then they are connected by a shortest path. Thus, assume v_i and v_{i+1} are neither x- nor y-neighboring. Let v^x be the x-neighboring point of v_{i+1} on the left of v_{i+1} and v^y be the y-neighboring point of v_i below v_i. Since v_i and v_{i+1} belong to the same staircase, v^x lies above v_i and v^y on the right of v_{i+1}. See Figure 4.25. By the neighborhood of v^x and v_{i+1} there is a shortest (v^x, v_{i+1})- and by the neighborhood of v^y and v_i a shortest (v^y, v_i)-path. Together these paths establish a shortest (v_i, v_{i+1})-path. □

With the last two lemmata at hand we can finish our statement that the set MN contains an extended staircase boundary for each staircase.

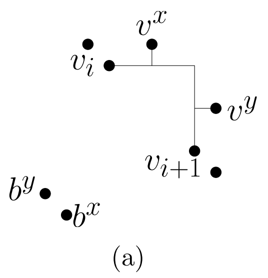

Figure 4.25: Proof of Lemma 4.32.

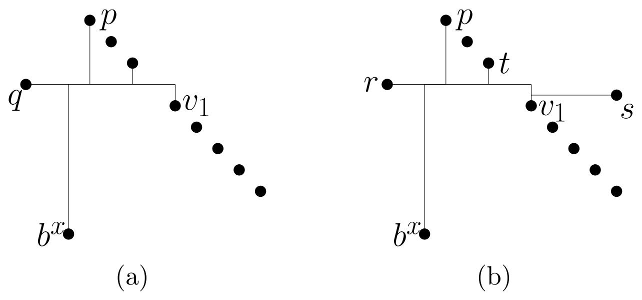

Figure 4.26: Proof of Theorem 4.33.

Theorem 4.33. *Let MN be the set of line segments computed by Algorithm 5* Compute Boundaries. *For a staircase with base points b^x and b^y the set MN contains paths from b^x and b^y to the outer points of the staircase sequence in the appropriate extended base rectangles.*

Proof. Let $(v_1, \ldots, v_n)$ be a staircase sequence with base points b^x and b^y. If b^x and v_1 are x-neighboring then they are connected by a shortest path. Thus assume b^x and v_1 are not x-neighboring. Let p be the point x-neighboring to b^x on the right of b^x. See Figure 4.26. Since p is x-neighboring to b^x we know that there exists a shortest (p, b^x)-path. Consider the point q y-neighboring to v_1 above v_1 We distinguish two cases. First assume q lies to the left of v_1. Since v_1 is the outer point of the staircase and p is x-neighboring to b^x the point q lies on the left of b^x. Again, there exists a shortest (q, v_1)-path due to the neighborhood of q and v_1. Together with the shortest (p, b^x)-path we get a (v_1, b^x)-path in the extended x-base rectangle. See Figure 4.26 (a).

Now assume the point q lies to the right of v_1. Since p does not belong to the staircase sequence $(v_1, \ldots, v_n)$ there is a point above v_1 and to the left of b^x. Let r be the one with smallest y-coordinate. The point r is y-neighboring below to a point s on the right of v_1 (s can be q or a point above q). There exists a shortest (r, s)-path. Since b^x is x-neighboring to v_1 in $Q_3(v_1)$ but is not globally x-neighboring the point t x-neighboring to v_1 on the left of v_1 lies above r. Again there exists a shortest (v_1, t)-path. Together with the shortest (p, b^x)-

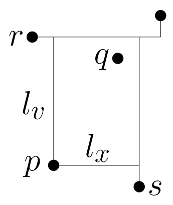

Figure 4.27: Proof of Theorem 4.35.

and (r, s)-path we get a (v_1, b^x)-path inside the extended x-base rectangle. See Figure 4.26 (b).

In almost the same manner we can prove that we get a (v_n, b^y)-path inside the extended y-base rectangle. These two paths together establish also a (v_1, b^y)- and a (v_n, b^x)-path closing the proof. □

We can conclude with the following corollary:

Corollary 4.34. *The set MN of line segments computed by Algorithm 5* Compute Boundaries *contains for each staircase an extended staircase boundary.*

Now we prove that we use at most the length of a minimum Manhattan network inside the neighboring point area.

Theorem 4.35. *Let MN be the set of line segments computed by Algorithm 5* Compute Boundaries. *The total length of the line segments is at most the length of a minimum Manhattan network inside the neighboring point area $\mathcal{N}$.*

Proof. By Lemma 4.27, the length of the x- and y-coverings H and V have length at most the length of a minimum Manhattan network inside $\mathcal{N}$. Processing steps 2 to 7 does not increase the length of the segments in MN. Now, we will consider the line segments added by step 9. W.l.o.g. let $p, q \in P$ are x-neighboring and let $p_x \leq q_x$ and $p_y \leq q_y$. See for the following explanation Figure 4.27. Assume after step 7 p and q are not connected by a shortest path. By Lemma 4.31 the y-connection consists of one segment l_v. The segment l_v is either incident to p or to q. W.l.o.g. let l_v incident to p. By Lemma 4.30 there exists at least one x-connection inside $R(p, q)$ and each x-connection consists of exact one line segment running completely through $R(p, q)$. The topmost x-connection lies below q (otherwise we have already a shortest (p, q)-path). Furthermore, l_v ends at a horizontal line segment l_h of H (otherwise we could move the part of l_v above the topmost x-connection to establish a shortest (p, q)-path). The presence of the segment l_h indicates that there exists a point $r \in P$ in $Q_2(p) \cap Q_2(q)$. The segments l_h and l_v are part of a shortest (p, r)-path. For this path it is necessary that l_v is incident to p and cannot be moved to the right to the width of q. Now consider an x-connection l_x inside $R(p, q)$. The line segment l_x is caused by a point $s \in P$ in $Q_4(q)$ and is (possibly together with a segment of l_v or other segments) necessary for a shortest (p, s)-path. The segment l_x cannot be moved upwards (to the height of q) to establish a shortest (p, q)-path because we would lose the shortest (p, s)-path. We see that with the given x- and y-connections of the x- and y-coverings H and V we cannot get simultaneous shortest (p, r)-, (p, s)- and (p, q)-paths (see again Figure 4.27). Thus, if we add to MN the shortest line segment to achieve a shortest (p, q)-path the length of MN is already at most the length of the line segments of a minimum Manhattan network to connect points being x- or y-neighboring (that is, at most the length of line segments inside $\mathcal{N}$). □

Thus, Algorithm 5 Compute Boundaries fulfills step 1 of Algorithm 4 Manhattan Network. Last we want to mention the running time of the algorithm.

Lemma 4.36. *The running time of Algorithm 5* Compute Boundaries *for a set P of n points is $O(n \log n)$.*

Proof. First of all we must sort the points of P. This can be done in time $O(n \log n)$. The sweeps to get x- and y-coverings can be transformed in $O(n)$. Just as the sweeps used by steps 2 through 7. By Lemma 4.12 we can compute with Algorithm 3 Finding Staircases all staircases in time $O(n \log n)$. This yields the total running time for the algorithm of $O(n \log n)$. □

Together with Corollary 4.13 the following corollary is an easy observation.

Corollary 4.37. *The running time of Algorithm 5* Compute Boundaries *for a set P of n* sorted *points is $O(n)$.*

4.6 More Insights

In this section we present further insights into the Manhattan network problem. We want to sensitize the reader to the dependencies of the staircase boundaries of different staircases. If we fix the staircase boundary of each staircase, we can answer the Manhattan network problem in polynomial time as we will present in Section 5.1. That is, the crucial point to get a minimum Manhattan network is to choose the right staircase boundaries. To approach the problem we will give a short overview about possible intersections of sequence and base rectangles of different staircases which are the areas where the staircase boundaries lie.

See Figure 4.28 (a) for an example of overlapping sequence rectangles. (In all examples we depict the sequence and base rectangles by dashed lines.) We get two staircases, the first with sequence $(v_1, \ldots, v_6)$ and base points b^x and b^y and the second with sequence $(v'_1, v_4, \ldots, v_6)$ and base point b. The sequence rectangle $R(v_3, v_4)$ overlaps with the sequence rectangle $R(v'_1, v_4)$ without being identical.

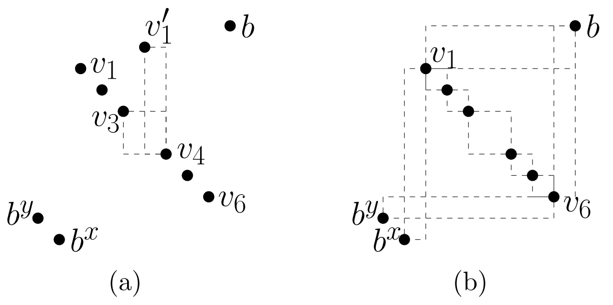

Figure 4.28: Overlapping sequence rectangles.

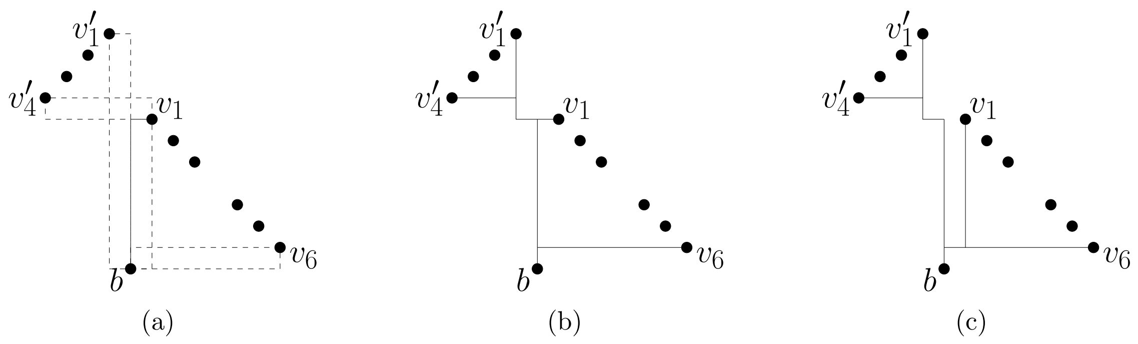

Figure 4.29: Base rectangles with the same boundary.

But there are also cases where the overlapping rectangles are identical. See Figure 4.28 (b) for an example. The sequence $(v_1, \ldots, v_6)$ belongs to two staircases, the first with base points b^x and b^y and the second with base point b.

In Figure 4.29 we see a typical situation for the intersection of two base rectangles. We have to staircase sequences $((v_1, \ldots, v_6)$ and $(v_1', \ldots v_4'))$ both with the same base point b. The base rectangles $R(v_1, b)$ and $R(v_1', b)$ overlap. In Figure 4.29 (b) and (c) we depict possible boundaries inside the base rectangles.

In Figure 4.30 and 4.31 we see an example of the possible complex overlappings of base rectangles. We depict in Figure 4.30 only the base rectangles for x-base points and in Figure 4.31 the base rectangles for y-base points of the same instance.

Last in Figure 4.32 we depict an example with overlapping sequence and base rectangles. In the example there are two staircases. First the one with sequence $(v_1, \ldots, v_5)$ and base point b and second the one with sequence $(v_1', \ldots, v_4')$ and base point v_3. The sequence rectangle $R(v_2, v_3)$ overlaps with the base rectangle $R(v_1', v_3)$ and the sequence rectangle $R(v_3, v_4)$ with the base rectangle $R(v_4', v_3)$.

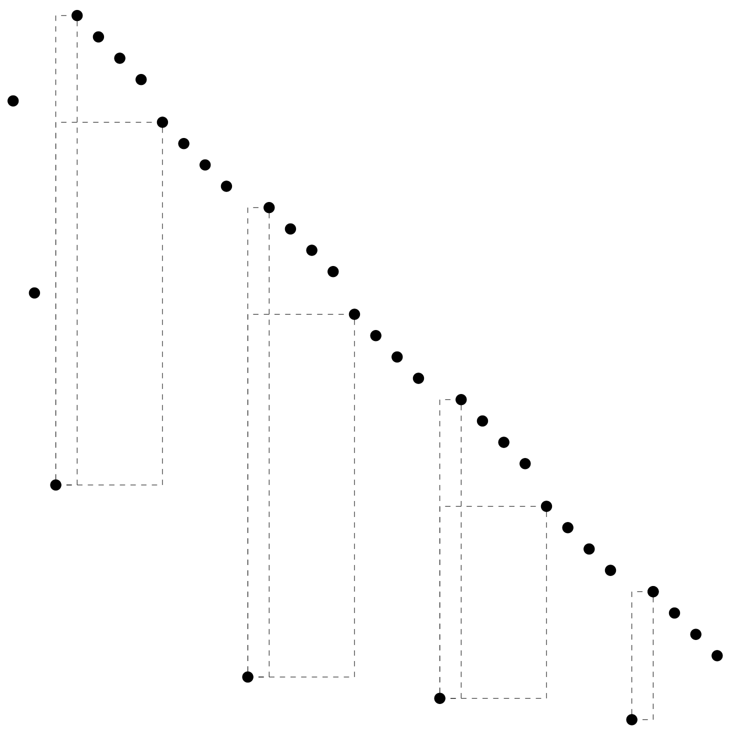

Figure 4.30: Overlapping x-base rectangles.

4.7 Conclusion

In this chapter we introduced the Manhattan network problem which asks after a rectilinear network containing a shortest rectilinear path between each pair of terminals. This problem can be seen as a variant or extension of the Steiner tree problem. We introduced the notion of staircases which play an important role in solving the Manhattan network problem. More precisely, we can partition the Manhattan network problem into a set of Manhattan network problems for staircases. We pointed out how to find the staircases and presented an algorithm computing a Manhattan network on the base of staircases. We presented an easy sweep line approach to get a set of line segments containing a boundary for each staircase. Furthermore, we bound the length of this set of line segments by the length of a Manhattan network inside the area defined by x-or y-neighboring points. These definitions and results are used in the next chapters. Last we gave a few examples of overlapping base and sequence rectangles to point out the dependencies of the staircases and their boundaries among each other.

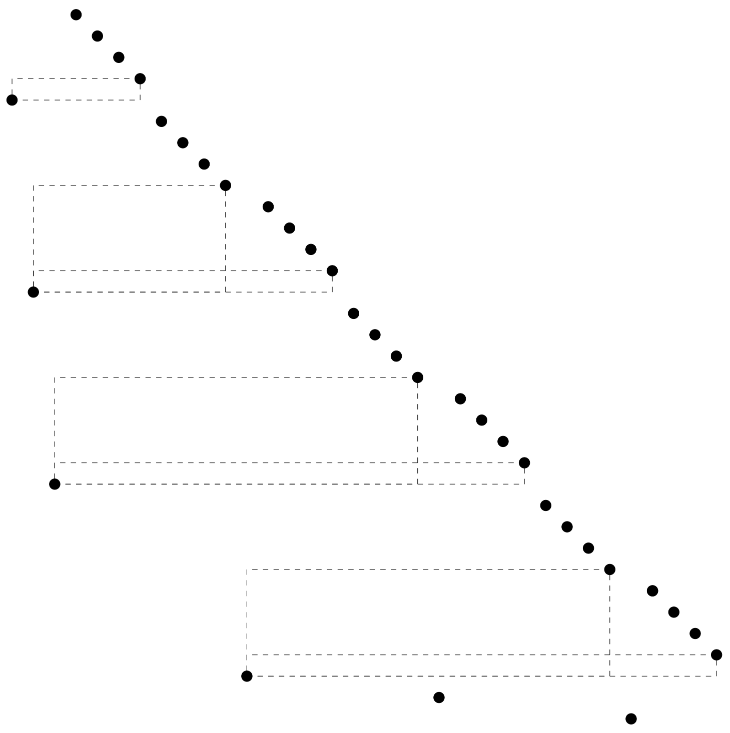

Figure 4.31: Overlapping y-base rectangles.

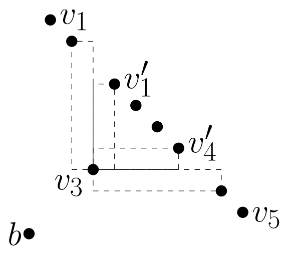

Figure 4.32: Base and sequence rectangles with the same boundary.

Chapter 5

Manhattan Networks of Staircases

In this chapter we study staircases. In contrast to the general Manhattan network problem we can find minimum Manhattan networks for staircases in polynomial time.

We concern with the question getting Manhattan networks for input points forming a staircase. We additionally get the boundary of the staircase as input. In Section 5.1 we present a dynamic programming approach to compute a minimum Manhattan network in time $O(n^3)$. Afterwards, we state a 2-approximation for the problem in Section 5.2.

Staircases are used by almost all approximation algorithms for the minimum Manhattan network problem. They were first used for Manhattan networks by Gudmundsson et al. [GLN01] to achieve an 8- and 4-approximation. They prove that given a staircase boundary a rectangulation of the polygon defined by the boundary with minimum total length is a minimum Manhattan network for the points defining the staircase boundary. Lingas et al. [LPRS82] show that a minimum rectangulation of a rectilinear polygon with n vertices can be computed in time $O(n^4)$. They state, that for a special case of so-called *histograms* the running time can be reduced to $O(n^3)$. We achieve this result by computing a minimum Manhattan network directly without the detour to the rectangulation. Also they introduce a 2-approximation for rectangulations which we point out directly to Manhattan networks. Benkert et al. [BWWS06] introduced a similar approximation result as our but by the algorithms presented in Chapter 6 we get other properties of the staircase area defined by the given boundary. We adopt the algorithm and statements to our slightly other definition of a staircase and to our conditions.

The scheme of Algorithm 4 Manhattan Network to solve the Manhattan network problem is to partition the problem into Manhattan network problems

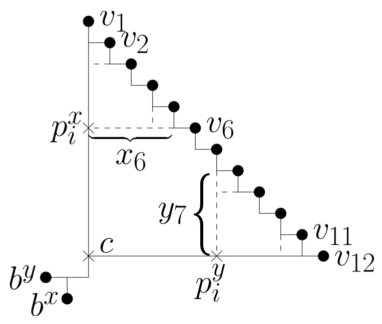

Figure 5.1: The definition of x_i and y_i.

of staircases. For this for each staircase a boundary is computed (step 1 of Algorithm 4). We can compute the staircase boundaries with Algorithm 5 Compute Boundaries. In this chapter we deal with the problem to compute Manhattan networks for staircases given the staircase boundary. Note, if we have to compute a Manhattan network for only one staircase it is rather easy to assign a staircase boundary. In that case we would select the staircase boundary with smallest staircase area. In Section 5.1 we introduce an algorithm which solves the problem to optimality by a dynamic program in time $O(n^3)$ (for n sequence points). Afterwards, in Section 5.2 we present a 2-approximation algorithm for the problem which runs in time $O(n \log n)$.

Given the staircase boundary of a staircase with sequence $(v_1, \dots, v_n)$, for a point v_i, $1 \leq i \leq n$, of the staircase sequence we denote by x_i the missing line segment of a shortest path to the vertical left boundary segment, by y_i we denote the missing line segment of a shortest path to the horizontal bottom boundary segment. See Figure 5.1 for an illustration. Note, that we do not count the length of the possibly used boundary segments between v_i and the left and right boundary, respectively. Let p_i^x be the intersection of x_i with the vertical left boundary segment and p_i^y the intersection of y_i with the horizontal bottom boundary segment.

5.1 An Exact Algorithm for Staircases

In this section we present an algorithm which computes for the given boundary a minimum Manhattan network for the staircase. We need the following property of a minimum Manhattan network for our algorithm.

