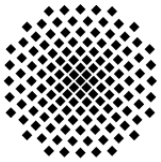

University of Stuttgart
Germany

Institut für Formale Methoden der Informatik

Master Thesis

The Generalized Minimum Manhattan Network Problem

by Michael Schnizler
matriculation number: 2525758

Supervisors:
Prof. Dr. Volker Diekert,
Prof. Dr. Stefan Funke,
Dipl.-Inf. Martin Seybold

May 4th, 2015

Abstract

In this thesis we consider the Generalized Minimum Manhattan Network Problem: given a set containing n pairs of points in $\mathbb{R}^2$ or $\mathbb{R}^d$, the goal is to find a rectilinear network of minimal length which contains a path of minimal length (a so-called Manhattan path) between the two points of each pair. We restrict our search to a discrete subspace and show that under specific conditions an optimal solution can be found in polynomial time using a dynamic program. The conditions concern the intersection graph of the bounding boxes of the pairs. Its maximum degree as well as the treewidth must be bounded by two constants which are independent of n. Finally, we present a simple greedy algorithm for practical purposes.

Kurzfassung

Wir wollen in dieser Arbeit das verallgemeinerte Minimum-Manhattan-Netzwerk-Problem betrachten: Wir suchen, gegeben eine Menge von n Punktepaaren aus $\mathbb{R}^2$ oder $\mathbb{R}^d$, ein achsenparalleles Netzwerk von kleinstmöglicher Gesamtlänge, das für jedes Paar jeweils einen sogenannten Manhattan-Pfad enthält, einen achsenparallelen Pfad minimaler Länge, der die beiden Punkte verbindet. Wir schränken die Suche auf einen diskreten Teilraum ein und zeigen, dass es unter bestimmten Voraussetzungen möglich ist, mittels eines dynamischen Programms in Polynomialzeit eine optimale Lösung zu finden. Die Voraussetzungen betreffen den Schnittgraph der Bounding-Boxen der Punktepaare. Sein Maximalgrad und seine Baumweite müssen durch zwei Konstanten beschränkt sein, die unabhängig von n sind. Zuletzt stellen wir einen einfachen Greedy-Algorithmus für die Anwendung in der Praxis vor.

Contents

1 Introduction

1.1 Motivation

The Generalized Minimum Manhattan Network Problem, this title refers to Manhattan, borough of New York City, where the street map looks like a grid with most of the streets intersecting at a 90° angle. If the grid was complete, you could get from one intersection to another without ever making a detour.

For a given set of points in $\mathbb{R}^2$ (or, more general, in $\mathbb{R}^d$), the goal of MMN (Minimal Manhattan Network) is to find a rectilinear network connecting all of the points in such a way that the distance from one point to another is always minimal.

The generalized version (GMMN) requires only a given subset of the points to be connected via shortest paths. It arises in the context of VLSI circuit design where e.g. certain transistors must be connected in a rectilinear fashion. It is not necessary to connect all of the transistors and logic gates, this is why GMMN describes this application better than MMN does. In general, a circuit designed using GMMN techniqes is significantly shorter than one relying on MMN-based layouts alone. Gudmundsson et al. ([GLN01]) wrote "Many VLSI circuit design applications require that a given set of terminals in the plane must be connected by networks of small total length." and "Manhattan networks are likely to have many applications in geometric network design and in the design of VLSI circuits."

1.2 Background and Related Work

The above quote was one of the motivations when Gudmundsson et al. first introduced the Minimum Manhattan Network Problem in 1999 (we reference their verison from 2001: [GLN01]). They also presented an algorithm with an approximation factor of 4, i.e. the computed network was at most 4 times as long as the minimal network for any set of points. In 2003, Benkert et al. [BWW04] came up with an improved factor of 3, followed by a rounding 2-approximation by Chepoi et al. [CNV08] in 2008. More approximations followed, c.f. [GSZ08a], [GSZ08b] or [FS08]. Das et al. provided an approximation algorithm for MMN in higher dimensions in [DGK$^+$11]. Benkert et al. [BWWS06] were the first to study exact solutions for MMN. The complexity was settled in 2011 by Chin et al. when they proved MMN to be NP-complete, c.f. [CGS11]. Very closely related to MMN is RSA, the

rectilinear Steiner arborescence Problem. For a more detailed view on RSA, we refer the reader to [SS05] and the references there.

Chepoi et al. introduced the generalized version (GMMN) in [CNV08]. There is a $\mathcal{O}(log(n))$-approximation for two and a $\mathcal{O}(\log^{d+1}(n))$-approximation for higher ($d > 2$) dimensions (c.f. [DFK+13] and [DGK+15]). The currently best approximation was given by Funke and Seybold in [FS14]. They achieved an approximation factor of $\mathcal{O}(\mathcal{D})$, where $\mathcal{D} \in \mathcal{O}(\log(n))$, so it matches the $\mathcal{O}(\log(n))$-bound, but performs better on instances with specific properties.

1.3 Contribution

In this thesis, we want to determine under which premises it is possible to find an exact solution for the generalized minimum Manhattan network problem in polynomial time. Chapter 2 will provide some basics as well as terminology and notation. In Chapter 3 we will look at the two-dimensional version of GMMN and identify some instances which are solvable in polynomial time. Particularly we look at intersection graphs generated by GMMN instances, those roughly represent the basic structure of the instances. Using a dynamic program we can find exact solutions for some types of intersection graphs. We try to generalize the results (and lemmas) from Chapter 3 in Chapter 4 so we can solve certain instances of the d-dimensional GMMN problem. The most important proposition will be Theorem 4.4 which yields a polynomial upper bound on the runtime of the generalized dynamic program.

The main result of this thesis will be proven in Chapter 5, Theorem 5.4 states a one-to-one correspondence: provided that the intersection graph's maximum degree is bounded, the premises of Theorem 4.4 are fulfilled if and only if the treewidth of the intersection graph is bounded by some constant.

Finally, in Chapter 6 we present a simple greedy algorithm for practical purposes.