Lemma 5.1. *Let $(v_1, \dots, v_n)$ be a staircase sequence with base points b^x and b^y. There exists a minimum Manhattan network MMN such that for at least one $i \in \{1, \dots, n\}$ the segment x_i and the segment y_{i+1} is contained in MMN.*

Proof. Let MMN be an arbitrary minimum Manhattan network. Each minimum Manhattan network for the staircase contains a shortest (v_1, b^x)- and a shortest

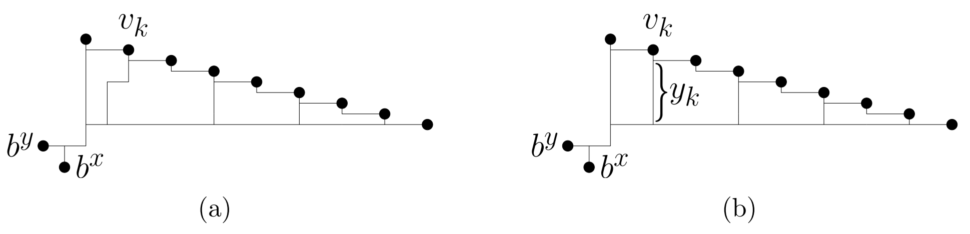

Figure 5.2: Proof of Lemma 5.1 (first case).

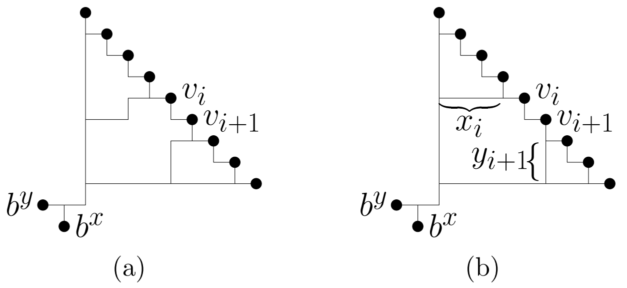

Figure 5.3: Proof of Lemma 5.1 (second case).

(v_n, b^x)-path and a shortest (v_i, v_{i+1})-path for each $i \in \{1, \ldots, n\}$. In other words, each minimum Manhattan network contains a staircase boundary. First assume for all $i \in \{1, \ldots, n\}$ the shortest path between v_i and the cross point run either all to the left boundary segment or all to the bottom boundary segment. W. l. o. g. assume all paths run to the bottom boundary segment. See Figure 5.2 (a). If a point $v_k \in \{v_1, \ldots, v_n\}$ which is connected to the bottom boundary segment is not connected to the boundary by the shortest path y_k then we can shorten the network by replacing this connection by y_k. See Figure 5.2 (b). For the leftmost point v_1 it holds that x_1 is in MMN (being of length 0). Furthermore, y_2 is also in MMN by the preceding considerations. Thus, the lemma holds for $i = 1$ in this special case.

Next, assume there exists an $i \in \{1, \ldots, n-1\}$ such that v_i is connected to the left boundary segment and v_{i+1} to the bottom. See Figure 5.3 (a). By the same argument as above, we can shorten the network by using x_i and y_i to connect v_i and v_{i+1} to the left and bottom boundary segment, respectively. See Figure 5.3 (b). □

Using Lemma 5.1 we can specify a dynamic program computing a minimum Manhattan network. See Algorithm 6 Dynamic Program For Staircases for the algorithm. If the path from v_1 or v_n to the cross point is not a straight line segment, we define an artificial sequence point at the end of the line incident to the cross point (w. l. o. g. assume that the vertical boundary segment incident to the cross point ends at height at least the one of v_1 and the horizontal boundary

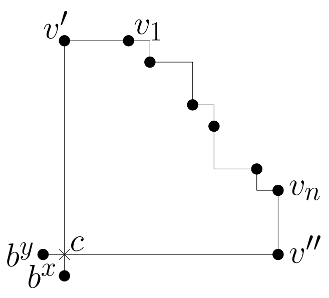

Figure 5.4: The points v' and v'' are the artificial sequence points.

segment incident to the cross point at width at least the one of v_n). See Figure 5.4.

To get a minimum Manhattan network for a staircase with sequence $(v_1, \dots, v_n)$, the algorithm tries all points $\{v_2, \dots, v_{n-1}\}$ as the split point v_i distinguished by Lemma 5.1 (i. e., the index for which a minimum Manhattan network contains the segments x_i and y_{i+1}). This is done in step 6 of Algorithm 6. For this the algorithm uses the length of the beforehand computed minimum Manhattan networks of the staircases with sequences $(v_1, \dots, v_i)$ and $(v_{i+1}, \dots, v_n)$. By inserting x_i and y_{i+1} we get two new staircases with sequences $(v_1, \dots, v_i)$ and $(v_{i+1}, \dots, v_n)$, respectively. The `for`-loop of step 3 considers the magnitude of the staircase, i. e., the number of sequence points. By the `for`-loop of step 4 all staircases with i sequence points are considered. The split point is fixed by the `for`-loop of step 6.

Algorithm 6 Dynamic Program For Staircases

```
Require: A staircase with sequence $(v_1, \dots, v_n)$, $n \geq 3$ and a staircase boundary.
 1: for all $i = 1, \dots n-1$ do
 2:     Set $m(i, i+1) = 0$.
 3: for all $i = 3, \dots, n$ do
 4:     for all $j = 1, \dots, n-(i-1)$ do
 5:         Set $m(j, j+(i-1)) = \infty$.
 6:         for all $k = j+1, \dots, j+(i-1)-1$ do
 7:             if $m(j, j+(i-1)) > m(j,k) + m(k+1, j+(i-1)) + |x_k| + |y_{k+1}|$ then
 8:                 Set $m(j, j+(i-1)) = m(j,k) + m(k+1, j+(i-1)) + |x_k| + |y_{k+1}|$.
 9:                 Set $sp(j, j+(i-1)) = k$.
10: return $Min\_SC(1, n)$
```

Theorem 5.2. *Algorithm 6* Dynamic Program For Staircases *computes a minimum Manhattan network for a staircase and a given boundary.*

Procedure 7 $Min\_SC(1, n)$

```
1: if $n > 1$ then
2:     $k = sp(1, n)$ return $Min\_SC(1, k) \cup Min\_SC(k + 1, n) \cup \{x_k\} \cup \{y_{k+1}\}$
3: else
4:     return $\emptyset$
```

Proof. The variable $m(i, j)$ denotes the the minimum total weight of a Manhattan network for the staircase with sequence $(v_i, \ldots, v_j)$ and boundary segments y_i and x_j. The algorithm computes these variable by recursion:

$$m(j, j + (i - 1)) = \min_{k \in \{j+1,\ldots,j+i-2\}} \{m(j, k) + m(k + 1, j + (i - 1)) \\ +|x_k| + |y_{k+1}|\}.$$

That is, at the vertices v_k and v_{k+1} the line segments x_k and y_{k+1} are inserted, respectively. By the recursion formula each $k \in \{j + 1, \ldots, j + i - 1\}$ is tried and the one which delivers the minimum cost is chosen. By Lemma 5.1 we know that a minimum Manhattan network contains the same chosen segment.

In the first step the straight line segments from v_1 and v_n to the cross points are in the staircase boundary. Inductively, if we call in the recursion $m(j, k)$, the boundary segment x_k is inserted by the recursion formula and y_j exists by induction hypothesis. The same holds for $m(k + 1, j + (i - 1))$, where y_{k+1} is inserted by the recursion formular and $x_{j+(i-1)}$ exists by induction hypothesis.

For each staircase with sequence $(v_i, \ldots, v_j)$ the variable $sp(i, j)$ denotes the split point at which the staircase is split into two partial staircases. The outer `for`-loops in step 3 and 4 computes all staircases with $i = 3, \ldots, n$ consecutive sequence points. Thus, the algorithm enumerates all possible subsets of $(1, \ldots, n)$ and the cost of the two related minimum Manhattan networks which together establish a minimum Manhattan network for the staircase with sequence $(v_1, \ldots, v_n)$. In Procedure 7 the best splitting of a staircase is identified recursively. □

Since the algorithm contains three nested `for`-loops of range $O(n)$ we get the following running time.

Theorem 5.3. *The running time of Algorithm 6* DYNAMIC PROGRAM FOR STAIRCASES *for a staircase with n sequence points is $O(n^3)$.*

5.2 A 2-Approximation for Staircases

In this section we present an approximation of minimum Manhattan networks for staircases. The algorithm partitions recursively a staircase into two new staircases. This proceeding is also known as *thickest-first* partitioning and was analyzed to yield a 2-approximation. Since both the definitions of staircases and the boundary of staircases are not standardized we specify the algorithm adapted to our conditions.

As Algorithm 6 DYNAMIC PROGRAM FOR STAIRCASES our algorithm partitions the staircase into two staircases for which Manhattan networks are computed recursively. In the partitioning step two new line segments are inserted, each of them being a new boundary segment of one of the two new staircases. We get as input a staircase boundary of a staircase with sequence $(v_1, \dots, v_n)$ and base points b^x. We consider the segments x_i and y_i, $1 \leq i \leq n$ to connect a point v_i to the left and bottom boundary segment, respectively. We select the point v_i, $1 \leq i < n$, for which $|x_i| \leq |y_i|$ and $|x_{i+1}| \geq |y_{i+1}|$ holds and add the segments x_i and y_{i+1} to our network and compute recursively Manhattan networks for the staircases with sequence $(v_1, \dots, v_i)$ and the one with sequence $(v_{i+1}, \dots, v_n)$. See Algorithm 8 RECURSION FOR STAIRCASES for a detailed description.

Algorithm 8 RECURSION FOR STAIRCASES

```
Require: A staircase with sequence $(v_1, \dots, v_n)$, $n \geq 3$ and a staircase boundary $B$.
1: Set $SC = \emptyset$.
2: Let $v_i, v_{i+1}$ be the unique pair of neighbors with $|x_i| \leq |y_i|$ and $|x_{i+1}| \geq |y_{i+1}|$.
3: Set $SC = SC \cup \{x_i\} \cup \{y_{i+1}\}$.
4: Let $SC'$ = RECURSION FOR STAIRCASES($(v_1, \dots, v_i), SC \cup B$).
5: Let $SC''$ = RECURSION FOR STAIRCASES($(v_{i+1}, \dots, v_n), SC \cup B$).
6: Set $SC = SC \cup SC' \cup SC''$.
7: return $SC$.
```

Let $(v_1, \dots, v_n)$ be a staircase sequence with base points b^x and b^y. Let $\mathcal{A}_{app}$ be the staircase area of the staircase boundary given to Algorithm 8 RECURSION FOR STAIRCASES for the instance. A minimum Manhattan network contains shortest (b^x, v_1)- and (b^y, v_n)-paths and shortest (v_i, v_{i+1})-paths, $1 \leq i < n$, inside the neighboring point area $\mathcal{N}$ constituting a staircase boundary B_{opt} inside $\mathcal{N}$. Let $\mathcal{A}_{opt}$ the staircase area of B_{opt}. In the following we require that $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ holds. Let MMN be the whole Manhattan network for the staircase. In Chapter 6 a main task is to design algorithms satisfying this requirement.

Theorem 5.4. *Algorithm 8* RECURSION FOR STAIRCASES *computes a Manhattan network $SC \cup B$ for a staircase satisfying $SC \leq 2 \cdot |MMN \cap \mathcal{A}_{app}|$ if $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ holds.*

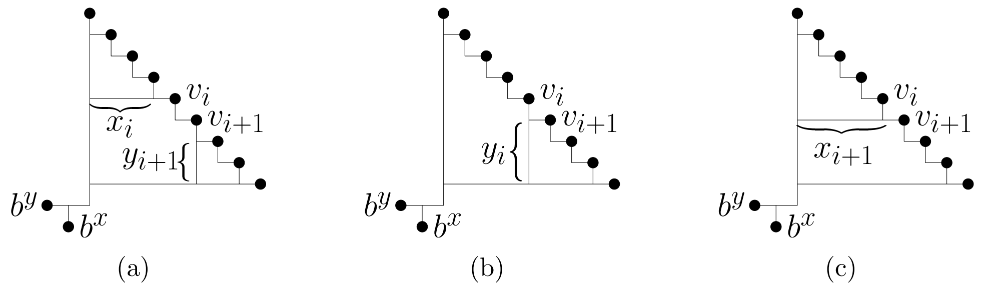

Figure 5.5: Proof of Theorem 5.4.

Proof. Due to the fact that we recursively call the algorithm and in each call, we connect two points of the sequence to the base points, we get a Manhattan network for the staircase.

We prove the ratio between the length of the solution of Algorithm 8 and the length of a minimum Manhattan network to be two by the use of an inductive argument over the number of sequence points. Assume we insert in the i-th step a segment x_k and a segment y_{k+1}. Let x_i^{opt} and y_i^{opt}, $1 \leq i \leq n$, the segments x_i and y_i for B_{opt} but only considered inside $\mathcal{A}_{app}$. We get $|x_i| = |x_i^{opt}|$ and $|y_i| = |y_i^{opt}|$. If we insert x_i, it holds $|x_i| \leq |y_i|$ and therefore $|x_i| \leq min\,\{|x_i^{opt}|, |y_i^{opt}|\}$. In an analogous manner it holds $|y_{i+1}| \leq min\,\{|x_{i+1}^{opt}|, |y_{i+1}^{opt}|\}$ if we insert y_{i+1}. To connect v_i and v_{i+1} to the cross point a minimum Manhattan network needs at least the length of $min\,\{|x_i^{opt}|+|y_{i+1}^{opt}|, |y_i^{opt}|, |x_{i+1}^{opt}|\}$ inside $\mathcal{A}_{app}$. See Figure 5.5 for an illustration of the three possible alternatives a minimum Manhattan network has to connect v_i and v_{i+1} to the boundary. If the minimum is adopted for $|x_i^{opt}| + |y_{i+1}^{opt}|$ then our algorithm takes the right choice. Otherwise we get the following estimation:

$$\begin{aligned}
|x_i| + |y_{i+1}| &\leq min\,\{|x_i^{opt}|, |y_i^{opt}|\} + min\,\{|x_{i+1}^{opt}|, |y_{i+1}^{opt}|\} \\
&\leq min\,\{|x_{i+1}^{opt}|, |y_i^{opt}|\} + min\,\{|x_{i+1}^{opt}|, |y_i^{opt}|\} \\
&\leq 2\,min\,\{|y_i^{opt}|, |x_{i+1}^{opt}|\}.
\end{aligned}$$

So in step k we insert at most twice the required length inside $\mathcal{A}_{app}$ of the minimum Manhattan network to connect v_i and v_{i+1} to the base points. Furthermore, we split the staircase in two disjoint staircase with smaller number of sequence points. According to the induction hypotheses for these two staircases the approximation ratio holds. □

Theorem 5.5. *The running time of Algorithm 8* Recursion For Staircases *for a staircase with n sequence points is $O(n \log n)$.*

Proof. The only time consuming work to be done is step 2 which can be executed by binary search in time $O(\log n)$. The recursion have to be proceeded $O(n)$ times. Together we get the running time of the algorithm to be $O(n \log n)$. □

Instead of binary search to identify the unique pair v_i, v_{i+1} in step 2 one can use exponential search combined with binary search as described in [Wid05] and get:

Corollary 5.6. *The running time of the Algorithm 8* Recursion For Staircases *for a staircase with n sequence points is $O(n)$.*

5.3 Conclusion

In this chapter we presented two algorithms computing Manhattan networks for staircases. For this, we assumed that a staircase boundary is given (otherwise we would choose the one with smallest staircase area). First, we introduced an algorithm computing a minimum Manhattan network for a staircase. This is done by a dynamic program based on the observation that we can split the problem into the problem to compute minimum Manhattan networks for two staircases with fewer sequence points. This leads to an algorithm with running time $O(n^3)$ for n staircase points.

Additionally, we introduced a 2-approximation algorithm for this problem again by recursively splitting the problem into two subproblems. At this time we do not search for the optimal split point but select a point such that the length to connect this and the next consecutive sequence point to the boundary is at most twice the length of the segments of a minimum Manhattan network to connect these points to the boundary. With this approach we achieve a better running time of $O(n \log n)$.

Similar approaches were discussed by other authors but since both the definition of staircases and the properties used in Chapter 6 are different, we specified the algorithms adapted to our conditions.

Chapter 6

Approximation Algorithms for Manhattan Networks

In this chapter we present three new approximation algorithms for the Manhattan network problem. Each algorithm realizes Algorithm 4 MANHATTAN NETWORK presented in Section 4.4. In Section 6.3 we introduce a combinatorial 3-approximation algorithm for the Manhattan network problem with running time $O(n \log n)$. Both our algorithm and in particular its analysis are much simpler than the prior 3-approximation algorithm of Benkert et al. [BWWS06] which has a technically quite involved analysis. Our result uses a 2-approximation of minimum Manhattan networks of staircases which we introduced in Section 5.2. Afterwards, we state two 2-approximation algorithms, the first in Section 6.4 with running time $O(n^3)$ and the second in Section 6.5 with running time $O(n \log n)$. Both algorithms use Algorithm 5 COMPUTE BOUNDARIES to fix staircase boundaries. The first algorithm uses minimum Manhattan networks for staircases which we can compute in time $O(n^3)$ as we illustrated in Section 5.1. The second algorithm uses again the 2-approximation algorithm for Manhattan networks of staircases. We will explain differences and similarities of the algorithms in Section 6.2. Last, in Section 6.6 we give an idea how we could possibly improve the approximation ratio.

As already mentioned earlier, up to now it is not known whether the minimum Manhattan network problem is NP-hard. The algorithm with the best approximation ratio published so far is a 2-approximation algorithm presented by Chepoi et al. [CNV08] and is based on LP-rounding. Their LP consists of $O(n^3)$ variables and constraints. Kato et al. [KIA02] proposed a 2-approximation algorithm with running time $O(n^3)$, however the proof of the correctness seems to be incomplete [BWWS06]. Seibert and Unger [SU05] presented an approximation algorithm and claimed that it yields a 1.5-approximation. As remarked by Chepoi et al. [CNV08] both the description of the algorithm and the performance guarantee are incomplete and not fully understandable. In the next section we show by a counterexample that an important intermediate step is incorrect. Thus, our 2-approximation algorithm presented in Section 6.5 achieves the best known approximation ratio and running time so far.

6.1 The Algorithm of Seibert and Unger

In this section we point out that the proof of the approximation ratio of the algorithm presented by Seibert and Unger [SU05] is not correct. For this we name two different points of their analysis, at which it is incorrect. Their algorithm computes in a first phase two sets of vertical and horizontal line segments by two plane sweeps. The sweeps fix line segments for neighboring points such that for two neighboring points p and q there is at most one line segment of $\partial R(p,q)$ in the respective dimension fixed. A minimum Manhattan network also contains the lengths of these line segments, thus the two sets together contain segments of total length at most the length of a minimum Manhattan network. They choose the cheaper of the two sets to use at most half the length of the optimal solution. They claim that this chosen set separates all staircases of each other and that they can independently compute Manhattan networks for the staircases. Since the set of line segments computed in the first phase does not contain a staircase boundary for each staircase, they also have to insert remaining line segments of the boundaries. They claim that this can be done independently of the other staircases (and staircase boundaries) and that this costs all in all at most the length of a minimum Manhattan network. This is the first crux in their argumentation because the boundaries are not independently. They compute for each staircase a minimum Manhattan network containing also a boundary for the staircase with a dynamic program. For a staircase sequence $(v_1, \ldots, v_k)$ of points lying top-right which have to be connected to a point top-left of them, they select a help point $z_{1,k} = (v_{1_x}, v_{k_y})$ and compute a Manhattan network for $(v_1, \ldots, v_k)$ and $z_{1,k}$. In this step they do not consider computed boundaries of other staircases. They argue that solving all sub-problems of this type costs all in all not more than the optimal solution. But this is not necessarily true and it is not clear how to fix this issue. See Figure 6.1 and 6.2 for an example consisting of two staircases. The length of the vertical and horizontal line segments inserted in the first phase is equal. w. l. o. g. assume the algorithm chooses the set of vertical line segments. See Figure 6.1 (a). Afterwards, the algorithm of Seibert and Unger computes for the two staircases independently minimum Manhattan networks including the boundary for each staircase (Figure 6.1 (b) and (c)). The resulting network consists of these two computed Manhattan networks for the staircases (Figure 6.2 (a)). See Figure 6.2 (c) for a minimum Manhattan network for the instance. If we delete the line segments computed in the first phase, Seibert and Unger claim that the remaining line segments have length at most the one of a minimum Manhattan network. See Figure 6.2 (b). We see that the claim does not

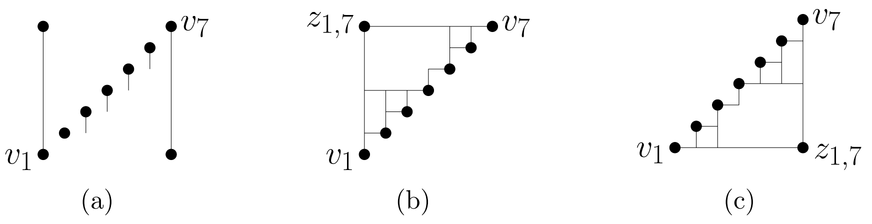

Figure 6.1: (a) The set of vertical line segments computed in the first phase of the algorithm of Seibert and Unger. (b) and (c) The independently computed solutions two staircases.

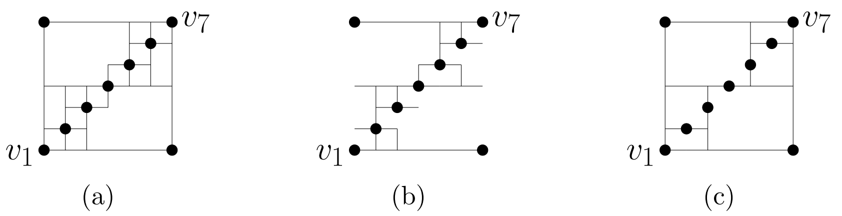

Figure 6.2: (a) The composed solution. (b) The composed solution without the segments of the first phase. (c) A minimum Manhattan network for the same point set.

hold. There are at least too many horizontal edges. More precisely, the critical rectangles defined by the inner points of the staircase sequence contains two times the length of the length of line segments of a minimum Manhattan network inside these rectangles. If we compress the example to shorten the vertical line segments, the length of the line segments depicted in Figure 6.2 (b) exceeds the length of the minimum Manhattan network (Figure 6.2 (c)). Thus, they cannot compute the minimum Manahattan networks of staircases independently if they separate the staircases only by horizontal *or* vertical line segments computed in the first phase. The total length of all minimum Manhattan networks exceeds the length of a minimum Manhattan network, in contradiction to their claim.

If one considers to fix this problem by inserting only one connection between two points if they belong to two staircases, one can see that the choice of the connection affects the solution of the involved staircases. See Figure 6.3 for an example of nested staircases.
There are two staircases, the first with sequence $(p_1, \ldots, p_7)$ and base point b and the second with sequence $(q_1, \ldots, q_5)$ and base point p_4. The connection between p_3 and p_4 and between p_4 and p_5 affects the the second staircase. To use for all staircases together at most the length of a minimum Manhattan network we have to enumerate all alternatives for each such connection. Since this nesting of staircases can be iterated, generally this selection takes exponential time.

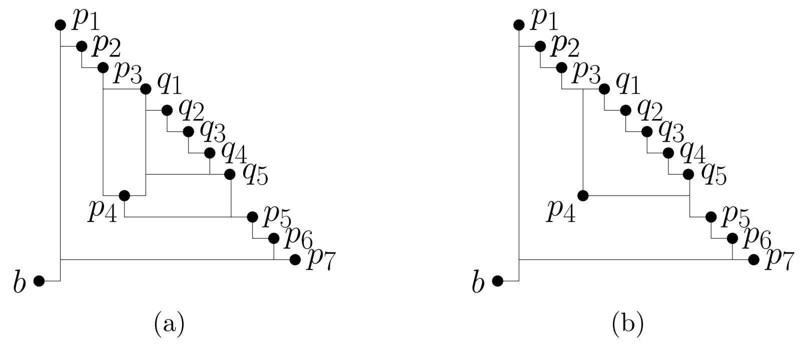

Figure 6.3: Nested staircases.

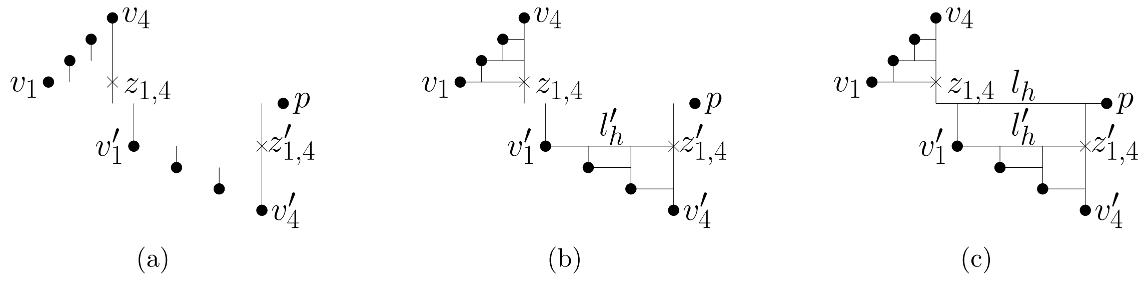

Figure 6.4: (a) The vertical line segment computed by the algorithm of Seibert and Unger. (b) The minimum Manhattan networks for the staircases. (c) The remaining connections.

After computing Manhattan networks for staircases, they have to insert further line segments to establish remaining connections. They claim that this can be done without consuming (together with the previously computed Manhattan networks for staircases) more than the length of a minimum Manhattan network. See Figure 6.4 for an example. Assume the set of vertical line segments is cheaper than the horizontal ones (see Figure 6.4 (a)). They first compute Manhattan networks for the staircase with sequence $(v_1, \ldots, v_4)$ and help point $z_{1,4}$ and for the sequence $(v'_1, \ldots, v'_4)$ and help point $z'_{1,4}$ independently. For the latter point set the Manhattan network contains also the line segment l'_h (see Figure 6.4 (b)). Afterwards, they have to connect $z_{1,4}$ to the point p by inserting the horizontal line segment l_h (see Figure 6.4 (c)). A minimum Manhattan network for this instance contains only the line segment l_h and not l'_h in contradiction to their claim.

We are not aware how to fix the problems both of the algorithm and the analysis of it. We thin that it is necessary to use both the vertical and horizontal line segments (computed in the first phase) to separate the staircases.

6.2 Overview of the Algorithms

In this section we give an overview of the algorithms presented in this chapter and point out differences and similarities.

All presented algorithms implement Algorithm 4 MANHATTAN NETWORK of Section 4.4. The differences between the approximation algorithms for Manhattan networks we present in this chapter lie mainly in the choice of the staircase boundaries computed in the first phase of the algorithm. The choice of the staircase boundaries is the crucial point to get an optimal solution for a Manhattan network problem. See Figure 6.3 for an instance with two differently chosen staircase boundaries. In this example there are two staircases, first the one with sequence $(p_1, \ldots, p_7)$ and base point b and second the one with sequence $(q_1, \ldots, q_5)$ and base point p_4. We see that the choice of the boundary for the first staircase affects the staircase area of the second staircase. In Figure 6.3 (b) we see that the two boundaries can share segments. If we would try all possibilities this leads to an exponential algorithm. To get a polynomial time algorithm we have to guess a boundary. This is done in different ways for the different algorithms. However, all selected boundaries lie in the neighboring point area $\mathcal{N}$ defined by the rectangles of x- or y-neighboring points.

Generally, we use two different strategies for our algorithms. The first is to partition the plane into two disjoint regions and to determine the approximation ratio separately in each region. To illustrate this in more detail, let MN_{app} be the output of one of our algorithms following this strategy. The network MN_{app} contains inside $\mathcal{N}$ a staircase boundary for each staircase. A minimum Manhattan network MMN also contains for each staircase inside $\mathcal{N}$ a staircase boundary. For a staircase let $\mathcal{A}_{opt}$ be the staircase area of the boundary of MMN inside $\mathcal{N}$ and let $\mathcal{A}_{app}$ be the staircase area of the boundary chosen by our algorithm. If $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ holds, by Theorem 5.4 we can apply Algorithm 8 RECURSION FOR STAIRCASES to get a 2-approximation of the Manhattan network of this staircase. Furthermore, if $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ holds, for the length of the network Min_SC computed by Algorithm 6 DYNAMIC PROGRAM FOR STAIRCASES we get $|Min\_SC| \leq |MMN \cap \mathcal{A}_{app}|$.

Let $k_1 = 2$, when we apply Algorithm 8 RECURSION FOR STAIRCASES and let $k_1 = 1$, when we use Algorithm 6 DYNAMIC PROGRAM FOR STAIRCASES. Assume that our algorithm inserts inside $\mathcal{N}$ at most k_2 times the length of $MMN \cap \mathcal{N}$ establishing a staircase boundary for each staircase and satisfying $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$. Then we can bound the overall approximation ratio of our algorithm by

$$\begin{aligned}|MN_{app}| &= |MN_{app} \cap (\mathbb{R}^2 \setminus \mathcal{N})| + |MN_{app} \cap \mathcal{N}| \\ &\leq k_1 \cdot |MMN \cap (\mathbb{R}^2 \setminus \mathcal{N})| + k_2 \cdot |MMN \cap \mathcal{N}| \\ &\leq \max\{k_1, k_2\} \cdot |MMN|.\end{aligned}$$

This strategy is used by Algorithm 9 Best Boundary For Manhattan presented in Section 6.3 for $k_1 = 2$ and $k_2 = 3$ and by Algorithm 11 Optimal Boundaries for $k_1 = k_2 = 2$ in Section 6.5.

The second strategy is to compute in each of the two phases line segments of length at most k times the length of a minimum Manhattan network to achieve an overall approximation ratio of $2k$. This is done by Algorithm 10 Sufficient Boundary For Manhattan presented in Section 6.4 for $k = 1$.

To work out the difference between the two strategies we want to point out that the second strategy maybe chooses a non-optimal boundary. That is, we may have chosen the staircase boundaries such that the staircase area of a minimum Manhattan network is smaller than our staircase areas. Then the Manhattan network inside the staircase area of our boundary has length exceeding the length of the Manhattan network inside the staircase area of the optimal boundary. Thus, the argument that a Manhattan network for a staircase in the interior of the staircase area has length at most k times the length of a Manhattan network inside the interior of the staircase area can only be used by the first strategy. The idea of the second strategy is to partition in the first phase the problem only into disjoint regions (i. e., staircase areas) such that we can compute in the second phase Manhattan networks of staircases independently of the other staircases. Theorem 4.23 states that this proceeding leads to a Manhattan network. In each of the two phases we use at most k times the length of a minimum Manhattan network.

Now we illustrate in more detail the different algorithms. The 3-approximation makes sure that the staircase boundaries of a minimum Manhattan network are contained in the identified boundaries. To achieve this we consider neighboring point pairs. Assume two points $p, q \in P$ are x-neighboring. We select the two vertical boundary segments of the rectangle $R(p, q)$. We know that there exists a minimum Manhattan network that contains a y-connection of length $|p_y - q_y|$ lying inside the segments selected by our algorithm. Unfortunately, we do not know exactly which parts of the two boundary segments are contained in a minimum Manhattan network. On this account we choose both segments. In almost the same matter for two y-neighboring points $p, q \in P$ we select the horizontal boundary segments of $R(p, q)$. We will prove that the extracted segments contain a staircase boundary for each staircase. Furthermore, since we choose both segments we know that the staircase area for each staircase has size equal to or

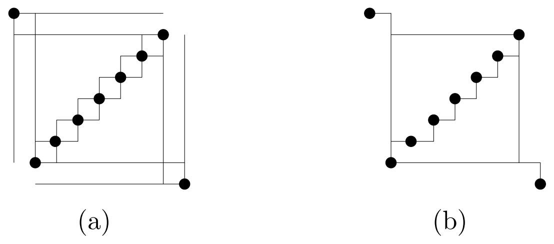

Figure 6.5: (a) Staircase boundaries with smallest areas as computed by the 3-approximation. (b) The boundary computed by Algorithm 5 Compute Boundaries.

smaller than the staircase area of a minimum Manhattan network. Namely, we have to insert fewest segments to get a Manhattan network for the staircase. In Section 6.3 we show that we have to insert at most three times the length of a minimum Manhattan network to get the boundaries with the required properties. More precisely, let MMN be a minimum Manhattan network. We insert boundary segments only inside $\mathcal{N}$. We show that the segments computed in the first phase have length at most $3 \cdot |MMN \cap \mathcal{N}|$. Furthermore, for each staircase, MMN contains a staircase boundary in $\mathcal{N}$. And the assumption $\mathcal{A}_{app} \subseteq \mathcal{A}_{app}$ is fulfilled for the appropriate staircase areas. That is, we can apply Algorithm 8 Recursion For Staircases to achieve a 3-approximation for the entire problem.

To get better approximation ratios we have to take more care when computing the staircase boundaries. For the remaining algorithms we use the approach introduced in Section 4.5. By Theorem 4.35 the overall length of the computed boundaries is at most the length of a minimum Manhattan network inside $\mathcal{N}$. The disadvantage is that the property $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ does not hold. See Figure 6.5. For the network in Figure 6.5 (b) we do not know that $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ holds. Up to now the staircase boundaries partition the plane only into disjoint regions (i. e., the staircase areas) for which we can compute Manhattan networks independently of the other regions.

The 2-approximation algorithm we introduce in Section 6.4 computes in the second phase a minimum Manhattan network for each staircase with a given staircase boundary. Since we partitioned by the extended staircase boundaries the plane into disjoint staircase areas, we can compute a Manhattan network inside an area independently of the networks in the other areas by Theorem 4.23. Therefore, the total length of all optimal solutions of these disjoint problems is at most the length of a minimum Manhattan network for the whole instance. In each of the two phases we inserted segments of length at most the length of a minimum Manhattan network, thus we get a 2-approximation. Since we compute a minimum Manhattan network for each staircase the algorithm has running time $O(n^3)$.

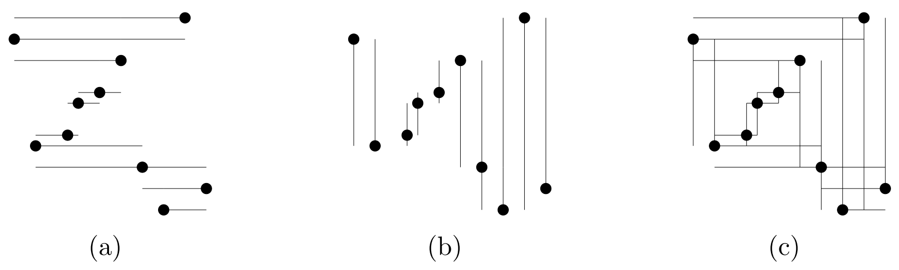

Figure 6.6: Line segments inserted by the sweep steps.

To achieve a better running time, we use the 2-approximation algorithm for staircases for the 2-approximation algorithm presented in Section 6.5. For that we have to guarantee that $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ for the staircase areas holds. This is done by some postprocessing in the first phase. We explain this in more detail in Section 6.5.

6.3 A 3-Approximation of Minimum Manhattan Networks

Our 3-approximation algorithm for minimum Manhattan networks proceeds in two phases. In the first phase we compute a basic set of line segments in each of the two dimensions. These segments ensure that only sequence points for staircases remain unconnected to the appropriate base points and that the segments constitute an extended staircase boundary with smallest area relative to the neighboring point area $\mathcal{N}$. More precisely, we do not need to insert line segments inside $\mathcal{N}$ in Phase II. Furthermore, we do not need line segments of a minimum Manhattan network inside $\mathcal{N}$ to justify segments inserted by Phase II because each staircase area $\mathcal{A}_{app}$ computed in Phase I is contained in a staircase area $\mathcal{A}_{opt}$ inside $\mathcal{N}$ of a minimum Manhattan network (i. e., $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$). In the second phase we compute Manhattan networks for the staircases. We now describe our approach in more detail. We first examine the points from bottom to top. For two y-neighboring points p and q considered by this sweep we insert the horizontal boundary segments of the rectangle $R(p, q)$ into our network. Afterwards we perform an analogous sweep from left to right. Similarly, for two x-neighboring points p and q considered by the sweep we insert the vertical boundary segments of the rectangle $R(p, q)$. See Figure 6.6 (a) to (c) for an example of such a sweep in the two directions. After sorting the points which can be done with running time $O(n \log n)$, the sweeps can be performed in time $O(n)$. After these two sweeps the only remaining parts to get shortest paths between all point pairs are shortest paths between sequence points and their appropriate cross point. Furthermore, the up to now identified line segments contain a boundary for each staircase. Afterwards, we identify all staircases. For this purpose we assign to each point its two base points and then consider the points with the same base points as a staircase. This can be done in time $O(n)$ as stated in Section 4.3. In the last step we compute for each staircase (given a staircase boundary) a Manhattan network. As stated in Section 5.2 we can compute a 2-approximation for a staircase in time $O(n \log n)$ for n input points. Altogether, we achieve a Manhattan network for the input points. See Algorithm 9 for a detailed description.