2 Essentials

We assume the reader is familiar with basics of graph theory. Chapter 1 and 4 in [Die12] offer a good introduction. Throughout this thesis, all graphs are undirected, finite and do not have parallel edges or loops.

We introduce some notation for the Minimum Manhattan Network problem :

Definition 2.1

For any point $p = (\xi_1, \ldots, \xi_d)^T \in \mathbb{R}^d$ the L^1 norm is defined as

$$\|p\|_1 = \sum_{i=1}^{d} |\xi_i|.$$

For the L^1 distance or *Manhattan distance* of two points $p, q \in \mathbb{R}^d$ we write

$$d_1(p, q) = \|p - q\|_1.$$

Let $\pi = (p_1, \ldots, p_m)$ be a path in $\mathbb{R}^d$. Then π is called *Manhattan path* or *M-path* for short, if it is axis aligned and its length is the same as the Manhattan distance from p_1 to p_m. That is an axis aligned path of minimum length. Equivalently we could require the sequence $p_1, \ldots, p_m$ to be monotonous in every coordinate.

Let P be a finite set of points in $\mathbb{R}^d$. A graph $G = (V, E)$ is a *Manhattan network* for P, if P is contained in V, every edge is axis aligned and for each pair of points $(p, q) \in P \times P$, G contains an M-path $\pi_{p,q}$ from p to q in E.

It has been shown by Chin et al. [CGS11] in 2011 that MMN is NP-complete. In 2008, Chepoi et al. [CNV08] introduced a generalization, where the input consists of n pairs of points and the problem is to find a network connecting each of the pairs rather than one where all the points are connected to each other. The following notation is now common:

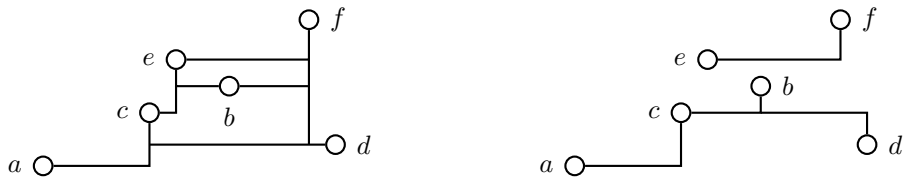

Figure 2.1: example of 2-MMN and 2-GMMN where $P = \{a, \ldots, f\}$

Definition 2.2

An *instance* R is a subset of all possible pairs, i.e. $R \subseteq P \times P$. We denote by $P(R) \subseteq P$ the set of points actually appearing in R and refer to those points as *terminals*.

$G = (V, E)$ is a *generalized Manhattan network* or *GMN* for R, if the vertex set V contains $P(R)$, all the edges are axis aligned and G contains an M-path $\pi_{p,q}$ for each pair $(p, q) \in R$.

$G = (V, E)$ is a *generalized minimum Manhattan network* or *GMMN*, if the overall edge length of G is minimal among all generalized minimum Manhattan networks for R.

For the d-GMMN problem we are interested in finding a GMMN for a given set R of n terminal pairs in $\mathbb{R}^d$. It generalizes the d-MMN problem, where the set R always consists of all pairs of terminals. Note that we consider R to be the input, so the input size n is the number of pairs, not the number of terminals.

Definition 2.3

For $i \in \{1, \ldots, d\}$ let

$$\begin{aligned} \pi_i \ : \ \mathbb{R}^d &\to \mathbb{R} \\ (\xi_1, \ldots, \xi_d)^T &\mapsto \xi_i \end{aligned}$$

be the projection onto the i-th coordinate. Let $\pi_i(R)$ denote the image of $P(R)$ under π_i. Furthermore, let

$$V = \bigotimes_{i=1}^{d} \pi_i(R)$$

and

$$E = \left\{ (v, w) \in V \times V \ : \ \begin{array}{l} \pi_i(v), \pi_i(w) \text{ are neighboring in } \pi_i(R) \text{ for some } i \\ \text{and } p_j(v) = p_j(w) \text{ for all } j \neq i \end{array} \right\}.$$

This graph is called *Hanan grid* for R, we write $\mathcal{H}(R) = (V, E)$ for short.

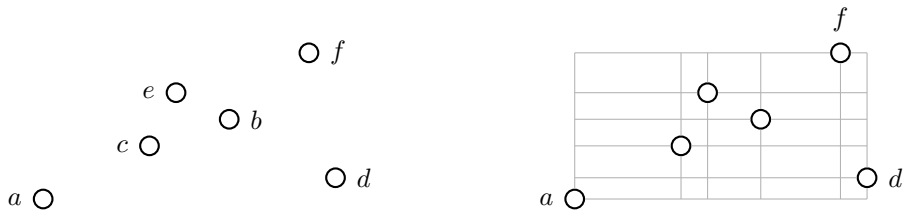

Figure 2.2: example of a Hanan Grid in 2 dimensions

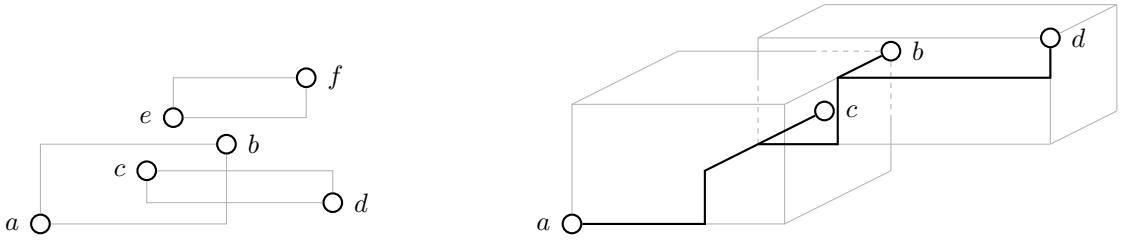

Figure 2.3: example of 2- and 3-dimensional boxes and a 3-GMMN solution

The Hanan Grid is the axis aligned grid which is basically obtained by shooting a line through every terminal in each direction $x_1, \ldots, x_d$. See Figure 2.2 for an example.