Algorithm 9 Best Boundary For Manhattan

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
Phase I:
1: Set $CR = \emptyset$.
2: Sweep over the points of $P$ bottom-up. Let $p$ be the currently considered
   point and $q$ be the previously processed point. Add to $CR$ the horizontal line
   segments of $\partial R(p, q)$.
3: Sweep over the points of $P$ from left to right. Let $p$ be the currently considered
   point and $q$ be the previously processed point. Add to $CR$ the vertical line
   segments of $\partial R(p, q)$.
Phase II:
4: Set $MN = \emptyset$.
5: for each Staircase $S$ do
6:     Compute with Algorithm 8 a Manhattan network $2MN$ of $S$ with the
       staircase boundary computed in Phase I.
7:     Set $MN = MN \cup 2MN$.
8: return $MN \cup CR$.
```

To analyze the algorithm we can interpret our strategy in such a way that we partition the plane into regions belonging to two disjoint areas. The first one is the neighboring point area $\mathcal{N}$, the second one the complement $\mathbb{R}^2 \setminus \mathcal{N}$. In the first phase we include line segments in the first area and in the second phase in the second area. For each of these two areas we examine the approximation ratio separately. This strategy for the analysis is similar to the one of Benkert et al. [BWWS06]. They also consider x- or y-neighboring points. Whereas we include for pairs of x-neighboring points p and q both horizontal line segments of $\partial R(p, q)$ into our network (i. e., two y-connections), they first include only one y-connection into $R(p, q)$. A symmetric statement holds for y-neighboring points. They also get a staircase boundary for each staircase. Unfortunately, since they include for two, w. l. o. g. x-neighboring, points p and q only one of y-connection into $R(p, q)$, these segments do not guarantee a partition into two disjoint areas for which they can bound the length of the inserted segments separately. In contrast, we get two disjoint areas since we insert both vertical boundary segments

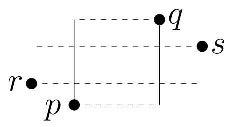

Figure 6.7: Proof of Lemma 6.1.

to maximize the area isolated in the first phase. They have to insert further line segments. This stepwise proceeding complicates the analysis and the algorithm in comparison to our approach.

To prove the correctness of Algorithm 9 we first show that neighboring pairs are connected by a shortest path after Phase I. We then show that Phase I obtains us an extended staircase boundary for each staircase.

Lemma 6.1. *After Phase I of Algorithm 9* Best Boundary For Manhattan, *two x- or y-neighboring points are connected by line segments of CR forming a shortest path.*

Proof. Let p and q be neighboring in x-direction. W. l. o. g. let $p_x \leq q_x$ and $p_y \leq q_y$. If the points are neighboring also in y-direction then CR contains all line segments of $\partial R(p, q)$. Thus, assume p and q are not neighboring in y-direction. There exists a point r with $p_y < r_y < q_y$. W. l. o. g. let $r_x < p_x$ and r be the topmost one. See Figure 6.7. Thus, r is y-neighboring either to q or to another point s in $Q_4(q)$. Since r and s are y-neighboring, the horizontal segments of $\partial R(r, s)$ are inserted into CR. Together with the vertical segments of $\partial R(p, q)$ they constitute a shortest path between p and q. □

After showing the general statement that neighboring points are connected, we now prove that the line segments, added during the sweeps of steps 2 and 3, yield an extended staircase boundary for each staircase. That is, we get paths from the outer points to the base points inside the extended base rectangles (as we prove in Lemma 6.2) and shortest paths between consecutive sequence points (see Lemma 6.3).

Lemma 6.2. *After Phase I of Algorithm 10* Best Boundary For Manhattan, *the outer points of a staircase sequence are connected to both base points by paths in the appropriate extended base rectangles.*

Proof. Let $(v_1, \ldots, v_k)$ be a staircase sequence with base points b^x and b^y. If b^x and v_1 are x-neighboring then they are connected by a shortest path by Lemma 6.1. Thus assume b^x and v_1 are not x-neighboring. Let p be the point x-neighboring to b^x on the right of b^x. See Figure 6.8. Since p is x-neighboring to b^x we know that there exists a shortest (p, b^x)-path. Consider the point q y-neighboring to v_1 above v_1. We distinguish two cases. First assume q lies in $Q_2(v_1)$. (Note that q is the point r^y defined in Definition 4.15 in Section 4.4.)

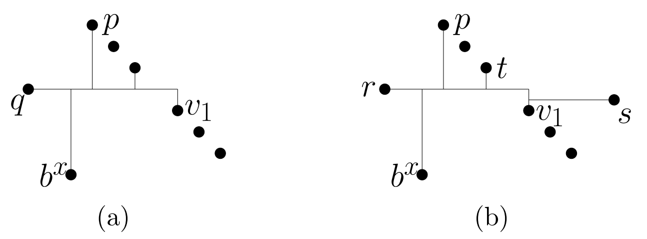

Figure 6.8: Proof of Lemma 6.2.

Since v_1 is the outer point of the staircase and p is x-neighboring to b^x the point q lies on the left of b^x. Again there exists a shortest (v_1, q)-path due to the neighborhood of v_1 and q. Together with the shortest (p, b^x)-path we get a (v_1, b^x)-path in the extended x-base rectangle. See Figure 6.8 (a).

Now assume the point q lies to the right of v_1. Since p does not belong to the staircase sequence $(v_1, \ldots, v_k)$ there is a point above v_1 and to the left of b^x. Let r be the one with smallest y-coordinate. (Note that r is the point r^y defined in Definition 4.15 in Section 4.4.) The point r is y-neighboring below itself to a point s on the right of v_1. There exists a shortest (r, s)-path. Since b^x is not globally x-neighboring to v_1 and v_1 is an outer point the point t x-neighboring to v_1 on the left of v_1 lies above r. Again, there exists a shortest (v_1, t)-path. Together with the shortest (p, b^x)- and (r, s)-path we get a (v_1, b^x)-path inside the extended x-base rectangle. See Figure 6.8 (b).

In almost the same manner we can prove that we get a (v_k, b^y)-path inside the extended y-base rectangle. These two paths together establish also a (v_1, b^y)- and a (v_k, b^x)-path. □

Lemma 6.3. *After Phase I of Algorithm 9* Best Boundary For Manhattan*, two consecutive points v_i and v_{i+1} of a staircase sequence are connected by a shortest path.*

Proof. W. l. o. g. let v_i and v_{i+1} be part of a staircase as depicted in Figure 6.9 and let v_i being on the left of v_{i+1}. Assume v_i and v_{i+1} are not connected after Phase I. Thus, by Lemma 6.1, v_i and v_{i+1} are neither x- nor y-neighboring. Since v_i and v_{i+1} belong to the same staircase and are consecutive in the sequence, the point v^x x-neighboring to v_{i+1} on the left is above v_i. See Figure 6.9. The vertical segments of $\partial R(v^x, v_{i+1})$ are contained in CR. Again, since v_i and v_{i+1} belong to the same staircase and v_i and v_{i+1} are not y-neighboring there exists a y-neighboring point v^y of v_i below and to the right of v_{i+1}. The horizontal segments of $\partial R(v^y, v_i)$ are contained in CR. Thus, by these four line segments v_i and v_{i+1} are connected by a shortest path. □

By Lemma 6.2 and 6.3 we get an extended staircase boundary for each staircase after Phase I.

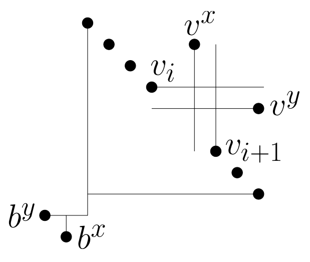

Figure 6.9: (a) Proof of Lemma 6.3.

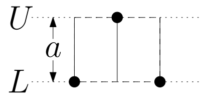

Figure 6.10: Proof of Theorem 6.6.

Lemma 6.4. *After Phase I of Algorithm 10* Best Boundary For Manhattan*, the set CR contains an extended staircase boundary for each staircase.*

Thus, Algorithm 9 is a direct implementation of Algorithm 4 Manhattan Network. By Theorem 4.23 we get that Algorithm 9 computes a Manhattan network.

Theorem 6.5. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane, Algorithm 9* Best Boundary For Manhattan *computes a Manhattan network for P.*

Next, we show the approximation ratio of the algorithm.

Theorem 6.6. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane Algorithm 9* Best Boundary For Manhattan *computes a Manhattan network for P with total length at most three times the length of a minimum Manhattan network for P.*

Proof. To achieve the approximation ratio we have to account each length of a line segment of a minimum Manhattan network at most three times to justify a line segment inserted by Algorithm 9. Our strategy of the proof is to partition the plane into two areas and to compare the length of a minimum Manhattan network in each area separately. First, we want to remind that we only need to consider shortest paths for point pairs constituting critical rectangles. The first area we consider is the neighboring point area $\mathcal{N}$ formed by the union of all critical rectangles of x- or y-neighboring points. Consider the horizontal sweep of the algorithm done in step 3. For each pair of neighboring points p and q we add the vertical segments of $\partial R(p, q)$. In the minimum Manhattan network the length of $|p_y - q_y|$ is required to connect the points. Thus, for this rectangle we insert at most twice the length of the required length. However, we must consider also neighboring critical rectangles having common boundaries as depicted in Figure 6.10. We add three line segments to CR while it suffices to include the middle of the three vertical line segments. Generally for k points alternating on the lines L and U our algorithm adds k line segments of length a while $\lfloor k/2 \rfloor$ suffice. The ratio $k/\lfloor k/2 \rfloor$ is largest for $k = 3$ and achieves the value 3. Thus, we add in Phase I line segments of length at most three times the length of segments of a minimum Manhattan network inside the neighboring point area $\mathcal{N}$.

For any two w. l. o. g. x-neighboring points p and q we know that that there exists a minimum Manhattan network that contains vertical line segments on the boundary of $R(p, q)$ of length $|p_y - q_y|$. We add both vertical boundary segments. Thus, it would not be reasonable to add further vertical segments inside $R(p, q)$. Therefore, we maximize the area separated by the sweep (this is the neighboring point area $\mathcal{N}$) and minimize the remaining area ($\mathbb{R}^2 \setminus \mathcal{N}$). The remaining area is the union of the staircase areas defined by the staircase boundaries given by line segments of Phase I. At this, each staircase area is accounted as an open area. More precisely, let $(v_1, \ldots, v_n)$ be a staircase sequence with base points b^x and b^y and let $\mathcal{A}_{app}$ be an the staircase area of the boundary computed by Algorithm 9. A minimum Manhattan network contains (b^x, v_1)- and (b^y, v_n)-paths and shortest (v_i, v_{i+1})-paths, $1 \leq i < n$, inside the neighboring point area $\mathcal{N}$ constituting an extended staircase boundary B_{opt} inside $\mathcal{N}$. Let $\mathcal{A}_{opt}$ the staircase area of B_{opt}. By the construction of the sweeps we get $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$.

By the property $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ and since $\mathcal{A}_{app} \cap \mathcal{N} = \emptyset$ holds and since we use only line segments of $MMN \cap \mathcal{N}$ to justify segments inserted in Phase I, we partitioned the problem in two areas which can be considered separately. As stated above, for the first area we add at most three times the length of the line segments constituting a minimum Manhattan network in this area. By Theorem 5.4 we can compute a Manhattan network for staircases with approximation ratio two inside the staircase area.

By the calculation done in Section 6.2 with $k_1 = 2$ and $k_2 = 3$, we get that our algorithm approximates minimum Manhattan networks with a ratio of three. □

The core of the proof is that our Phase I yields a 3-approximation outside of staircases and minimizes the staircase areas. That is, we get a partitioning into two disjoint areas (defined by critical rectangles of neighboring points and the staircase areas) which can be examined separately. Thus, together with the 2-approximation for staircases we get a 3-approximation altogether.

Theorem 6.7. *The running time of Algorithm 9* Best Boundary For Manhattan *for n points is $O(n \log n)$.*

Proof. First of all we must sort the points of P horizontally and vertically. This can be done in time $O(n \log n)$. The sweeps in steps 2 and 3 then can be performed in time $O(n)$. With Algorithm 3 Finding Staircases we can find all staircases.

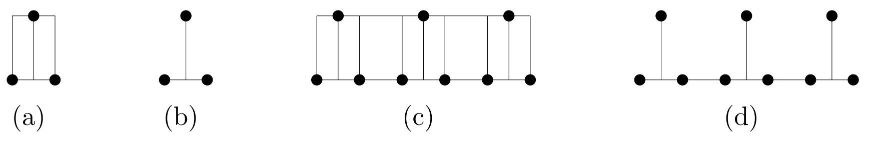

Figure 6.11: (a) and (c) A Manhattan network computed by our approximation. (b) and (d) A minimum Manhattan network.

By Lemma 4.12 this can be done in running time $O(n \log n)$. The running time to compute a 2-approximation for a Manhattan network of a staircase given the staircase boundary is $O(k \log k)$ for k sequence points by Theorem 5.5. Each sequence point can only belong to at most three different staircases with more than two sequence points by Lemma 4.11, thus we get a total running time of $O(n \log n)$ to compute Manhattan networks for all staircases.

This yields the total running time for the algorithm of $O(n \log n)$. □

Actually, combining exponential with binary search we can 2-approximate minimum Manhattan networks of staircases in linear time by Corollary 5.6. Thus, by the analysis for the running time it is obvious that our algorithm has linear running time, except for sorting in x- and y-direction.

Corollary 6.8. *The running time of Algorithm 9* Best Boundary For Manhattan *for n sorted points is $O(n)$.*

See Figure 6.11 (a) and (b) for an example that the approximation ratio of our algorithm is tight. We use three vertical segments instead of one. If we consider to fix this by the observation that we do not need the outer line segments we see by the example given in Figure 6.11 (c) and (d) that redundant line segments can also lie in the interior of the instance.

6.4 A 2-Approximation of Minimum Manhattan Networks

Our 2-approximation algorithm is a direct realization of Algorithm 4 Manhattan Network. First we compute with Algorithm 5 Compute Boundaries a boundary for each staircase. Afterwards we determine the staircases of the instance (this can be done with Algorithm 3 Finding Staircases) and compute for each staircase a minimum Manhattan network with Algorithm 6 Dynamic Program For Staircases. See Algorithm 10 for a detailed description.

We name the algorithm Sufficient Boundary For Manhattan because the lines of the x- and y-covering computed by step 1 guarantee a partitioning of

Algorithm 10 SUFFICIENT BOUNDARY FOR MANHATTAN

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
 Phase I:
 1: Let $MN$ be the return of Algorithm 5 COMPUTE BOUNDARIES.
 Phase II:
 2: for each staircase $S$ do
 3:     Compute with Algorithm 6 a minimum Manhattan network $MMN$ of $S$
        with the staircase boundary computed in Phase I.
 4:     $MN = MN \cup MMN$.
 5: return $MN$.
```

the plane into staircases in such a way that we get for each staircase a staircase boundary so that we can compute Manhattan networks for the staircases independently. If we delete one of these line segments the remaining segments would not suffice.

By Theorem 4.23 we get the following statement:

Theorem 6.9. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane, Algorithm 10* SUFFICIENT BOUNDARY FOR MANHATTAN *computes a Manhattan network for P.*

To achieve the approximation ratio, our algorithm is allowed to use at most twice the length of a minimum Manhattan network. By the following theorem we prove that we use in each of the two phases at most the length of a minimum Manhattan network to achieve an overall approximation ratio of two.

Theorem 6.10. *For a set $P \subseteq \mathbb{R}^2$ of points in the plane, Algorithm 10* SUFFICIENT BOUNDARY FOR MANHATTAN *computes a Manhattan network for P with total length at most twice the length of a minimum Manhattan network for P.*

Proof. By Theorem 4.35 the length of the line segments computed in Phase I is at most the length of a minimum Manhattan network. By Lemma 4.19 and Theorem 4.21 the Manhattan networks of the different staircases are independent of each other. We compute for each staircase a minimum Manhattan network in Phase II. No line segment computed in Phase II for a staircase can contribute to a Manhattan network of another staircase since we have disjoint regions. Thus, the overall length of all these Manhattan networks computed in Phase II is again at most the length of a minimum Manhattan network. Altogether, the total length of the Manhattan network for P is at most twice the length of a minimum one, concluding the proof of the theorem. □

Theorem 6.11. *The running time of Algorithm 10* SUFFICIENT BOUNDARY FOR MANHATTAN *for n points is $O(n^3)$.*

Proof. By Lemma 4.36 the running time of Phase I is $O(n \log n)$. By Lemma 4.12 it requires $O(n \log n)$ time to find the staircases. By Theorem 5.3 the running time to compute a minimum Manhattan network for a staircase with k sequence points is $O(k^3)$. Each point can only belong to at most four different staircase sequences. Therefore, we get the total running time of $O(n^3)$ to compute Manhattan networks for all staircases.

This delivers the total running time for Algorithm 10 of $O(n^3)$. □

6.5 A Fast 2-Approximation of Minimum Manhattan Networks

In this section we present a 2-approximation algorithm with a better running time of $O(n \log n)$. Actually, since we have to sort all points, usually this is the best running time which can be expected for an algorithm solving the Manhattan network problem. Even though all in all the algorithm acts as Algorithm 4 MANHATTAN NETWORK we have to make more effort. To achieve the desired approximation ratio it is important that our staircase areas are at least so small as the staircase areas of a minimum Manhattan network coming from line segments inside rectangles defined by neighboring points. Given this property we regard the plane as partitioned into two areas and compare the length of a minimum Manhattan network in each area separately to achieve the approximation ratio. In each area we use line segments of length at most twice the length of a minimum Manhattan network inside the appropriate area. Altogether, our network has length at most twice the length of a minimum one. We want to point out the difference in this approach to the 2-approximation algorithm introduced in Section 6.4 where we used in each phase of our algorithm at most once the length of a minimum Manhattan network leading to an overall approximation ratio of two as well. For the former approximation algorithm we cannot ensure that we have best possible staircase boundaries relative to neighboring points. Therefore, we have to solve the Manhattan networks for staircases two optimality to get a 2-approximation. Unfortunately, this leads to the worse running time of $O(n^3)$. The algorithm we introduce in this section guarantees that each staircase area $\mathcal{A}_{app}$ computed in Phase I is contained in a staircase area $\mathcal{A}_{opt}$ of boundaries lying inside the neighboring point area $\mathcal{N}$ of a minimum Manhattan network (i. e., $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$). In the following we specify how we obtain the desired property.

6.5.1 The Algorithm

As the algorithm basically runs like Algorithm 4 MANHATTAN NETWORK, we first fix, just as in Algorithm 10 SUFFICIENT BOUNDARY FOR MANHATTAN, with Algorithm 5 COMPUTE BOUNDARIES the boundaries of the staircases. See

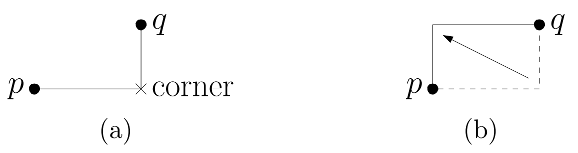

Figure 6.12: An up-switch.

Algorithm 11 Optimal Boundaries for a detailed description of our algorithm. Afterwards, we perform some improvements to the boundaries. For this we want to define an operation called *switch* which moves some line segments of the coverings. For two line segments $l_h \in H$ and $l_v \in V$ let $R(l_h, l_v)$ be the smallest rectangle containing l_h and l_v completely. A *corner* is a point $c \notin P$ where two line segments of H and V end. A corner c has to be incident to exact two line segments. Thus, for two line segments $l_h \in H$ and $l_v \in V$ forming a corner, the segments l_h and l_v both lie on boundary edges of $R(l_h, l_v)$.

Definition 6.12. *Let H be an x-covering and V be a y-covering of a set $P \subseteq \mathbb{R}^2$ of points in the plane. Let $l_h \in H$ and $l_v \in V$ be two line segments forming a corner. A* switch *of l_h and l_v is an operation moving the line segments to the opposite sides of the rectangle $R(l_h, l_v)$. An* up-switch *is performed if l_h lies at the bottom of $R(l_h, l_v)$ before the switch-operation, otherwise we call it* down-switch.

See Figure 6.12 for an illustration of the defined terms. All points connected before a switch-operation by a shortest path are still connected afterwards. The length of a network is not increased by a switch-operation.

All line segments of a minimum Manhattan network and also all segments picked by our algorithm lie in the smallest rectangle containing all input points. Furthermore, we can limit the points to a smaller region. This region was first considered in the context of Manhattan networks by Chepoi et al. [CNV08] and is called *pareto envelope*. Given a set $P \subseteq \mathbb{R}^2$ of points in the plane, a point $q \in \mathbb{R}^2$ is called *efficient* point of P [CFK81, CNV08, WHL71] if there does not exist another point $r \in \mathbb{R}^2$ such that $d(r, p) \leq d(q, p)$ for each $p \in P$ and $d(r, p') < d(q, p')$ for at least one $p' \in P$. The *pareto envelope* of P is the set of all efficient points. The pareto envelope can be computed in time $O(n \log n)$ [CFK81]. The pareto envelope $\mathcal{P}$ can also be characterized by $\mathcal{P} = \cap_{p \in P} \cup_{q \in P} R(p, q)$. Procedure 12 Make Envelope called in step 2 moves the segments found by Algorithm 5 Compute Boundaries to circumscribe an area as small as possible (i. e., the pareto envelope). (For our approach it is not necessary to prove formally that the area we generate by calling Procedure 12 is the pareto envelope.) See Figure 6.13 (c) and (d) for an example of the set of line segments in MN before and after calling Procedure 12 Make Envelope.