Definition 2.4

We denote by *box* the axis aligned bounding box of a pair of terminals (p, q). Now let R be a GMMN instance and

$$V = R,$$
$$E = \{(\text{pair}_1, \text{pair}_2) \in V \times V \ : \ \text{box}(\text{pair}_1) \cap \text{box}(\text{pair}_2) \neq \emptyset\}\,.$$

The graph $G = (V, E)$ is called *intersection graph* for R.

Note that every M-path connecting a pair (p, q) is fully contained in the pair's bounding box. Two Examples of boxes are shown in Figure 2.3. Two M-paths cannot share a segment if their corresponding pairs' boxes intersection is empty. Therefore a set of disjoint boxes yields a lower bound for the length of a minimal network. In case the intersection graph has more than one connected component, we can just split this one (big) instance into multiple (smaller) instances. Their minimal solutions then add up to a minimal solution for the original instance. A disconnected intersection graph is shown in Figure 2.4. This is why we look at intersection graphs: it shows which M-paths might share segments and (just as important) which do not.

For sake of simplicity we will from now on identify a pair of terminals with its bounding box and as well with the correspondig vertex of the intersection graph.

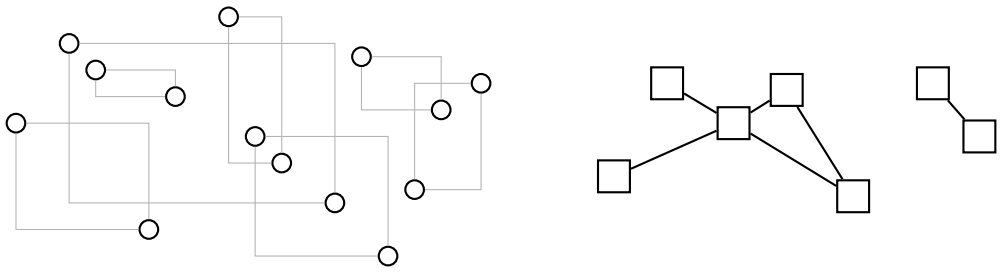

Figure 2.4: instance with boxes and corresponding intersection graph

Lemma 2.5
For any given d-GMMN instance R, the Hanan Grid $\mathcal{H}(R)$ contains no more than $(2n)^d$ vertices and consists of at most $d \cdot 2n$ straight lines. The number of edges is bounded by $d \cdot (2n)^d$.

Proof : Thanks to $|\pi_i(R)| \leq |P(R)| \leq 2n$ the number of straight lines in direction x_i is bounded by $2n$. That leaves us with an overall total of $2dn$ lines. Every vertex lies on exactly d lines, therefore the number of vertices is upper bounded by $(2n)^d$. Considering only one direction x_i, a single vertex has at most 2 neighbors, namely one to the left and one to the right. So the overall number of adjacent edges for that single vertex is $2d$. Adding all these up (thereby counting each edge twice) and dividing by 2 yields the desired upper bound. ■

3 Two-dimensional Case

For now we restrict ourselves to the 2-dimensional GMMN problem. We write x for x_1 and y for x_2, not only for coordinates but also for derived expressions like e.g. $\pi_y(R)$ instead of $\pi_2(R)$. First, we want to narrow down our search space from a possibly uncountable space to a finite subspace. Namely, we restrict ourselves to the Hanan grid.

Lemma 3.1

Let N be a (not necessarily optimal) generalized Manhattan network for a 2-GMMN instance R and denote by ℓ the operator mapping a network to its total length. Then there exists a generalized Manhattan network N' which is completely contained in the Hanan Grid for R and also satisfies

$$\ell(N') \leq \ell(N).$$

Particularly, the existance of an optimal network implies the existance of an optimal network within the Hanan Grid.

The idea of the following proof has been given in [FS14].

Proof : We only give the proof for the x-coordinate, y follows analogously. Again, $\pi_x(R) = \{\xi^{(1)}, \ldots, \xi^{(k)}\}$ denotes the projection of the terminals in R onto their x-coordinate. Let S be the set of vertical segments in N with constant x-value $\xi^* \notin \pi_x(R)$ where ξ^* is minimal. Let a_L and a_R be the number of edges adjacent to S in N on the left and right respectively.

If $a_L = a_R$, we can just shift S to the left until we reach the next $\xi^{(i)}$. The edges adjacent to S are of course shortened or extended (depending on whether they adjoin on the left or on the right). The number of edges shortened is a_L which equals a_R, the number of edges extended by the same length. Therefore, the overall length of the network remains the same.

If $a_L < a_R$, we can shift S to the right until we reach the next $\xi^{(i)}$ or the nearest x-coordinate showing up in N, whichever comes first. As before, we adjust the adjacent edges. Since we shorten more edges than we extend, the overall network length decreases. If $a_L > a_R$, we can shift S to the left accordingly.

Now S is fully contained in $\mathcal{H}(R)$. Iterating this proces will eventually yield a network without vertical edges. It remains to show that it remains a valid Manhattan network.