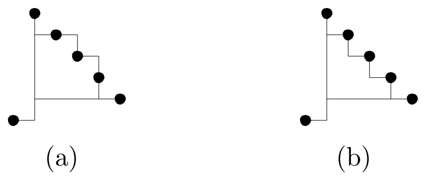

Figure 6.13: (a) The set MN before calling Procedure 12 Make Envelope. (b) The set MN after calling Procedure 12 Make Envelope.

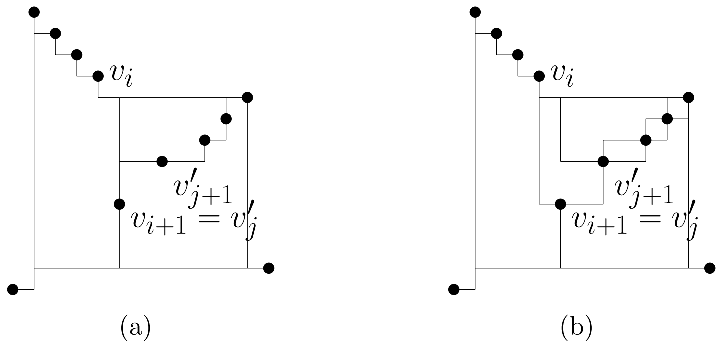

Figure 6.14: (a) The set MN before calling Procedure 13 Make Smallest Sequence. (b) The set MN after calling Procedure 13 Make Smallest Sequence.

Now we attain the main difference to the 2-approximation presented in Section 6.4. By the time the only remaining connections to get a Manhattan network are paths inside staircase areas. In other words we already chose an extended staircase boundary for each staircase. Unfortunately, our choice is maybe not optimal. We always use as little staircase boundaries and thus get as large staircase areas as possible. The Manhattan network in the interior of the staircase area has length exceeding the length of a minimum Manhattan network in the interior of the staircase area of the optimal boundary. It may well pay off to use more boundary segments in order to decrease the staircase areas. See Figure 4.19 for an example. If we diminish the staircase areas, the lengths of the networks inside the staircase areas will decrease as well. Note that until now we only examined x- or y-neighboring points (i. e., we only added segments inside the neighboring point area $\mathcal{N}$). We do not know the optimal staircase boundaries, nevertheless we now want to make the staircase areas as small as possible with line segments inside $\mathcal{N}$. On this behalf we examine the staircases and their boundaries.

We differentiate whether the considered line segments are segments between neighboring sequence points or between the base points and the outer points. The first segments are examined by Procedure 13 Make Smallest Sequence called in step 3 of Algorithm 10. Assume that two neighboring staircase sequence

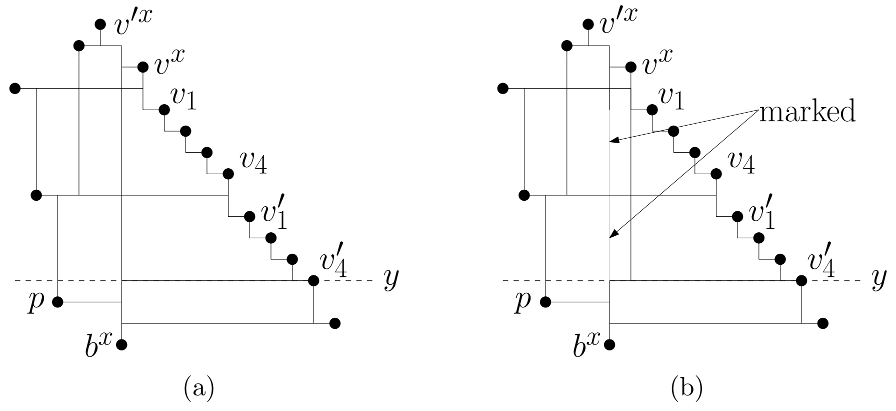

Figure 6.15: (a) The set MN before calling Procedure 14 Make Smallest Base. (b) The set MN after calling Procedure 14 Make Smallest Base.

points v_i and v_{i+1} are x-neighboring. We do not know which shortest path between neighboring sequence points is chosen by a minimum Manhattan network. For this reason we insert the missing line segments of $\partial R(v_i, v_{i+1})$ if they diminish the staircase area (step 2 of Procedure 13). See Figure 6.14. Given line segments solely inside rectangles of x- or y-neighboring points, the boundary between sequence points is chosen in such a way that the staircase area is smallest possible. Nevertheless, we have to take care that we do not select too much line segments. Therefore, we keep in mind the already considered and inserted segments (step 5 and 8 of Procedure 13) and delete superfluous line segments (step 4 and 7 of Procedure 13). In the example in Figure 6.14, we insert for both, the rectangle $R(v_i, v_{i+1})$ and the rectangle $R(v'_j, v'_{j+1})$ the vertical boundary segments. Assume the algorithm considers first the rectangle $R(v_i, v_{i+1})$. The algorithm marks the two vertical line segments of $R(v_i, v_{i+1})$. If the algorithm considers the rectangle $R(v'_j, v'_{j+1})$ it inserts the remaining vertical boundary segments of it. and deletes the vertical segment incident to $v_{i+1} = v'_j$ inside $R(v'_j, v'_{j+1})$ (step 4 of Procedure 13). If we would not delete this segment we could not guarantee that we add at most twice the length of line segments of a minimum Manhattan network inside $R(v_i, v_{i+1}) \cup R(v'_j, v'_{j+1})$. We will prove in Lemma 6.14 that we preserve shortest paths by this proceeding.

Now we consider boundary segments between the base and the outer points. For the following look at Figure 6.15 (a). The vertical line segment incident to b^x takes account for the staircase areas with staircase sequences $(v_1, \dots, v_4)$ and $(v'_1, \dots, v'_4)$ being not smallest with respect to the neighboring point area. We look at the x-neighboring point v^x of b^x and add the vertical line segment beginning at v^x which ends at the next horizontal line segment below or at the same height as v'_4. See Figure 6.15 (b). This is done by Procedure 14 Make Smallest Base which is called in step 4 of the algorithm. Procedure 14 considers all x-base points and examine whether there exist staircases for which the boundary is not optimal according to segments inside rectangles of neighboring points. As above we have to take care about superfluous line segments. The procedure marks all segments which are doubled (step 9 of Procedure 14). See Figure 6.15 for the marked segments. If in a later step a marked segment is again doubled (to the other side as before), it will be deleted (step 8 of Procedure 14). After calling Procedure 14, for all staircases the boundary between base points and outer points lie in such a way that the staircase area is smallest possible. Procedure 14 describes the proceeding only for staircase sequences lying on the right of the base point. The same has to be done with the sequences on the left. In an analogous way we have to consider also the y-base points with associated staircase sequences with one point y-neighboring and certainly all other staircase types as depicted in Figure 4.6. These steps are not mentioned in Algorithm 11.

In Phase II of the algorithm we compute a Manhattan network for each staircase given the staircase boundary with Algorithm 8 Recursion For Staircases achieving for each staircase a Manhattan network with length at most twice the length of a minimum Manhattan network for this staircase. Altogether, Algorithm 11 computes a Manhattan network for a set P of points.

Algorithm 11 Optimal Boundaries

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
 Phase I:
1: Let $MN = H \cup V$ be the return of Algorithm 5 Compute Boundaries.
2: Let $MN$ = Make Envelope($P, H, V$).
3: Let $MN$ = Make Smallest Sequence($P, MN$).
4: Let $MN$ = Make Smallest Base($P, MN$).
 Phase II:
5: for each staircase $S$ do
6:     Compute with Algorithm 8 a Manhattan network $2MN$ of $S$ with the
   staircase boundary computed in Phase I.
7:     $MN = MN \cup 2MN$.
8: return $MN$.
```

The benefit of Procedure 13 Make Smallest Sequence and Procedure 14 Make Smallest Base is, if we allow only line segments in the region defined by rectangles of x- or y-neighboring points (the neighboring point area $\mathcal{N}$), for each staircase the staircase area is smallest. More precisely, each staircase area $\mathcal{A}_{app}$ computed in Phase I is contained in a staircase area $\mathcal{A}_{opt}$ defined by a boundary inside the neighboring point area $\mathcal{N}$ of a minimum Manhattan network (i. e., $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$). This is an important point to guarantee our approximation ratio. Furthermore, we will show that after calling the Procedure 14 Make Smallest

Procedure 12 Make Envelope

```
Require: A set $P \subseteq \mathbb{R}^2$ of points and an $x$-covering $H$ and a $y$-covering $V$ of $P$.
1: for each horizontal line segment $l_h \in H$ under which there do not lie any
   other line segments of $H$ do
2:     if $l_h$ forms a corner with a vertical line segment $l_v \in V$ then
3:         Perform an up-switch if possible.
4: for each horizontal line segment $l_h \in H$ above which there do not lie any
   other line segments of $H$ do
5:     if $l_h$ forms a corner with a vertical line segment $l_v \in H$ then
6:         Perform a down-switch if possible.
7: return $H$ and $V$.
```

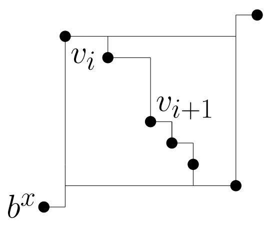

Figure 6.16: Proof of Lemma 6.13.

Base we inserted at most twice the length of all line segments of a minimum Manhattan network inside the neighboring point area $\mathcal{N}$. In the second phase we compute a 2-approximation for each staircase given the staircase boundary. Since we know that no staircase area of a minimum Manhattan network defined by shortest paths between x- or y-neighboring points is smaller than our staircase areas, we also add for all staircases together at most twice the length of segments of a minimum Manhattan network in the staircase areas.

6.5.2 Analysis of the Algorithm

First we prove that no shortest path is deleted by calling Procedure 13 Make Smallest Sequence. For this consider the following lemma which can be easily seen by considering the possible intersections of sequence rectangles.

Lemma 6.13. *If a line segment is deleted in step 4 or 7 of Procedure 13* Make Smallest Sequence*, it was already in MN before calling Procedure 13.*

Proof. Let $R(v_i, v_{i+1})$ be a sequence rectangle which is considered by Procedure 13. W.l.o.g. let v_i and v_{i+1} be x-neighboring with $v_{i_x} \leq v_{i+1_x}$ and base points b^x and b^y lying bottom-left. See Figure 6.16. Procedure 13 inserts in step 2 line segments inducing a smaller staircase area. If in step 4 such a segment l_v would be deleted, it has to be contained in a further sequence rectangle $R(v'_j, v'_{j+1})$ of two x-neighboring points v'_j and v'_{j+1}. One of the two points v'_j

Procedure 13 MAKE SMALLEST SEQUENCE

```
Require: A set $P \subseteq \mathbb{R}^2$ of points and a set $MN$ of vertical and horizontal line segments.
1: for each rectangle $R(v_i, v_{i+1})$ defined by neighboring staircase sequence points $v_i, v_{i+1}$ of a staircase with more than two sequence points where $v_i$ and $v_{i+1}$ are at least either $x$- or $y$-neighboring do
2:     Insert into $MN$ the missing line segments of $\partial R(v_i, v_{i+1})$ which induce a smaller staircase area.
3:     if $v_i$ and $v_{i+1}$ are $x$-neighboring and vertical line segments are inserted by step 2 then
4:         Delete all marked vertical line segments of $\partial R(v_i, v_{i+1})$.
5:         Mark the remaining vertical line segments of $\partial R(v_i, v_{i+1})$.
6:     if $v_i$ and $v_{i+1}$ are $y$-neighboring and horizontal line segments are inserted by step 2 then
7:         Delete all marked horizontal line segments of $\partial R(v_i, v_{i+1})$.
8:         Mark the remaining horizontal line segments of $\partial R(v_i, v_{i+1})$.
9: return $MN$.
```

and v'_{j+1}, say v'_j, has to be the point v_i. First assume v'_{j+1} lies to the left of v'_j. The left x-neighbor of $v'_j = v_i$ is either the x-base point b^x or a point above v_i. In the latter case, $R(v'_j, v'_{j+1}) \cap R(v_i, v_{i+1}) = v_i$ contradicting that l_v is also in $R(v'_j, v'_{j+1})$. Thus, assume the left x-neighbor of v_i is the x-base point b^x and $R(v_i, b^x) = R(v'_j, v'_{j+1})$. (The base points of the sequence rectangle $R(v'_j, v'_{j+1})$ lie top-left.) The rectangle $R(v_i, b^x)$ contains before calling Procedure 13 a y-connection which lies incident to b^x (otherwise Procedure 13 would not have inserted the left vertical boundary segment of $R(v_i, v_{i+1})$). It follows that after considering the rectangle $R(v_i, v_{i+1})$ the rectangle $R(v'_j, v'_{j+1})$ contains on the height of $R(v_i, v_{i+1})$ both vertical boundary segments. Thus, Procedure 13 would not delete the segment incident to v_i. Now assume v'_{j+1} lies to the right of v'_j. That is, v'_{j+1} is the point v_{i+1} and therefore $R(v_i, v_{i+1}) = R(v'_j, v'_{j+1})$. Since each rectangle is considered only once, the segment incident to v_i would not be deleted. □

Now we prove that calling Procedure 13 MAKE SMALLEST SEQUENCE does not delete any shortest path.

Lemma 6.14. *All points of P connected by a shortest path before calling Procedure 13* MAKE SMALLEST SEQUENCE *in step 3 of Algorithm 11* OPTIMAL BOUNDARIES *are still connected after step 3.*

Proof. Consider steps 2, 4 and 7 of Procedure 13 MAKE SMALLEST SEQUENCE. Let l_v be a vertical line segment that is deleted in step 4. If a line segment is deleted and therefore marked before, this implies that it is also part of a rectangle

Procedure 14 Make Smallest Base