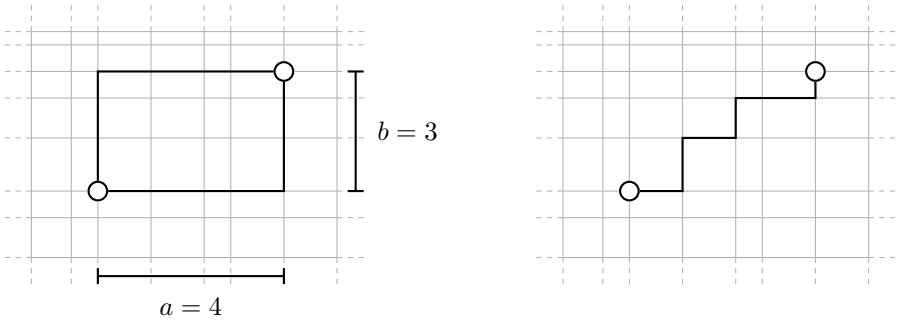

Figure 3.1: left: 'small' box of width $a = 4$ and height $b = 3$ in a large Hanan grid; right: 1 out of the $\binom{7}{3} = 35$ M-paths for that box

Any M-path using a (part of a) segment of S cannot end on S (unless $\xi^* \in \pi_x(R)$ in the first place), so it must run from left to right or vice versa. We added some length to one edge of the M-path and subtracted the same length from some other edge on the path, so in total, the length did not change, so it continues having M-path properties. ■

This restriction suggests a naive brute force algorithm. There are $2^{(8n^2-4n)}$ subsets of edges in the Hanan Grid by Lemma 2.5. For each of those subsets we can test if a single pair of terminals (p, q) is connected via M-path by breadth-first search in $\mathcal{O}(n^2)$. Testing all n pairs then requires $\mathcal{O}(n^3)$ time, which leaves us with an overall worst case runtime of $\mathcal{O}(n^3 \cdot 256^{n^2})$. By Lemma 3.1, this algorithm eventually will find an optimal network for any 2-GMMN instance.

Lemma 3.2

The number of M-paths connecting both terminals of a box (p, q) within $\mathcal{H}(R)$ is exactly

$$\binom{a+b}{a} = \binom{a+b}{b} = \frac{(a+b)!}{a! \cdot b!},$$

where a denotes the distance of $\pi_x(p)$ to $\pi_x(q)$ in $\pi_x(R)$, i.e. the number of horizontal edges from the left to the right end of the box, and b correspondingly the distance form $\pi_y(p)$ to $\pi_y(q)$ in $\pi_y(R)$.

For this notation we assume $\pi_x(R)$ and $\pi_y(R)$ to be sorted.

Proof : Any M-path consists of a horizontal and b vertical edges within $\mathcal{H}(R)$. Basically we have a bijection between the set of M-paths in $\mathcal{H}(R)$ and the set of bit strings

$$B = \left\{ w \in \{x, y\}^{a+b} \ : \ |w|_x = a, |w|_y = b \right\}$$

by identifying each path with the concatenation of the directions the path takes from p to q. Combinatorics takes the rest. ■

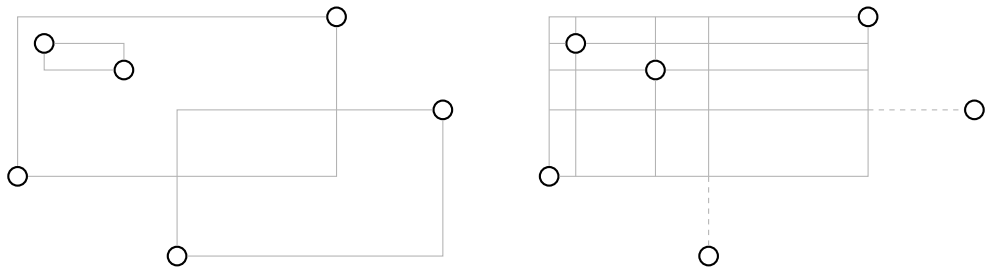

Figure 3.2: left: 2-GMMN instance with boxes; right: local grid for a single box; each terminal of a neighboring box contributes at most 2 lines

For the rest of the chapter, the Hanan grid will be considered a collection of at most $2n$ horizontal and $2n$ vertical lines.

3.1 High Level Idea

Knowing that a minimal network can be found within the Hanan Grid (Lemma 3.1), we might suspect that a stronger restriction is also possible. For a fixed box (p, q), consider the smaller Hanan Grid generated by the smaller instance consisting of the box itself and all of its neighbors (i.e. boxes intersecting (p, q)). This smaller instance has size $m + 1$, where m denotes the degree of (p, q) in the intersection graph. For fixed m, Lemma 3.2 yields a constant upper bound on the number of possible M-paths for that box, namely $\binom{4m+2}{2m+1}$, since every neighbor contributes at most 2 horizontal and 2 vertical lines, one of each through each terminal.

However, there are counterexamples showing that the above restriction to this sort of *local* grid cannot guarantee to always find an optimal solution. In Figure 3.3, consider all vertical distances to be very small, i.e. assume they add up to length less than one of the long horizontal ones. It is easy to verify that the center gap in the local grid forces any solution in this grid to contain at least 7 of the long horizontal edges in contrast to the minimal solution containing only 6. Stretching the instance horizontally will cause the difference between any solution contained in the local grid and a minimal solution to become arbitrarily large.

Also, by adding more thinner boxes on the left and right, this counterexample can be easily modified in such a way, that even the local grid generated from the box, its neighbors and their neighbors (up to the k^{th} iterated neighbors) will not contain a minimal solution.

It is still possible to use Lemma 3.2, which gives us an upper bound on the number of M-paths for a single box. The above idea is slightly modified: every neighboring box contributes at most 2 horizontal and 2 vertical lines (we will see why in the following sections), but simply chosing the 4 lines defining the neighboring box (as we tried before) is not enough. If there are not too many choices, a dynamic program might be able to optimally solve 2-GMMN at least for some 'simple' instances.

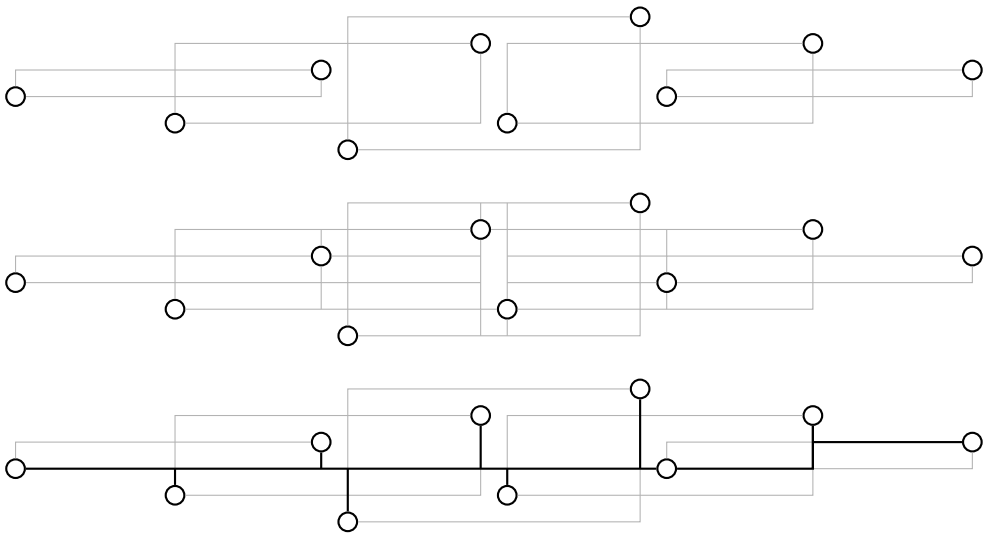

Figure 3.3: top: 2-GMMN instance with boxes; middle: local grid; bottom: minimal network not contained in the local grid

In each of the following sections we consider one specific type of intersection graph and present a corresponding dynamic program.

3.2 Simple Path

In case the intersection graph G of an instance R turns out to be a simple path, each box $(p, q) \in R$ has at most two neighbors. Each of those neighbors must contain an M-path connecting their terminals. Given entry points $p_1^{(\text{in})}, p_2^{(\text{in})}$ and exit points $p_1^{(\text{out})}, p_2^{(\text{out})}$ for both M-paths, $p_i^{(\text{in})}$ and $p_i^{(\text{out})}$ must be connected via M-path, just like p and q. This is in turn a GMMN instance

$$\tilde{R} = \left\{ \left(p, q\right), \left(p_1^{(\text{in})}, p_1^{(\text{out})}\right), \left(p_2^{(\text{in})}, p_2^{(\text{out})}\right) \right\}.$$

In the case the M-path starts (or ends) inside of (p, q), $p_i^{(\text{in})}$ (or $p_i^{(\text{out})}$) is defined as the corresponding terminal.

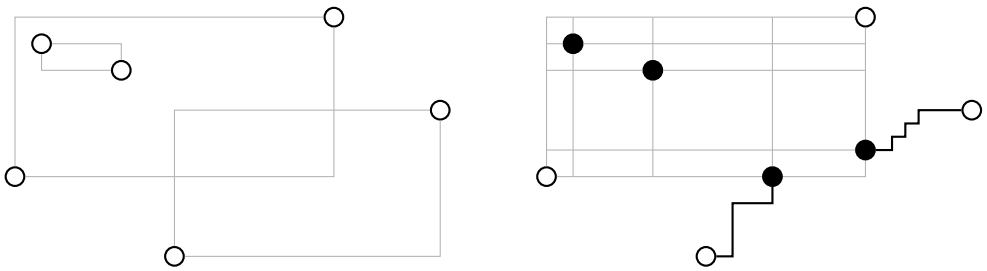

Figure 3.4: left: box with two neighboring boxes; right: relevant Hanan lines for this box, entry and exit points shown in black

Now Lemma 3.1 enables us to restrict ourselves to the grid consisting of at most 12 lines, namely the 4 lines defining the bounding box plus at most 2 additional lines for each entry or exit point of either of the two M-paths (totalling at most 8 extra lines), still being guaranteed to find an optimal network. An example is shown in Figure 3.4. Such lines are called *relevant* lines. Therefore we have $a + b \leq 10$ in Lemma 3.2, so we have a maximum of $\binom{10}{5} = 252$ different M-paths going from p to q along those 12 lines. Remember that we need to know the points where both of the other M-paths enter and leave the box (p, q). So we use a dynamic program to compute optimum network lengths for any set of such points.

We enumerate all terminal pairs from 1 to n according to their position in the intersection graph. Up next, we enumerate all the subsets of size 8 out of the set of all lines formed by the Hanan Grid. Recall that by Lemma 2.5, there are $4n$ lines, which leaves us with at most $N = \binom{4n}{8}$ subsets. Lastly, we enumerate the different M-paths from 1 to 252. This number only describes the way the M-path follows in terms of the grid generated by the current subset of lines, e.g. (left, up, up, ...).

For the dynamic program, we want to fill a 3-dimensional matrix A with entries $\square_{i,j,k}$ where $i \in \{1, \ldots, n\}$ $j \in \{1, \ldots, N\}$, $k \in \{1, \ldots, 252\}$. Each entry $\square_{i,j,k}$ is supposed to indicate the minimum length of a valid Manhattan network for the first i terminal pairs, assuming that for box i itself j is the subset of relevant lines and box i is connected via the k-th M-path with respect to the subset of lines known by index j.

For $i = 1$ we can achieve this by setting

$$\square_{1,j,k} = d_1(p_1, q_1)$$

for all j, k. Via induction on i we compute

$$\square_{i,j,k} = \min_{j',k'} \{\square_{i-1,j',k'} + d_1(p_i, q_i) - c(j, k, j', k')\},$$

where the minimum is taken over all $j' \in \{1, \ldots, N\}$ and $k' \in \{1, \ldots, 252\}$. The term $c(j, k, j', k')$ denotes the accumulated length of all edges contained both in the network used for $\square_{i-1,j',k'}$ and in the network obtained by connecting (p_i, q_i) on the subgrid j via M-path k.

It should be clear, that this process will always produce a valid network for boxes 1 to i with the desired requirements (see above). So for $i = n$ we can compute an optimal solution:

$$\ell_{\text{opt}} = \min_{j,k} \{\square_{n,j,k}\}.$$

Now that we know the length of a minimum Manhattan network, we can use a simple backtracking algorigthm to find the network itself.