```
Require: A set $P \subseteq \mathbb{R}^2$ of points and a set $MN$ of vertical and horizontal line segments.
 1: for each $x$-base point $b^x$ do
 2:     Let $v^x$ be the $x$-neighboring point of $b^x$ on its right.
 3:     if there exists a staircase sequence with more than 2 points and $b^x$ as $x$-base point then
 4:         Let $(v_1, \dots, v_k), k \geq 3$, be the staircase sequence with $v_k$ has the smallest $y$-coordinate among all staircase sequences with $b^x$ as $x$-base point.
 5:         Let $y$ be the $y$-coordinate of the next horizontal line segment of $MN$ below $v_k$ touching the $x$-coordinate $v^x$.
 6:         if $v^x$ is not incident to a vertical line segment covering $[v^x, (v^x_x, y)]$ then
 7:             Add to $MN$ the line segment $[v^x, (v^x_x, y)]$.
 8:             Delete all marked line segments of $[(b^x_x, v^x_y), (b^x_x, y)]$.
 9:             Mark the remaining line segments of $[(b^x_x, v^x_y), (b^x_x, y)]$.
10: return $MN$.
```

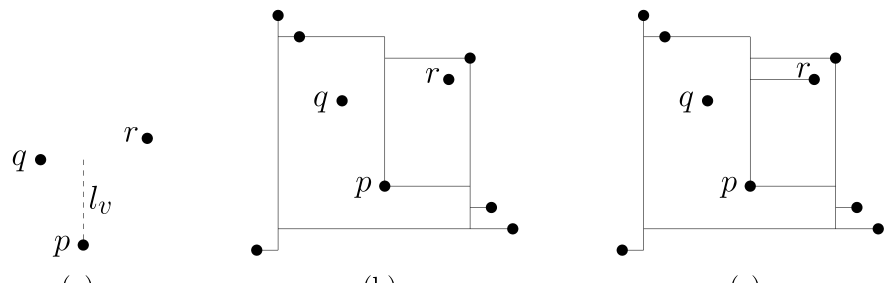

Figure 6.17: Proof of Lemma 6.14.

considered before. The algorithm marks vertical line segments if they are induced by x-neighboring points. Thus, l_v is the intersection of two sequence rectangles $R(p, q)$ and $R(p, r)$ with common point p. That is, the segment l_v is incident to the point p which is x-neighboring to the points q and r either both above or both below p. W. l. o. g. let $q_y \geq r_y \geq p_y$. See Figure 6.17 (a).

The points p and q and the points p and r are neighboring staircase sequence points of two different staircases. The segment l_v is deleted if it would be doubled to the left and to the right. This implies that p and q belong to a staircase with base points bottom-left and p and r to one with base points bottom-right. See Figure 6.17 (b). Thus, p and q are also y-neighboring and the y-connection inside $R(p, q)$ consists of exactly one line segment by Lemma 4.30. The segment l_v is deleted if the line segments $[q, (q_x, p_y)]$ and $[r, (r_x, p_y)]$ are inserted by step 2 and the segment l_v is already in MN before calling Procedure 13. Since p and q have its base points bottom-left and p and r bottom right, by step 2 also the horizontal line segments $[p, (q_x, p_y)]$ and $[p, (r_x, p_y)]$ are inserted if they do not already exist. Since p and r are x-neighboring there is a shortest (p, r)-path in MN after step 1 of Algorithm 11. Assume one of the paths between p and q and between p and r, say the (p, r)-path, is missing after calling Procedure 13. That is, either the vertical line segment $[r, (r_x, p_y)]$ or the horizontal line segment $[p, (r_x, p_y)]$, w. l. o. g. $[p, (r_x, p_y)]$, is also deleted. By Lemma 6.13, $[p, (r_x, p_y)]$ is determined by an x-covering in step 1 of Algorithm 11 and belongs to H. Thus, before calling Procedure 13 in MN are the segments $[p, (p_x, r_y)]$, $[p, (r_x, p_y)]$ and either $[r, (p_x, r_y)]$ or $[r, (r_x, p_y)]$. If $[r, (p_x, r_y)]$ is already in MN after step 1, $[p, (r_x, p_y)]$ would not be deleted by step 7 of Procedure 13. See Figure 6.17 (c). If $[r, (r_x, p_y)]$ is already in MN after step 1, l_v would not be deleted by step 4 of Procedure 13. Therefore, there exists a shortest (p, r)-path after calling Procedure 13. □

Now we examine the Procedure 14 Make Smallest Base and motivate why no shortest path is deleted by calling it. First we show that the procedure is completely passed through only for a very special constellation.

Lemma 6.15. *The if-clause in step 6 of Procedure 14* Make Smallest Base *is fulfilled only if the x-neighboring points of b^x lie both above or both below b^x.*

Proof. Assume v^x is not incident to a line segment covering $[v^x, (v^x_x, y)]$ (y as defined by step 5 of Procedure 14). W. l. o. g. let $v^x_y \geq b^x_y$. Since b^x and v^x are x-neighboring, b^x is incident to a vertical line segment pointing upwards. This line segment is inserted into MN and could be moved or switched to the width of v^x if there would not be an x-neighboring point of b^x on the other side of b^x prohibiting this. Therefore, this point lies also above b^x. □

Now we can prove that Procedure 14 does not delete any possible shortest path between neighboring points.

Lemma 6.16. *All points of P connected by a shortest path before calling Procedure 14* Make Smallest Base *in step 4 of Algorithm 11* Optimal Boundaries *are still connected after step 4.*

Proof. Consider steps 8 and 9 of Procedure 14 Make Smallest Base. W. l. o. g. assume v^x lies above b^x. If a line segment is marked this implies that it is also considered either by Procedure 13 Make Smallest Sequence (i. e., is part of a rectangle defined by neighboring sequence points) or part of another base rectangle already considered before by Procedure 14.

Assume we lose a shortest path. This path can be only be a path between b^x and a point either in the first or second quadrant of b^x. The horizontal line segment with y-coordinate y as defined in step 5 of Procedure 14 which touches the x-coordinate of v^x is justified by two y-neighboring points $p, q \in P$ above or on the

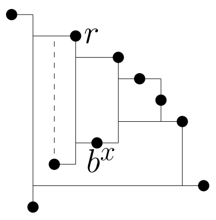

Figure 6.18: Proof of Lemma 6.16.

same height as b^x. One of the two points, say p, lies to the left of b^x and the other to the right (p can be b^x). Before calling Procedure 14 there exists a shortest (b^x, v^x)-path and a shortest (p, q)-path. Since we add in step 7 the missing line segments of $[v^x, (v^x_x, y)]$ together with the shortest (p, q)-path and the parts of the former (b^x, v^x)-paths with y-coordinates smaller or equal y we already keep after step 8 all shortest paths to points in the first quadrant of b^x. See Figure 6.15 for an example.

Now assume we lose a shortest path to a point in the second quadrant of b^x because of the absence of a line segment $[(b^x_x, y_1), (b^x_x, y_2)], y_1 \leq y_2$. The deleted segments of $[(b^x_x, v^x_y), (b^x_x, y)]$ are marked during considering the other point $r \in P$ x-neighboring to b^x which lies above b^x by Lemma 6.15. See Figure 6.18. If the segment is marked in a preceding step of Procedure 14 Make Smallest Base, by the same argument as above we conserve also all shortest paths to points in the second quadrant of b^x.

Thus, assume that b^x and r are neighboring points of a staircase sequence and $[(b^x_x, y_1), (b^x_x, y_2)]$ is marked by step 5 of Procedure 13 Make Smallest Sequence. The appropriate staircase has its base points bottom-left. The line segments $[r, (r_x, b^x_y)]$ and $[b^x, (r_x, b^x_y)]$ are inserted by step 2 of Procedure 13 if they are not already contained in MN.

The line segment $[r, (r_x, b^x_y)]$ is deleted if it es doubled to the left (dashed line segment in Figure 6.18). That is, $[r, (r_x, b^x_y)]$ is in the set V by Lemma 6.13. But then the line segment $[(b^x_x, y_1), (b^x_x, y_2)]$ would not be doubled to the left and therefore not marked when considering the sequence rectangle $R(b^x, r)$. See again Figure 6.18.

Now consider the segment $[b^x, (r_x, b^x_y)]$. Since b^x and r are not y-neighboring (otherwise r would lie below the y-coordinate y), the line segment $[b^x, (r_x, b^x_y)]$ is not marked by step 8 of Procedure 13 Make Smallest Sequence and therefore not deleted when considering $R(b^x, r)$. The line segment $[b^x, (r_x, b^x_y)]$ is deleted only if it is doubled upwards. A doubling upwards is performed if there is a staircase with more than two sequence points and base points top-right. The point b^x has to be one sequence point. But then the left x-neighbor r of b^x lies not above the height y and we do not lose to r a shortest path if we delete vertical segments above y. Thus, we do not delete a shortest path to a point in $Q_2(b^x)$.

Altogether we see that calling Procedure 14 MAKE SMALLEST BASE does not delete any shortest path. □

With this at hand it is easy to see that our algorithm computes a Manhattan network.

Theorem 6.17. *Algorithm 11* OPTIMAL BOUNDARIES *computes a Manhattan network for the input points P.*

Proof. By Corollary 4.34, Lemma 6.14 and Lemma 6.16 after Phase I the set MN contains for each staircase an extended staircase boundary. Thus, steps 1 through 4 perform step 1 of Algorithm 4 MANHATTAN NETWORK. The rest steps of Algorithm 11 are a direct implementation of Phase II of Algorithm 4. Therefore, Algorithm 11 computes a Manhattan network for P. □

To achieve the approximation ratio our algorithm is allowed to use at most twice the length of a minimum Manhattan network. By Theorem 4.35 the length of the segments inserted by step 1 is at most the length of a minimum Manhattan network inside the neighboring point area $\mathcal{N}$. As mentioned earlier the steps performed by Procedure 12 MAKE ENVELOPE does not increase the length of the line segments. In the next theorem we point out that the length of the line segments after calling Procedure 13 MAKE SMALLEST SEQUENCE and Procedure 14 MAKE SMALLEST BASE is at most twice the length of a minimum Manhattan network.

Theorem 6.18. *The total length of the line segments in MN after Phase I of Algorithm 11* OPTIMAL BOUNDARIES *is at most twice the length of a minimum Manhattan network inside the neighboring point area $\mathcal{N}$.*

Proof. Consider Procedure 13 MAKE SMALLEST SEQUENCE. W. l. o. g. assume the actually considered consecutive staircase sequence points v_i and v_{i+1} are x-neighboring. The y-covering contains a y-connection inside the vertical boundary segments of $R(v_i, v_{i+1})$. Each time we insert a vertical line segment by step 2 of Procedure 13, we mark all vertical line segments of $\partial R(v_i, v_{i+1})$. That is, we keep in mind that we doubled the y-covering in $\partial R(v_i, v_{i+1})$. If the algorithm wants to double such a segment a second time for a rectangle $R(v'_j, v'_{j+1})$ (having exactly this line segment of $R(v_i, v_{i+1})$ in common), it will be deleted by step 4. By Lemma 6.13 no segment inserted by Procedure 13 is itself doubled in a later step of Procedure 13. See Figure 6.14. Thus, after step 8 the vertical segments are only doubled and not tripled.

Now, consider the horizontal segment inserted by step 2. If v_i and v_{i+1} are not y-neighboring we cannot consult a horizontal line segment of the x-covering to

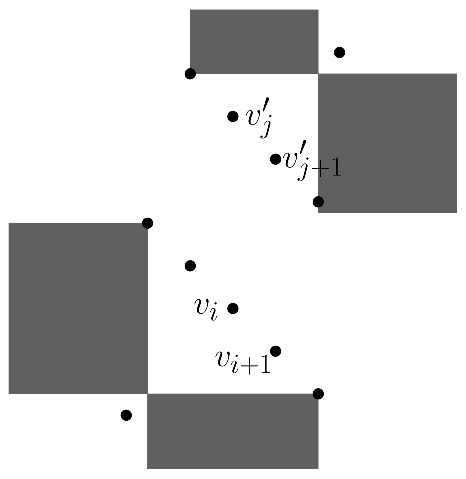

Figure 6.19: Proof of Theorem 6.18.

legitimate it. For this, we use the pareto envelope. In step 2 of Algorithm 11 we moved the line segments of the x- and y-covering to get a smallest area which have to be considered afterwards. No line segment of the pareto envelope has to be doubled by step 2 because all segments of staircase boundaries belonging to the envelope lying in such a way that the staircase area is smallest possible. Thus, if the algorithm inserts a horizontal line segment l_h in step 2 we want to justify it by line segments of the pareto envelope with same x-coordinates. We have to argue that each line segment of the envelope is used at most once. Assume that a part of two line segments l_h and l'_h are justified by the same segment of the envelope. For this, let v_i and v_{i+1} and v'_j and v'_{j+1} be the two pairs of neighboring staircase sequence points which are both not y-neighboring and for which l_h and l'_h are inserted by step 2. W. l. o. g. let $v_{i_x} \leq v'_{j_x}$. In order that one segment of the envelope is used to justify l_h and l'_h it holds $v'_{j_x} < v_{i+1_x}$. Thus, v'_j and v'_{j+1} could be only x-neighboring (as desired by the condition of step 1 of Procedure 13) if $v_{i_x} = v'_{j_x}$ and $v_{i+1_x} = v'_{j+1_x}$. Again by the condition of step 1 of Procedure 13 both sequences have more than two sequence points and since the shaded areas as depicted in Figure 6.19 are empty as mentioned in Section 4.2, the lower staircase has its base points below the sequence points and the upper staircase has them above. One of the two horizontal line segments l_h and l'_h is justified by an envelope segment lying below and the other by the upper envelope segment. Again since the shaded areas are empty there cannot lie above or below these two staircases further staircases for whom also horizontal line segments are added justified by the same envelope segment. Thus, each envelope segment is used to justify only one segment inserted by step 2.

Now we consider Procedure 14 Make Smallest Base. See again Figure 6.15. If the if-clause in step 6 is fulfilled, by Lemma 6.15 b^x is x-neighboring to two points v^x and v'^x above or below it. There is a y-connection in the rectangle $R(v^x, b^x)$ and one in the rectangle $R(v'^x, b^x)$. If v^x is not incident to a vertical line segment covering $[v^x, (v^x_x, y)]$ (y defined as in step 5 of Procedure 14 Make Smallest Base) the y-connection is incident to b^x. The algorithm inserts the missing line segments of $[v^x, (v^x_x, y)]$. If segments of $[(b^x_x, v^x_y), (b^x_x, y))]$ are already marked, this implies that also in the rectangle $R(v'^x, b^x)$ there are line segments inserted at the width of v'^x. That is, the line segments of $[(b^x_x, v^x_y), (b^x_x, y))]$ given by the y-covering is doubled already. If the algorithm wants to double it a second time we delete this segment by step 8. After step 8 MN contains at most twice the length of $[(b^x_x, v^x_y), (b^x_x, y))]$ which was given by step 1 of Algorithm 11. By the construction of the doubling of Procedure 14, for each staircase the doubling is performed to diminish the staircase area. That is, after doubling a segment, on one side of this segment lies the smallest staircase area and the doubled segment could not be doubled into this direction. Thus no doubled line segment will be itself be doubled. If a segment of H or V is doubled twice, it will be deleted. Thus after Phase I the total length of the line segments in MN is at most twice the length of a minimum Manhattan network inside $\mathcal{N}$. □

By now, we are prepared to prove that our algorithm computes a Manhattan network with approximation ratio two.

Theorem 6.19. *Algorithm 11* Optimal Boundaries *computes a Manhattan network for P with total length at most twice the length of a minimum Manhattan network for P.*

Proof. To achieve the approximation ratio we have to account each length of a line segment of a minimum Manhattan network MMN at most twice. Our strategy of the proof is to partition the plane into two areas and to compare the length of a minimum Manhattan network in each area separately. First, we want to remind that we only need to consider shortest paths for point pairs constituting critical rectangles. The first area we consider is the neighboring point area $\mathcal{N}$ defined by x- or y-neighboring points. After Phase I, by Theorem 6.18 the length of the line segments in MN is at most twice the length of a minimum Manhattan network inside $\mathcal{N}$. The remaining area is the area $\mathbb{R}^2 \setminus \mathcal{N}$ which is the union of the interiors of all staircase areas defined by line segments of MN in Phase I plus the area which lies outside the pareto envelope. At this, each staircase area is accounted as an open area. We are allowed to count the line segments added inside the staircase areas independently of the boundary segments only if we know that each staircase area is smallest possible with respect of the neighboring point area $\mathcal{N}$. More precisely, we maximize the area considered by the line segments of the first phase of the algorithm and minimize the staircase areas. A minimum Manhattan network contains for two x-neighboring points at least the length of one y-connection. We do not know where this y-connection lies. To count the segment inserted in the second phase of the algorithm independently we must choose the line segments in Phase I in such a way that they are best for the staircase areas (such that the areas are smallest concerning these segments). On account of this, we call Procedure 13 Make Smallest Sequence (see Figure 6.14 (a) and (b)) and Procedure 14 Make Smallest Base. By Procedure 13 we add for x-neighboring sequence points the vertical boundary segment (together with the horizontal one) of the sequence rectangle which diminishes the staircase area in the best manner. The same holds for y-neighboring sequence points. Now consider Procedure 14. Observe Figure 6.15: Our algorithm chooses the segment incident to b^x, as depicted in Figure 6.15 (a) before calling Procedure 14 Make Smallest Base. A minimum minimum Manhattan network contains either the segments just as our algorithm before calling the procedure or the vertical line segments incident to v^x as depicted in Figure 6.15 (b). Then the staircase area is smaller than the one defined by our line segments in MN after step 2. Therefore we call the Procedure 14 Make Smallest Base which selects the segment incident to v^x. Thus, for a staircase sequence $(v_1, \ldots, v_n)$ with base points b^x and b^y, let $\mathcal{A}_{app}$ be the staircase area of the boundary computed by Algorithm 11 in Phase I. The minimum Manhattan network MMN contains (b^x, v_1)- and (b^y, v_n)-paths and shortest (v_i, v_{i+1})-paths, $1 \leq i < n$, inside the neighboring point area $\mathcal{N}$ constituting a staircase boundary B_{opt} inside $\mathcal{N}$. Let $\mathcal{A}_{opt}$ be the staircase area of B_{opt}. After Phase I we get $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$.

By the property $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$ and since $\mathcal{A}_{app} \cap \mathcal{N} = \emptyset$ holds and since we use only line segments of $MMN \cap \mathcal{N}$ to justify segments inserted in Phase I, we partitioned the problem in two areas which can be considered separately. By Theorem 5.4 we can compute a Manhattan network for staircases with approximation ratio two inside the staircase area.

By the calculation done in Section 6.2 with $k_1 = k_2 = 2$, we get that our algorithm approximates minimum Manhattan networks with a ratio of two. □

Last, we want to prove the running time of our algorithm.

Theorem 6.20. *The running time of Algorithm 11* Optimal Boundaries *for n points is $O(n \log n)$.*

Proof. First of all we must sort the points of P. This can be done in time $O(n \log n)$. By Lemma 4.36 the running time of step 1 is $O(n \log n)$. The sweeps performed in steps 2, 3 and 4 each takes time $O(n)$. By Lemma 4.12 it requires $O(n \log n)$ time to find the staircases. The running time to compute a 2-approximation for a Manhattan network of a staircase given the staircase boundary is $O(k \log k)$ for k sequence points by Theorem 5.5. Each sequence point can only belong to at most three different staircases by Lemma 4.11, thus we get a total running time of $O(n \log n)$ to compute Manhattan networks for all staircases.

This yields the total running time for the algorithm of $O(n \log n)$. □

By Corollary 4.37 and 5.6 for a set of sorted points we get a slightly better running time for the algorithm.

Corollary 6.21. *The running time of Algorithm 11* Optimal Boundaries *for n* sorted *points is $O(n)$.*

6.6 On the Way to Better Approximations of Minimum Manhattan Networks

This section describes an approach for a better approximation ratio than two. Unfortunately, these ideas did not yet lead to a better approximation algorithm. Nevertheless, we get deeper insights into the problem and think that these ideas might lead to a better approximation ratio.

Consider the algorithm we introduced in the preceding section. The structure of the algorithm was to compute with Algorithm 5 Compute Boundaries a set of line segments containing a boundary for each staircase. By Theorem 4.35 this set has length no more than the length of a minimum Manhattan network inside the neighboring point area $\mathcal{N}$. Afterwards, the presented algorithm optimizes the staircase boundaries in the following way: For a staircase each minimum Manhattan network contains a staircase boundary inside the neighboring point area $\mathcal{N}$. Let $\mathcal{A}_{opt}$ be the appropriate staircase area and let $\mathcal{A}_{app}$ the staircase area of the boundary computed by our algorithm for the same staircase. After optimization, the boundary of the staircase area fulfills the property $\mathcal{A}_{app} \subseteq \mathcal{A}_{opt}$. The disadvantage is that we use by this optimization twice the length of a minimum Manhattan network inside $\mathcal{N}$. The idea to achieve a better approximation ratio is to compute a Manhattan network in a similar way to the 2-approximation algorithm presented in Section 6.4 but to use a different strategy to prove the approximation ratio. Particularly, we compute all Manhattan networks for staircases to optimality by Algorithm 6 Dynamic Program For Staircases. If we choose for a staircase a wrong boundary (i. e., the minimum Manhattan network contains a boundary such that the staircase area is smaller), we know that we have a credit since we computed the Manhattan networks for staircases to optimality. See Figure 6.20 for an example. In Figure 6.20 (a) we see the set of lines computed by Algorithm 5 Compute Boundaries. The algorithm chooses the line segment incident to b^x. That is, for the staircases with sequences $(v_1, \ldots, v_4)$, $(v'_1, \ldots, v'_4)$ and $(v''_1, \ldots, v''_4)$, respectively, the staircase area is not smallest possible. It could be that in the neighboring point area $\mathcal{N}$ a minimum Manhattan network contains boundary segments such that the appropriate staircase areas are smaller. Note that the x-neighboring points of b^x are the points v'_1 and v'''_1. The algorithm in the preceding section would diminish the staircase areas by

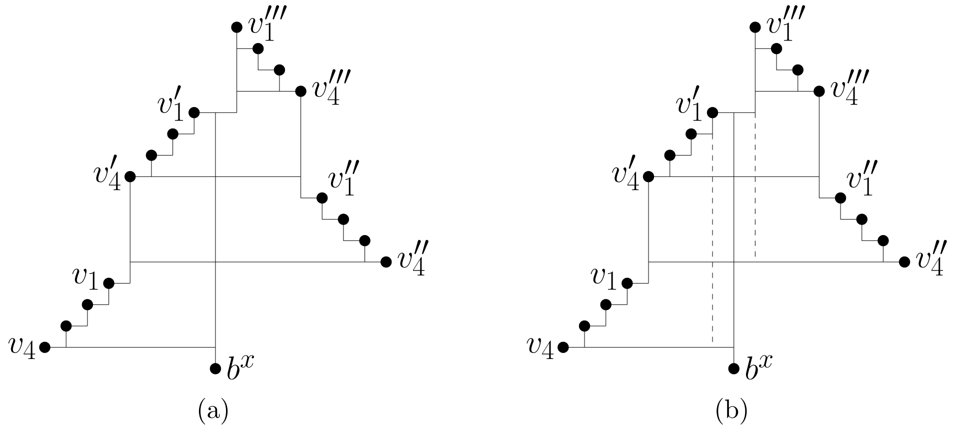

Figure 6.20: Example of a triple.

inserting the dashed line segments showed in Figure 6.20 (b) and deleting the vertical segments on the width of b^x above the height of v_4''. Unfortunately, if a minimum Manhattan network chooses the segment incident to b^x, the algorithm presented in the preceding section would use twice the length of the minimum Manhattan network inside the region $R(v_1', b^x) \cup R(v_1''', b^x)$. Therefore, to achieve a better approximation ratio we must not choose the line segments the algorithm of the preceding section would choose. Furthermore, instead of using Algorithm 8 Recursion For Staircases to solve Manhattan networks for staircases we solve the Manhattan networks for staircases to optimality. Now, consider what could happen. In the worst case, our solution contains the dashed line segments depicted in Figure 6.20 (b). Note that our solution already contains the line segment incident to b^x. Since we solve the staircases to optimality we know that in this case also the minimum Manhattan network would contain the dashed line segments (but not the line segment incident to b^x). Let ℓ_1 be the length of the dashed line segments on the left of b^x below the height of v_4'' and ℓ_2 be the length of the vertical line segments on the width of b^x above v_4''. We can upper bound the length of the vertical line segments in $R(v_1', b^x) \cup R(v_1''', b^x)$ by $2\ell_1 + 3\ell_2$. The length of the vertical line segments of a minimum Manhattan network inside $R(v_1', b^x) \cup R(v_1''', b^x)$ is at least $2(\ell_1 + \ell_2)$. Thus, inside $R(v_1', b^x) \cup R(v_1''', b^x)$ we achieve an approximation ratio of at most $\frac{3}{2}$. This argumentation can be applied if Algorithm 5 Compute Boundaries selects a line segment that is incident to the midpoint of an *alternating triple*.

Definition 6.22. *An* alternating x-triple (p, q, r) *is an ordered set of three points* $p, q, r \in P$ *with* q *x-neighboring to* p *and* r*, and* p *and* r *lie both either above or below* q*.*
An alternating y-triple (p, q, r) *is an ordered set of three points* $p, q, r \in P$ *with* q *y-neighboring to* p *and* r*, and* p *and* r *lie both either to the left or to the right of* q*.*

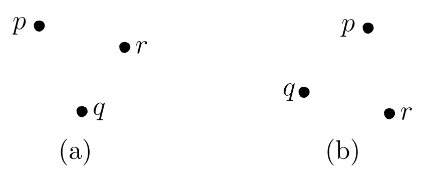

Figure 6.21: (a) An alternating x-triple. (b) An alternating y-triple.

See Figure 6.21 for examples of alternating x- and y-triples.

We can look at this also from another perspective: The line segment incident to b^x is exactly the intersection of the two base rectangles $R(v_1', b^x)$ and $R(v_1''', b^x)$. Generally, if a base or sequence rectangle does not overlap with a base or sequence rectangle of another staircase, we can choose the boundaries right. The problem occurs if two staircases share boundary segments and if the choice of a boundary segment which is optimal is suboptimal for the other affected staircase (as in Figure 6.20). Nevertheless, if all possibly wrong chosen boundary segments are incident to the midpoint of an alternating triple (e. g., the exact intersection of two base (or sequence) rectangles), we would reach a 1.5-approximation. Unfortunately, there exist examples at which the situation is more complicated. We will give them below.

Before, we will improve the boundary computed by Algorithm 5 Compute Boundaries. By the considerations given before, our idea would lead to an approximation ratio not better than 1.5. Consider once again Algorithm 11 Optimal Boundaries presented in the preceding section. As mentioned in the preceding section, we can limit the region to the pareto envelope. That is, after computing the staircase boundaries the algorithm calls Procedure 12 Make Envelope to limit the region where the Manhattan network is placed to the pareto envelope. The pareto envelope contains a subset $\mathcal{P}$ of the input points P. See Procedure 15 to get this set $\mathcal{P}$ of points called *pareto points*. The procedure starts with the bottommost point and runs along the pareto envelope in counterclockwise order and keep in mind the visited points of P. Note that at the end of the proceeding we reach again the point with smallest y-coordinate, which we consider in step 1 of Procedure 15. We keep in mind the order of the points included to $\mathcal{P}$ and call two points *neighboring* in $\mathcal{P}$ if they are included to $\mathcal{P}$ one after the other. The area defined by neighboring points of $\mathcal{P}$ will become essential in the following.

Definition 6.23. *The* pareto point area $\mathcal{N}_{\mathcal{P}}$ *is the union of all rectangles defined by neighboring points of* $\mathcal{P}$*. That is,*

$$\mathcal{N}_{\mathcal{P}} = \bigcup_{p,q \text{ neighboring points in } \mathcal{P}} R(p, q).$$

Procedure 15 Computing the Pareto Points

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
 1: Let $p \in P$ the point with smallest $y$-coordinate.
 2: Set $\mathcal{P} = \{p\}$.
 3: for $i = 1, \dots, 4$ do
 4:     if $i$ is odd then
 5:         Set $d = y$.
 6:     else
 7:         Set $d = x$.
 8:     while there exists a point $q \in P$ in $d$-neighboring to $p$ in $Q_i(p)$ do
 9:         Set $\mathcal{P} = \mathcal{P} \cup q$.
10:         Set $p = q$.
11: return $\mathcal{P}$.
```