The matrix A has size $n \times N \times 252 = \mathcal{O}(n) \cdot \mathcal{O}(n^8) \cdot \mathcal{O}(1) = \mathcal{O}(n^9)$. Computing a single entry corresponds to taking a minimum of $252N = \mathcal{O}(n^8)$ previously calculated entries, so we have a total runtime of $\mathcal{O}(n^9) \cdot \mathcal{O}(n^8) = \mathcal{O}(n^{17})$.

3.3 Circle

Provided the intersection graph G is a circle, we can use a similar dynamic program, since a path and a circle are pretty similar. First, we chose an arbitrary vertex $(p_1, q_1) \in R$ to start with and enumerate the remaining vertices in order. We also need an enumerated list of all the $252N$ different M-paths connecting p_1 to q_1. We add another dimension to the dynamic program, so we have new matrix entries $\square_{i,j,k,l}$, where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, N\}$, $k \in \{1, \ldots, 252\}$ and $l \in \{1, \ldots, 252N\}$. Like before, j and k represent the M-path for the i-th box, we refer to this as the *configuration* of a box. l represents the configuration for the first box (p_1, q_1).

We want to maintain the following invariant: every $\square_{i,j,k,l}$ denotes the minimum length of a Manhattan network for the boxes 1 through i, where the configuration for (p_i, q_i) is given by j and k and the configuration for (p_1, q_1) is given by l. We can basically achieve this as follows:

Let $i = 1$, then

$$\square_{1,j,k,l} = d_1(p_1, q_1)$$

for all j, k, l. Note that j and k are redundant here. Now for $1 < i < n$ we set

$$\square_{i,j,k,l} = \min_{j',k'} \left\{ \square_{i-1,j',k',l} + d_1(p_i, q_i) - c(j, k, j', k') \right\}.$$

For $i = n$ we've come full circle. So we set

$$\square_{n,j,k,l} = \min_{j',k'} \left\{ \square_{n-1,j',k',l} + d_1(p_n, q_n) - c(j, k, j', k') - c(j, k, l) \right\}.$$

As before, we minimize with respect to $j' \in \{1, \ldots, N\}$ and $k' \in \{1, \ldots, 252\}$ in the standard case $1 < i < n$. The most important difference is of course the last case, $i = n$. Here, box n is not only adjacent to box $n-1$, but also to box 1. However, we cannot continue the same scheme, since we already picked a configuration for box 1, namely configuration l. Therefore, we must substract $c(j, k, l) = c(j, k, j'', k'')$, where j'', k'' are the redundant indices (determined by l) from case $i = 1$.

Now that we are sure that $\square_{n,j,k,l}$ gives us the minimum length of a network for all boxes on condition that the first box is in configuration l, we compute

$$\ell_{\text{opt}} = \min_{j,k,l} \left\{ \square_{n,j,k,l} \right\},$$

minimizing over $j \in \{1, \ldots, N\}$, $k \in \{1, \ldots, 252\}$ and $l \in \{1, \ldots, 252N\}$. So we have found the minimal length of a Manhattan network.

The total runtime has increased, the matrix A has size $n \times N \times 252 \times 252N = \mathcal{O}(n^{17})$ now. However, computing a single entry $\square_{i,j,k,l}$ still takes $\mathcal{O}(n^8)$ time, since we only minimize over j' and k' while l is fixed. Overall, we end up in $\mathcal{O}(n^{25})$.

3.4 Tree

If R is a 2-GMMN instance whose intersection graph G is a tree, we can follow the same basic strategy. First, we need the following lemma:

Lemma 3.3
Given a single box with m neighbors, there are at most $\binom{4n}{4m} \cdot \binom{4m+2}{2m+1}$ possible M-paths within the Hanan Grid $\mathcal{H}(R)$.

Proof : Denote by p_i and q_i the points, where the i-th neighbor's M-path enters and leaves the box (p, q), i.e. p_i is either a terminal of the neighbor or lies on the boundary of (p, q). We can then apply Lemma 3.1 to the set

$$\tilde{R} = \{(p_i, q_i) \ : \ i \leq m\} \cup \{(p, q)\} .$$

Lemma 3.1 also allows us to assume the terminals p_i, q_i are contained in the Hanan Grid. So if we know the p_i, q_i, we have at most $2m + 2$ points generating a grid of $4m + 4$ lines. We have $\binom{4n}{4m}$ possible ways to chose a subset of size $4m + 4$ from the $4n$ lines of the Hanan Grid if we demand the set to contain the lines through p and q. By Lemma 3.2 there are $\binom{4m+2}{2m+1}$ possible M-paths for each subset. ■

This time we want $\square_{i,j,k}$ to be the minimum length of a network for the subtree under v_i when for (p_i, q_i) itself the configuration is given by j and k.

Let v_i be a leaf, then

$$\square_{i,j,k} = d_1(p_i, q_i)$$

for all j, k. Now for inner vertices $v_i = (p_i, q_i)$ we set

$$\square_{i,j,k} = \min_{j',k'} \left\{ \sum_{i'} \square_{i',j',k'} - c(j, k, j', k') + d_1(p_i, q_i) \right\} . \tag{3.1}$$

Now that there is no canonical ordering of the vertices, we proceed step by step. In the first step, we compute all the partial solutions $\square_{i,j,k}$ for leaves v_i of the intersection graph G. Next, all the vertices we have yet to deal with form a subtree of G. We consider a leaf v_i of this subtree. Since we already know the partial solutions for all of its children $v_{i'}$, we compute optimal partial solutions for the subtree under v_i via (3.1). Finally we arrive at some index i_r where v_{i_r} is the last vertex left in the subtree i.e. the root. In case there are 2 vertices left, we just chose one of them to be the root, then we make one more step and we are left with just the root. So the minimum length is

$$\ell_{\text{opt}} = \min_{j,k} \{\square_{i_r,j,k}\} ,$$

minimizing over j and k.

For each v_i we have to consider at most $\binom{4n}{4m} \cdot \binom{4m+2}{2m+1}$ different M-paths by Lemma 3.3, where $m = \deg(v_i)$. For each of these paths we compute a minimum over

$$\begin{aligned} \prod_{i'} \binom{4n}{4m_{i'}} \binom{4m_{i'}+2}{2m_{i'}+1} &= \prod_{i'} \mathcal{O}(n^{4m_{i'}}) \\ &= \mathcal{O}(n^{4 \cdot \sum m_{i'}}) \end{aligned} \tag{3.2}$$

combinations of already computed partial solutions in (3.1). Here, $m_{i'} = \deg(v_{i'})$.

In order to guarantee (3.2) to be a polynomial bound on the runtime, we demand m as well as all of the $m_{i'}$ to be bounded by a constant $M \in \mathbb{N}$, i.e. we only consider instances where the intersection graph has maximum degree at most M.

The overall runtime can then be bounded by

$$n \cdot \mathcal{O}(n^{4 \cdot \sum m_{i'}}) = n \cdot \mathcal{O}(n^{4M^2}) = \mathcal{O}(n^{4M^2+1}) = \mathcal{O}(n^{\mathcal{O}(1)}),$$

assuming $m = m_{i'} = M$ for all vertices in (3.2).

Summarizing the first 3 sections of this chapter, we get the following:

Theorem 3.4
Let G be the intersection graph of a GMMN instance R and suppose G is a path, a circle or a tree with bounded degree.

Then an optimal GMMN for R can be found in polynomial time.

3.5 Union Graph

In this section, we want to extend the methods we used before to more general instances, i.e. more general intersection graphs.

Definition 3.5
A graph $G = (V, E)$ is called *union graph with interface* c (where $c \in \mathbb{N}$), if there exists a rooted tree T satisfying the following properties:

(a) Every vertex of T is a subset $\emptyset \neq V_i \subseteq V$, the root of T is V.

(b) Every leaf of T is a singleton, i.e. $V_i = \{v\} \subseteq V$.

(c) The subset V_i of any inner vertex is partitioned amongst its children.

(d) The accumulated number of *interface vertices* of the children of V_i does not exceed c. In particular, each child alone has at most c interface vertices.

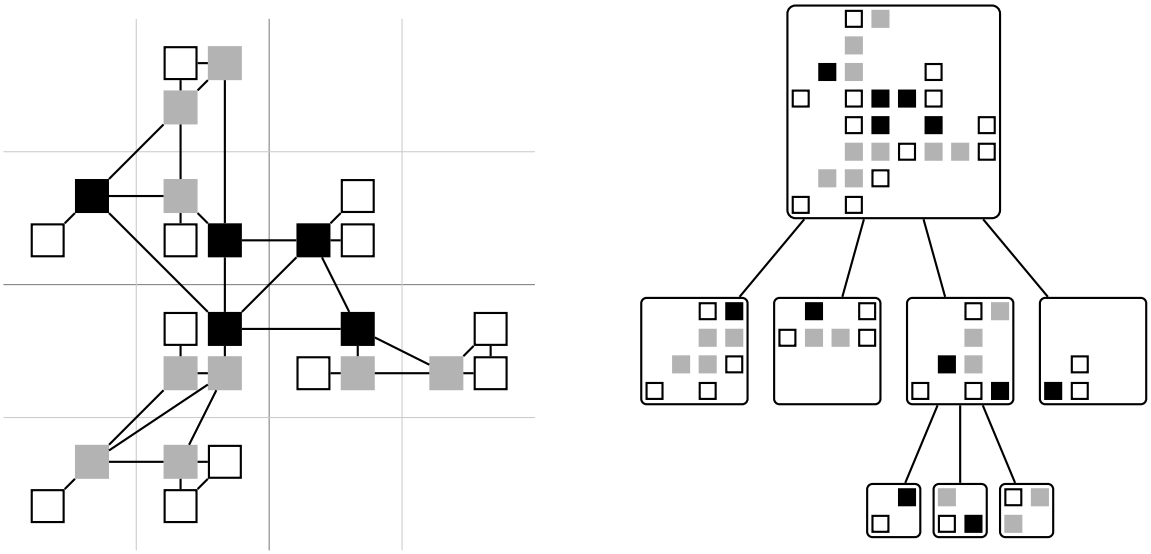

Figure 3.5: example of a union graph with interface 5 and its partially drawn union tree, interface vertices of the first (second) level shown in black (gray), note that black implies gray by definition

A vertex $v \in V_i \subseteq V$ is called *interface vertex* of V_i, if there exists a vertex $w \in V \setminus V_i$ such that $(v, w) \in E$ or equivalently, if there exists an edge between V_i and the rest of V. This tree T is called a *union tree*.

Definition 3.6

The *height* of a tree vertex v is defined as follows:

If v is a leaf, then height$(v) = 1$. If v is an inner vertex, then height$(v) = h+1$, where h denotes the height of its highest child.

If the intersection graph G is a union graph with maximum degree M, the same strategy can be used again.

We start with height $h = 1$ in the union tree T, let V' be a leaf, then $V' = \{v\}$ for some $v \in V$. For a single vertex there are exactly $\binom{4n}{4m} \cdot \binom{4m+2}{2m+1}$ possible ways to connect the corresponding terminal pair (p, q) via M-path. Once again, m denotes the degree of v in G. For each of these configurations we save the length of the corresponding network, i.e. we always save $d_1(p, q)$.

We want to maintain the following invariant: for each configuration of the interface vertices of $V' \subseteq V$, we want to know the minimum length of a network for V'.