To achieve a better approximation ratio we would also use and call Procedure 12 Make Envelope to limit the region to be considered to the pareto envelope. Moreover, our idea is to justify some segments inserted by our algorithm by segments of a minimum Manhattan network that are part of the pareto envelope. For this, consider a point $p \in P$ which is contained in the pareto point set $\mathcal{P}$ and the neighboring point $q \in \mathcal{P}$ concerning the order of $\mathcal{P}$. Inside $R(p, q)$ each minimum Manhattan network contains horizontal line segments of total length $|p_x - q_x|$ and vertical line segments of total length $|p_y - q_y|$. The same holds for the network defined by the line segments computed by Algorithm 5 Compute Boundaries. If for two neighboring points $p, q \in \mathcal{P}$, $R(p, q)$ contains a line segment of a staircase boundary, by Procedure 12 Make Envelope the line segment lies best possible for the staircase. Thus, for two neighboring points $p, q \in \mathcal{P}$ the network contains horizontal line segments of total length $|p_x - q_x|$ and vertical line segments of total length $|p_y - q_y|$ in $R(p, q)$. We use this observation in the next lemma.

Lemma 6.24. *Let MN be the set of segments computed by Algorithm 5* Compute Boundaries *after calling Procedure 12* Make Envelope*. If we select for each pair $p, q \in P$, $p \notin \mathcal{P}$ or $q \notin \mathcal{P}$, of x-neighboring points an additional horizontal line segment of length $|p_x - q_x|$ and for each pair of points $p, q \in P$, $p \notin \mathcal{P}$ or $q \notin \mathcal{P}$, of y-neighboring points an additional vertical line segment of length $|p_y - q_y|$ then the length of MN together with these segments is at most 1.5 times the length of a minimum Manhattan network inside the pareto point area $\mathcal{N}_{\mathcal{P}}$ plus once the length of a minimum Manhattan network inside the remaining parts of the neighboring point area $\mathcal{N} \setminus \mathcal{N}_{\mathcal{P}}$.*

Proof. Let MN be the network computed by Algorithm 5 Compute Boundaries after calling Procedure 12 Make Envelope and MMN be a minimum Manhattan network. By the preliminary considerations we know that

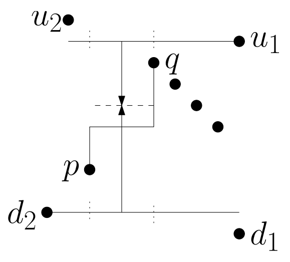

Figure 6.22: Proof of Lemma 6.24.

$|MN \cap \mathcal{N}_{\mathcal{P}}| = |MMN \cap \mathcal{N}_{\mathcal{P}}|$. Now consider two points $p, q \in P$ being x- or y-neighboring. W.l.o.g. let p and q be x-neighboring, $p_x \leq q_x$, and $p_y \leq q_y$. At least one of the two points, say q, is not element of the pareto point set $\mathcal{P}$. That is, both $Q_1(q)$ and $Q_2(q)$ contain at least one point of $\mathcal{P}$. Let $u_1 \in \mathcal{P}$ be the one in $Q_1(q)$ with minimum x-coordinate and $u_2 \in \mathcal{P}$ the one in $Q_2(q)$ with maximum x-coordinate. (Note that $u_{2_x} \leq p_x$ holds.) See Figure 6.22. The points u_1 and u_2 are neighboring points in $\mathcal{P}$ regarding the order of $\mathcal{P}$. In an analogous way, let $d_1, d_2 \in \mathcal{P}$ be the points in $Q_3(q)$ and $Q_4(q)$, respectively. MN as well as the minimum Manhattan network MMN contains horizontal line segments of length $|u_{1_x} - u_{2_x}|$ and $|d_{1_x} - d_{2_x}|$ inside $R(u_1, u_2)$ and $R(d_1, d_2)$, respectively. We use one half the length $|p_x - q_x|$ of $|u_{1_x} - u_{2_x}|$ and one half of the length $|p_x - q_x|$ of $|d_{1_x} - d_{2_x}|$ to justify the additional horizontal line segment inside $R(p, q)$. Since we insert an additional horizontal line segment only between x-neighboring points p and q, in the strip $\{r \in \mathbb{R}^2 \,|\, p_x \leq r_x \leq q_x\}$ at most one additional horizontal line segment is added. Therefore, the segments of the pareto envelope below and above this segment are used only once to justify such a segment. With the same argumentation we can justify vertical line segments for y-neighboring points. Altogether, we can estimate the used length of line segments by

$$|MN| \leq \frac{3}{2}|MMN \cap \mathcal{N}_{\mathcal{P}}| + |MMN \cap \mathcal{N} \setminus \mathcal{N}_{\mathcal{P}}|.$$

□

We use this lemma to insert further line segments without increasing the approximation ratio. See Procedure 16.

Procedure 16 Double Staircases

```
Require: A set $P \subseteq \mathbb{R}^2$ of points and a set $MN$ of vertical and horizontal line
    segments.
1: for each sequence rectangle $R(v_i, v_{i+1})$ that belongs to two staircases each
   with more than two sequence points do
2:     Insert to $MN$ the remaining segments of $\partial R(v_i, v_{i+1})$.
3: return $MN$.
```

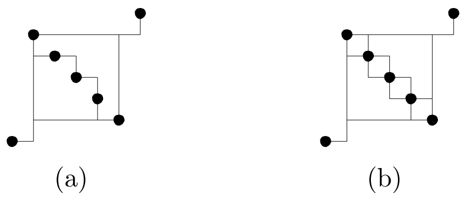

Figure 6.23: (a) Network before calling Procedure 16 Double Staircases. (b) Network after calling Procedure 16 Double Staircases.

With this procedure we can improve our computed boundaries. See Algorithm 17 for the approach.

Algorithm 17 Better Boundaries

```
Require: A set $P \subseteq \mathbb{R}^2$ of points.
 Phase I:
 1: Let $MN = H \cup V$ be the return of Algorithm 5 Compute Boundaries.
 2: Let $MN =$ Make Envelope$(P, H, V)$.
 3: Let $MN =$ Double Staircases$(P, MN)$.
 Phase II:
 4: for each staircase $S$ do
 5:     Compute with Algorithm 6 minimum Manhattan network $MMN$ of $S$.
 6:     $MN = MN \cup MMN$.
 7: return $MN$.
```

See Figure 6.23 for an example of calling Procedure 16 Double Staircases. Since Procedure 12 Make Envelope and Procedure 16 Double Staircases do not delete any shortest path, Algorithm 17 Better Boundaries is a direct implementation of Algorithm 4 Manhattan Network. Thus, by Theorem 4.23 we get that Algorithm 17 computes a Manhattan network.

Theorem 6.25. *For a set $P \subseteq \mathbb{R}^2$ of points Algorithm 17* Better Boundaries *computes a Manhattan network for P.*

Together with Lemma 6.24 we get the following statement.

Lemma 6.26. *The total length of the line segments in MN after Phase I of Algorithm 17* Better Boundaries *is at most* 1.5 *times the length of a minimum Manhattan network inside the pareto point area $\mathcal{N}_{\mathcal{P}}$ and once the length of a minimum Manhattan network inside the remaining parts of the neighboring point area $\mathcal{N} \setminus \mathcal{N}_{\mathcal{P}}$.*

Proof. By Theorem 4.35 after step 1 of Algorithm 17 the length of the line segments in MN is at most the length of a minimum Manhattan network inside $\mathcal{N}$.

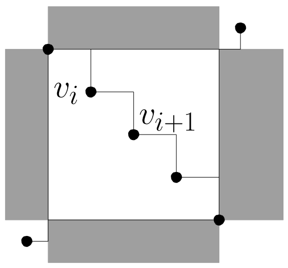

Figure 6.24: Proof of Lemma 6.26.

Calling Procedure 12 Make Envelope does not increase the length of MN. Now, consider Procedure 16 Double Staircases. Let v_i and v_{i+1} be two consecutive sequence points which belong to two staircases. Since $R(v_i, v_{i+1})$ is a sequence rectangle of two staircases, the staircases are either of type depicted in Figure 4.6 (a) and (c) or of type depicted in Figure 4.6 (b) and (d). As mentioned in Section 4.2 the grey shaded areas shown in Figure 6.24 do not contain a point of P. Thus, v_i and v_{i+1} are x- and y-neighboring.

Since the sequences of the appropriate staircases each contain at least three points, one of the two points v_i and v_{i+1} is an inner point. This point does not belong to the pareto envelope. Thus, we can apply Lemma 6.24 and get the desired statement. □

The boundary segments considered by Procedure 16 Double Staircases are exactly these segments lying inside a sequence rectangle which is identical to a sequence rectangle of another rectangle.

To summarize our insights, up to now, we can handle the intersection of base rectangles if they intersect in exactly one line segment and we can handle the intersection of sequence rectangles if they are identical. If two base rectangles overlap in more than one line segment, the staircases have to lie as in Figure 4.30 or the two sequences $(v_1, \ldots, v_4)$ and $(v'_1, \ldots v'_4)$ with x-base point b^x in Figure 6.21. In this case, for two staircases with intersecting base rectangles the upper staircase can be seen as independent of the lower one. Furthermore, the two (or more) affected sequences lie on the same side of the appropriate base point (in contrast to the case where two staircase sequences form an alternating triple). It follows, if we choose for the one staircase the boundary best possible (with respect to the neighboring point area), this does not conflict with the other staircase. The remaining cases which have to be considered are intersections of two different sequence rectangles and the intersection of a base rectangle with a sequence rectangle. We consider the latter and give two examples that need more sophisticated ideas to solve the problem. See Figure 6.25 (a). There are five staircases. The staircase with sequence $(v_1^1, \ldots, v_{k_1}^1)$ has an overlapping base rectangle $R(v_{k_1}^1, v_1'^1)$ with the staircase $(v_1'^1, \ldots, v_{k_1}'^1)$ which contains $R(v_{k_1}^1, v_1'^1)$ as

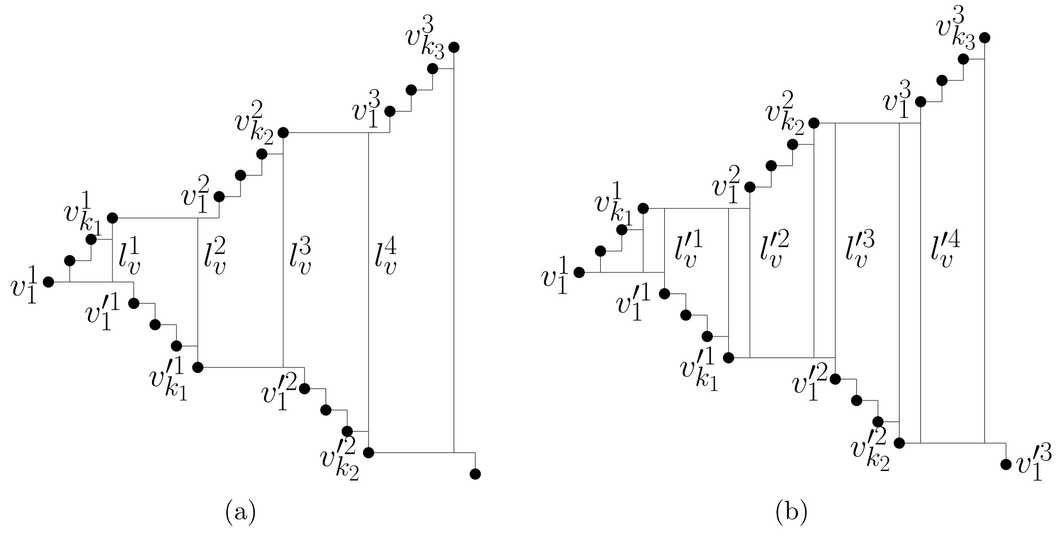

Figure 6.25: (a) Network computed in Phase I of Algorithm 17 Better Boundaries. (b) Possible segments chosen in Phase II.

a sequence rectangle. An analogous statement holds for the other staircases. (Note that we can also construct examples where the overlapping rectangles are not identical.) It could be (with a slightly different scaling of the instance) that our algorithm selects in Phase II the segments depicted in Figure 6.25 (b). The chosen line segments are used to connect the sequence points v'^1_1, v^2_1, v'^2_1 and v^3_1 to its base points v^2_1, v'^2_1, v^3_1 and v'^3_1, respectively. In the appropriate rectangles the network computed by our algorithm contains all vertical boundary segments of the rectangles. In contrast, there is the possibility that a minimum Manhattan network for such a rectangle contains only one of the two vertical boundary segments. This would be the one chosen in our Phase II. Thus, we achieve in these rectangles only an approximation ratio of at most two. To overcome this problem we have to choose the segments more carefully. If we want to try which of the two segments l^1_v and l'^1_v we should choose, we have to know which of the segments l^2_v and l'^2_v we should use. For these segments we have to know the right segment of the two l^3_v and l'^3_v. This iterates until l'^5_v and the example can be continued arbitrarily. There are $\Omega(n)$ staircases linked together.

If we want to try all possibilities to choose the segments $l^1_v, l'^1_v, l^2_v, l'^2_v, \ldots$, this leads to an exponential algorithm. Nevertheless, think of Algorithm 6 Dynamic Program For Staircases for computing minimum Manhattan networks of staircases. We can use (or more precisely extend) the dynamic program to solve this set of nested staircases. We start with the leftmost staircase depicted in Figure 6.25. We compute with the dynamic program the minimum Manhattan network for the staircase with sequence $(v^1_1, \ldots, v^1_{k_1})$ once with the line segment l^1_v and once with the segment l'^1_v and keep in mind the two computed values. In the

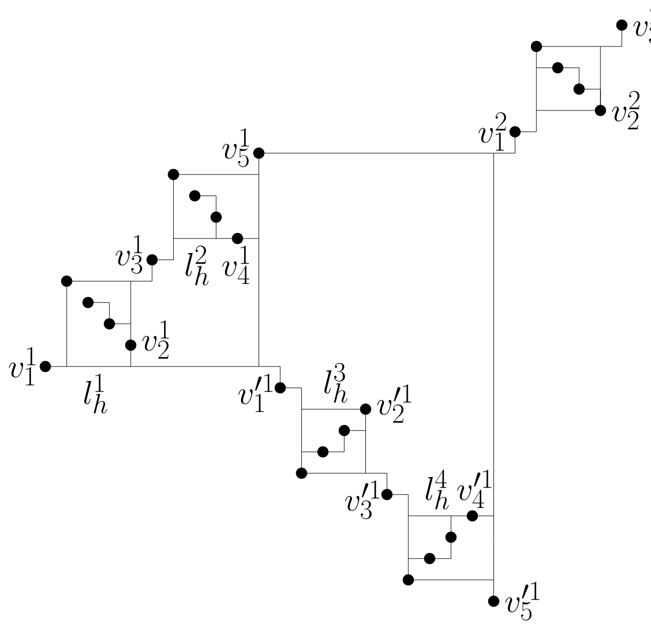

Figure 6.26: An example of a more complicated nesting of staircases.

next step, we consider the next staircase (the one with sequence $(v_1'^1, \ldots, v_{k_1}'^1)$). At this step we compute the minimum Manhattan network four times. We have to try all possibilities to choose l_v^1 or $l_v'^1$ and l_v^2 or $l_v'^2$. To determine the length of the minimum Manhattan network we have to take into account the values of the staircase considered first with respect to which segment l_v^1 or $l_v'^1$ is chosen. For this we do not have to compute the minimum Manhattan network of the first staircase again, but only look at the computed values. Note that we have to compute four times a minimum Manhattan network for the second staircase in this step, but by the computation we could decide for each of the two segments l_v^2 and $l_v'^2$ which segment l_v^1 or $l_v'^1$ should be chosen. That is, we fixed for each of the two segments l_v^2 and $l_v'^2$ the choice of l_v^1 or $l_v'^1$. It follows that for each successive staircase of the nested staircases we have to compute four minimum Manhattan networks of staircases. We could have $O(n)$ nested staircases. Thus, to solve these staircases we get a running time of $O(n^4)$ (the original Algorithm 6 has running time $O(n^3)$).

Unfortunately, the nesting of staircases can be more complicated than in this special case considered beforehand. Moreover, we can extend this example to achieve nestings for almost all sequence rectangles of Figure 6.25. See Figure 6.26. (Note that we draw only the left part of the example given in Figure 6.25.) In this example the sequence rectangles of the staircases depicted in Figure 6.25 overlaps with base rectangles. It could be that our algorithm selects in Phase II the segments depicted in Figure 6.27. In the appropriate rectangles the network computed by our algorithm contains the two horizontal boundary segments $l_h^i, l_h'^i$,

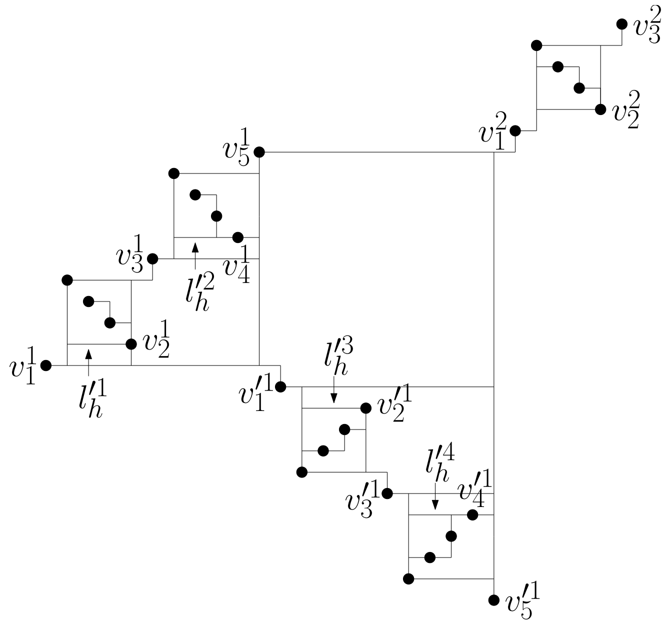

Figure 6.27: Possible segments chosen in Phase I and II.

$1 \leq i \leq 4$, of the rectangles. In contrast, a minimum Manhattan network could contain only one of the two horizontal boundary segments for such a rectangle. Furthermore, the problem which vertical segments should be chosen displayed in Figure 6.25 still exists.

To achieve a dynamic program which solves the problem in polynomial time also for this problem, we have to found a starting point or more precisely, a staircase with which we could start as in the example of Figure 6.25. In Figure 6.25, the leftmost staircase could be seen as the starting point. Generally, we think that a staircase sharing segments with the pareto envelope could be the right choice because the boundary segments contained in the pareto envelope are fixed (since they lie best possible for the staircase and do not affect other staircases). Nevertheless, this ideas have to be made more precise.

Another idea to deal with this problem, is to think of a cost function which assigns a cost to select the problematic line segments inside such a rectangle (for example the two segments l_v^1 and $l_v'^1$ in Figure 6.25) in a more abstract way than to actually compute the complete minimum Manhattan network for the staircase. It would suffice if we could upper bound the length of a minimum Manhattan network for a staircase adequately. This cost function could also be used by a dynamic program.

Indeed, if one of the two ideas presented could be made precise, one can also think to extend this dynamic program to an algorithm solving the Manhattan network problem to optimality.

6.7 Conclusion

In this chapter we introduced four approximation algorithms for the Manhattan network problem. To the best of our knowledge we gave the algorithm with the best approximation ratio of 2 and running time of $O(n \log n)$ for this problem. The algorithm of Seibert and Unger [SU05] who proposed a 1.5-approximation is incorrect in the analysis. We were not able to fix their analysis or to vary their algorithm to achieve the claimed result. We pointed out two different problems arising in their analysis.

Our 3-approximation algorithm is very simple both in the description and in the analysis compared to the algorithm and analysis of the 3-approximation of Benkert et al. [BWWS06]. It is based on two simple sweeps which insert for neighboring points two line segments on the boundary of the rectangle defined by the points. Afterwards, we only have to compute Manhattan networks for staircases. This proceeding can be done in running time $O(n \log n)$ for n input points.

We presented two different 2-approximation algorithms, the first with running time $O(n^3)$ and the second with running time $O(n \log n)$. As the 3-approximation, the first given 2-approximation is rather simple, whereas the second one needs more effort because we needed staircase boundaries fulfilling the assumption that the staircase area is contained in the staircase area of a minimum Manhattan network (defined by boundaries inside the neighboring point area $\mathcal{N}$).

It is a major challenge to clarify the complexity status of the problem. We think the difficulty to solve the Manhattan network problem to optimality lies in assigning the right staircase boundaries. As suggested in Section 6.2 and particularly in Section 6.6 staircases can be "nested". Thus we cannot try all possible staircase boundaries because this might affect the boundaries of all other staircases. Actually, we do not think that this will really happen but a detailed analysis has to be done on how staircase boundaries can interact. As an intermediate step we think that one could improve the ratio of an approximation algorithm to 1.5 as suggested in Section 6.6.

Bibliography

[APD03] E. Althaus, T. Polzin, and S.V. Daneshmand, *Improving linear programming approaches for the Steiner tree problem*, Research Report MPI-I-2003-1-004, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2003.

[Aro98] S. Arora, *Polynomial time approximation schemes for the euclidean traveling salesman and other geometric problems*, Journal of the ACM **45** (1998), 753–782.

[BD95] A. Borchers and D.-Z. Du, *The k-Steiner ratio in graphs*, STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing (New York, NY, USA), ACM Press, 1995, pp. 641–649.

[BDGW98] A. Borchers, D.-Z. Du, B. Gao, and P. Wan, *The k-Steiner ratio in the rectilinear plane*, Journal of Algorithms **29** (1998), 1–17.

[BKMK07a] G. Borradaile, C. Kenyon-Mathieu, and P. N. Klein, *A polynomial-time approximation scheme for Steiner tree in planar graphs*, Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 1285–1294.

[BKMK07b] ———, *Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon*, Proceedings of the 10th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, vol. 4719, Springer-Verlag, 2007, pp. 275–286.

[BP89] M. Bern and P. Plassmann, *The Steiner problem with edge lengths 1 and 2*, Information Processing Letters **32** (1989), 171–176.

[BR94] P. Berman and V. Ramaiyer, *Improved approximations for the Steiner tree problem*, Journal of Algorithms **17** (1994), 381–408.

[Bra01] M. Brazil, *Steiner minimum trees in uniform orientation metrics*, Steiner trees in Industries (D.-Z. Du and X. Cheng, eds.), Kluwer Academic Publishers, 2001, pp. 1–27.

[BTW00] M. Brazil, D.A. Thomas, and P. Winter, *Minimum networks in uniform orientation metrics*, SIAM Journal on Computing **30** (2000), no. 5, 1579–1593.

[BTWZ02] M. Brazil, D.A. Thomas, J.F. Weng, and M. Zachariasen, *Canonical forms and algorithms for Steiner trees in uniform orientation metrics*, Tech. Report TR-02/22, DIKU, Department of Computer Science, Copenhagen, Denmark, 2002.

[BWWS06] M. Benkert, A. Wolff, F. Widmann, and T. Shirabe, *The minimum Manhattan network problem: Approximations and exact solutions*, Computational Geometry: Theory and Applications **35** (2006), no. 3, 188–208.

[CCK$^+$03] H. Chen, C.K. Cheng, A.B. Kahng, I. Măndoiu, and Q. Wang, *Estimation of wirelength reduction for λ-geometry vs. Manhattan placement and routing*, Proceedings of SLIP'03, ACM Press, 2003, pp. 71–76.

[CFK81] G. Chalmet, L. Francis, and A. Kolen, *Finding efficient solutions for rectilinear distance location problems efficiently*, European Journal of Operational Research **6** (1981), 117–124.

[Che89] L. P. Chew, *There are planar graphs almost as good as the complete graph*, Journal of Computer and System Sciences **39** (1989), no. 2, 205–219.

[CKV87] K. Clarkson, S. Kapoor, and P. Vaidya, *Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time*, Proceedings of the 3th Annual ACM Symposium on Computational Geometry (SCG), 1987, pp. 251–257.

[CNV08] V. Chepoi, K. Nouioua, and Y. Vaxès, *A rounding algorithm for approximating minimum Manhattan networks*, Theoretical Computer Science **390** (2008), no. 1, 56–96.

[Cou03] C. Coulston, *Constructing exact octagonal Steiner minimal trees.*, ACM Great Lakes Symposium on VLSI (GLSVLSI), 2003, pp. 1–6.

[CR41] R. Courant and H. Robbins, *What is mathematics?*, Oxford Univ. Press, 1941.

[CW05] C. Chu and Y.-C. Wong, *Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design*, Proceedings of the 2005 international symposium on Physical design, 2005, pp. 28–35.

[dBCvKO08] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed., Springer-Verlag, Berlin, 2008.

[dBGK$^+$05] M. de Berg, J. Gudmundsson, M.J. Katz, C. Levcopoulos, M.H. Overmars, and A.F. van der Stappen, *TSP with neighborhoods of varying size*, Journal of Algorithms **57** (2005), no. 1, 22–36.

[DH92] D.-Z. Du and F.K. Hwang, *Reducing the Steiner problem in a normed space*, SIAM Journal on Computing **21** (1992), no. 6, 1001–1007.

[EFMS05] K.M. Elbassioni, A.V. Fishkin, N.H. Mustafa, and R. Sitters, *Approximation algorithms for euclidean group TSP*, ICALP, 2005, pp. 1115–1126.

[Epp00] D. Eppstein, *Spanning trees and spanners*, Handbook of Computational Geometry (Jörg-Rudiger Sack and Jorge Urrutia, eds.), Elsevier, 2000, pp. 425–461.

[GGJ77] M.R. Garey, R.L. Graham, and D.S. Johnson, *The complexity of computing Steiner minimal trees*, SIAM Journal on Applied Mathematics **32** (1977), 835–859.

[GJ77] M.R. Garey and D.S. Johnson, *The rectilinear Steiner tree problem is NP-complete*, SIAM Journal on Applied Mathematics **32** (1977), 826–834.

[GKR00] N. Garg, G. Konjevod, and R. Ravi, *A polylogarithmic approximation algorithm for the group Steiner tree problem*, Journal of Algorithms **37** (2000), no. 1, 66–84.

[GLN01] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan, *Approximating a minimum Manhattan network*, Nordic Journal of Computing **8** (2001), no. 2, 219–232.

[Han66] M. Hanan, *On Steiner's problem with rectilinear distance*, SIAM Journal on Applied Mathematics **14** (1966), 255–265.

[HAQG02] J. Hu, C.J. Alpert, S.T. Quay, and G. Gandham, *Buffer insertion with adaptive blockage avoidance*, Proceedings of the 2002 international symposium on Physical design, 2002, pp. 92–97.

[HRW92] F.K. Hwang, D.S. Richards, and P. Winter, *The Steiner Tree Problem*, Annals of Discrete Mathematics, vol.53, North-Holland, 1992.

[JK34] V. Jarník and O. Kössler, *O minimálních grafech osahujících n daných bodu*, Ĉas. Pêstování Mat. **63** (1934), 223–235.

[Kar72] R.M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, New York, 1972, pp. 85–104.

[KIA02] R. Kato, K. Imai, and T. Asano, *An improved algorithm for the minimum Manhattan network problem*, Proceedings of the 13th International Symposium on Algorithms and Computations, Lecture Notes in Computer Science, vol. 2518, Springer-Verlag, 2002, pp. 344–356.

[KMZ03] A.B. Kahng, I.I. Măndoiu, and A.Z. Zelikovsky, *Highly scalable algorithms for rectilinear and octilinear Steiner trees*, Proceedings 2003 Asia and South Pacific Design Automation Conference (ASP-DAC), 2003, pp. 827–833.

[Koh95] C.K. Koh, *Steiner problem in octilinear routing model*, 1995.

[KV07] B. Korte and J. Vygen, *Combinatorial optimization: Theory and Algorithms*, 4th ed., Springer-Verlag, Berlin, 2007.

[LPRS82] A. Lingas, R.Y. Pinter, L. Rivest, and A. Shamir, *Minimum edge length partitioning of rectilinear polygons*, Proceedings 20th Allerton Conference on Commun. Control Comput., 1982, pp. 53–63.

[LS96] D.T. Lee and C.-F. Shen, *The Steiner minimal tree problem in the λ-geometry plane*, Proceedings 7th International Symposium on Algorithms and Computations, Lecture Notes in Computer Science, vol. 1178, Springer-Verlag, 1996, pp. 247–255.

[LX00] G.-H. Lin and G. Xue, *Reducing the Steiner problem in four uniform orientations*, Networks **35** (2000), no. 4, 287–301.

[LYW96] D.T. Lee, C.D. Yang, and C.K. Wong, *Rectilinear paths among rectilinear obstacles*, Discrete Applied Mathematics **70** (1996), 185–215.

[Meh88] K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Information Processing Letters **27** (1988), 125–128.

[Mit99] J.S.B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM Journal on Computing **28** (1999), no. 4, 1298–1309.

[Mit00] ______, *Geometric shortest paths and network optimization*, 2000, pp. 633–701.

[MP03] M. Müller-Hannemann and S. Peyer, *Approximation of rectilinear Steiner trees with length restrictions on obstacles*, Proceedings of the 8th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, vol. 2748, Springer-Verlag, 2003, pp. 207–218.

[MS05a] M. Müller-Hannemann and A. Schulze, *Approximation of octilinear Steiner trees constrained by hard and soft obstacles*, Tech. report, University of Darmstadt and University of Cologne, 2005.

[MS05b] ______, *Hardness and approximation of octilinear Steiner trees*, Proceedings of the 16th International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, vol. 3827, Springer-Verlag, 2005, pp. 256–265.

[MS07] ______, *Hardness and approximation of octilinear Steiner trees*, International Journal of Computational Geometry and Applications (IJCGA) **17** (2007), 231–260.

[MT07] M. Müller-Hannemann and S. Tazari, *A near linear time approximation scheme for Steiner tree among obstacles in the plane*, Proceedings of the 10th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, vol. 4619, Springer-Verlag, 2007, pp. 151–162.

[Nou05] K. Nouioua, *Enveloppes de Pareto et Réseaux de Manhattan: Caractérisations et algorithmes*, Ph.D. thesis, Université de la Méditerranée, 2005.

[NS07] G. Narasimhan and M. Smid, *Geometric spanner networks*, Cambridge University Press, New York, NY, USA, 2007.

[NWZ02] B.K. Nielsen, P. Winter, and M. Zachariasen, *An exact algorithm for the uniformly-oriented Steiner tree problem*, 10th Annual European Symposium on Algorithms (ESA 2002), Lecture Notes in Computer Science, vol. 2461, Springer-Verlag, 2002, pp. 760–772.

[PLA02] L. Pachter, F. Lam, and M. Alexandersson, *Picking alignments from (Steiner) trees*, Proceedings of the sixth annual international conference on Computational biology (RECOMB 02), ACM, New York, 2002, pp. 246–253.

[Pro88] J.S. Provan, *An approximation scheme for finding Steiner trees with obstacles*, SIAM Journal on Computing **17** (1988), 920–934.

[PS02] H.J. Prömel and A. Steger, *The Steiner Tree Problem: A Tour through Graphs, Algorithms, and Complexity*, Advanced lectures in mathematics, Vieweg, 2002.

[PWZ04] M. Paluszewski, P. Winter, and M. Zachariasen, *A new paradigm for general architecture routing*, Proceedings of the 14th ACM Great Lakes Symposium on VLSI (GLSVLSI), 2004, pp. 202–207.

[RS98] S.B. Rao and W.D. Smith, *Approximating geometric graphs via "spanners" and "banyans"*, Proceedings of the 30th ACM Symposium on Theory of Computing, 1998, pp. 540–550.

[RZ00] G. Robins and A. Zelikovsky, *Improved Steiner tree approximation in graphs*, Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 770–779.

[Sch05] A. Schulze, *Approximationsverfahren für oktilineare Steinerbäume*, 2005.

[She97] C.F. Shen, *The λ-geometry Steiner minimal tree problem and visualization*, 1997.

[Sla97] P. Slavik, *The errand scheduling problem*, Tech. report, Department of Mathematics, SUNY, Buffalo, USA, 1997.

[SU05] S. Seibert and W. Unger, *A 1.5-approximation of the minimal Manhattan network problem*, Proceedings 16th International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, vol. 3827, Springer-Verlag, 2005, pp. 246–255.

[Tei02] S.L. Teig, *The X architecture: not your father's diagonal wiring*, SLIP '02: Proceedings of the 2002 international workshop on System-level interconnect prediction, ACM Press, 2002, pp. 33–37.

[vdS94] A.F. van der Stappen, *Motion planning amidst fat obstacles*, Phd.d. dissertation, Utrecht University, Nederlands, 1994.

[WHL71] R.E. Wendell, A.P. Hurter, and T.J. Lowe, *Efficient points in location theory*, AIEE Transactions **9** (1971), 314–321.

[Wid05] F. Widmann, *Manhattan-Netzwerke und Stufenpolygone*, Studienarbeit, Universität Karlsruhe, http://i11www.iti.uni-karlsruhe.de /teaching/theses/files/studienarbeit-widmann-05.pdf, 2005.

[WWSCW87] Y.F. Wu, P. Widmayer, M.D.F. Schlag, and C.K. C.K. Wong, *Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles*, IEEE Transactions on Computing (1987), 321–331.

[X] http://www.xinitiative.org.

[ZR03] M. Zachariasen and A. Rohe, *Rectilinear group Steiner trees and applications in VLSI design*, Mathematical Programming **94** (2003), 407–433.

[ZW99] M. Zachariasen and P. Winter, *Obstacle-avoiding Euclidean Steiner trees in the plane: An exact approach*, Workshop on Algorithm Engineering and Experimentation, Lecture Notes in Computer Science, vol. 1619, 1999, pp. 282–295.

[ZZJ$^+$04] Q. Zhu, H. Zhou, T. Jing, X. Hong, and Y. Yang, *Efficient octilinear Steiner tree construction based on spanning graphs*, Proceedings 2004 Asia and South Pacific Design Automation Conference (ASP-DAC), 2004, pp. 687–690.

Index

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe, dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat, dass sie – abgesehen von unten angegebenen Teilpublikationen – noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. R. Schrader betreut worden.

Teilpublikationen:

[1] B. Fuchs und A. Schulze, *A fast 2-Approximation of Manhattan Networks,* in Vorbereitung.

[2] B. Fuchs und A. Schulze, *A simple 3-Approximation of Manhattan Networks,* eingereicht bei Operation Research Letters.

[3] M. Müller-Hannemann und A. Schulze, *Hardness and Approximation of Octilinear Steiner Trees,* International Journal of Computational Geometry and Applications (IJCGA), vol. 17 (2007), pp. 231-260.

[4] M. Müller-Hannemann und A. Schulze, *Approximation of Octilinear Steiner Trees Constrained by Hard and Soft Obstacles,* SWAT 2006, 10th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, Springer 4059, pp. 242-254, Springer, 2006.

[5] M. Müller-Hannemann und A. Schulze, *Hardness and Approximation of Octilinear Steiner Trees,* ISAAC 2005, Proceedings of the 16th Annual International Symposium on Algorithms and Computation 2005, Sanya, Hainan, China, Lecture Notes in Computer Science 3827, pp. 256-265, Springer, 2005.