So let V' be a tree vertex of height $h > 1$. Inductively, we assume that we already know the desired lenghts for each configuration of the interface vertices of any child

of V', since their height is less than h. Label the children $V_1, \ldots, V_k$. Let I be the union of the interface vertices of $V_1, \ldots, V_k$ and note that $|I| \leq c$. We consider all c-tuples of configurations of all the interface vertices in I, there are at most

$$\left(\binom{4n}{4M} \cdot \binom{4M+2}{2M+1}\right)^c = (\mathcal{O}(n^{4M}))^c = \mathcal{O}(n^{4cM}) \tag{3.3}$$

of them by Lemma 3.3. For every combination of configurations of the interface vertices of any V_j we already know an optimal network. The union of these k optimal networks for $V_1, \ldots, V_k$ sure yields a valid network for V'. Now we look at the interface vertices of V', recall that there are at most c of those. For these c vertices, there are again

$$\left(\binom{4n}{4M} \cdot \binom{4M+2}{2M+1}\right)^c = (\mathcal{O}(n^{4M}))^c = \mathcal{O}(n^{4cM})$$

combinations of single configurations. So as we construct valid networks for V', we know they are not necessarily optimal, but if we always store the shortest one we have encountered so far for each configuration, we will end up with exactly what we wanted: an optimal network for V' when being restricted to a single configuration of all of its interface vertices. We iterate this process until we reach the full height of the union tree. For the root, where $V' = V$ by definition, there are no interface vertices, we can just take the minimum of all the unions of optimal solutions for the children yielding an optimal solution for R.

Theorem 3.7

Let R be a 2-GMMN instance and G the intersection graph of R. Furthermore, suppose G is a union graph with interface c and the maximum degree is bounded by some constant M. Then a minimum Manhattan network can be found in polynomial time.

Proof : For the algorithm: see above. Optimality for R follows from optimality for individual terminal pairs (trivial, see algorithm) and inductively by the maintained invariant. Having all optimal solutions for all tree vertices V' of height h, parametrized by the configurations of their respective interface vertices, we find the new optimal solutions for height $h+1$ by minimizing over all possible configurations and then parametrize those by the configurations of this height's interface vertices. Therefore, the obtained network is optimal for R.

For the runtime: we have $\mathcal{O}(n^{4M})$ configurations for each leaf by Lemma 3.3, so for at most n leaves we get a runtime of $\mathcal{O}(n^{4M+1})$. For each tree vertex we have $\mathcal{O}(n^{4cM})$ look-ups (see (3.3)) and the computation of the union within the set of edges in $\mathcal{H}(R)$ (which is linear in the number of edges, which is $2 \cdot (2n)^2$ c.f. Lemma 2.5) in $\mathcal{O}(n^2)$. We assume the union tree to be redundance-free, i.e. there cannot be a vertex V' with only one child V'', implying $V' = V''$. Therefore, the number of tree vertices is at most n, so the total runtime is bounded by $n \cdot \mathcal{O}(n^{4cM+1}) = \mathcal{O}(n^{4cM+2})$.
■

3.6 Series-parallel Graph

Definition 3.8

A graph $G = (V, E)$ is called a *series-parallel graph* or *sp-graph* for short, if one of the following conditions holds:

- $G = K_2$, i.e. $V = \{v_1, v_2\}$ and $E = \{(v_1, v_2)\}$.
- $G = C_S(G_1, G_2)$, i.e. G is the *series composition* of two sp-graphs G_1, G_2.
- $G = C_P(G_1, G_2)$, i.e. G is the *parallel composition* of two sp-graphs G_1, G_2.

A series parallel graph has two distinguished vertices $s, t \in V$. s will be referred to as *source* and t as *target*, in the literature t may be called *sink*. The compositions are defined as follows:

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be sp-graphs. Then

$$\begin{aligned} V' &= (V_1 \cup V_2) \text{ identifying } s_2 = t_1, \\ E' &= E_1 \cup E_2 \end{aligned}$$

and the series composition is

$$C_S(G_1, G_2) = (V', E') \text{ with source } s' = s_1 \text{ and target } t' = t_2.$$

Furthermore

$$\begin{aligned} V^* &= (V_1 \cup V_2) \text{ identifying } s_1 = s_2, t_1 = t_2 \\ E^* &= E_1 \cup E_2 \end{aligned}$$

and the parallel composition is

$$C_P(G_1, G_2) = (V^*, E^*) \text{ with source } s^* = s_1 \text{ and target } t^* = t_1.$$

In this definition we always assume V_1 and V_2 to be disjoint before a composition is formed. Also, if $G_i = K_2$ and $(s_j, t_j) \in E_j$ for $i \neq j$, we set

$$C_P(G_1, G_2) = G_j$$

to avoid parallel edges.

Proposition 3.9
Every series parallel graph with maximal degree M is a union graph.

Proof : Let $G = K_2$, then the union tree in Figure 3.6 will proof the claim.

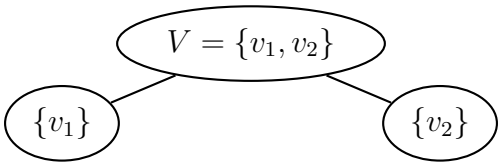

Figure 3.6: union tree for $G = K_2$

In this case $c = 2$.

Now let $G = C_S(G_1, G_2)$, where G_1, G_2 are series parallel. Since they have less vertices, they are union graphs (with interfaces c_1, c_2 respectively) by induction. Let

$$V_1 = \{v_1, \ldots, v_n\},$$
$$V_2 = \{w_1, \ldots, w_m\}$$

where $v_1 = s_1, v_n = t_1, w_1 = s_2$ and $w_m = t_2$. Then (using the identification $v_n = w_1$) $V = \{v_1, \ldots, v_n, w_2, \ldots, w_m\}$. Union trees T_1, T_2 for G_1, G_2 exist by induction, we stick them together as shown in Figure 3.7.

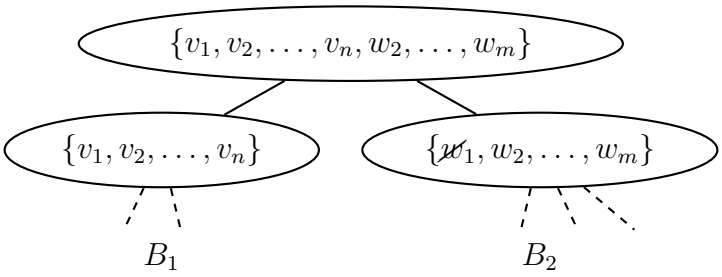

Figure 3.7: union tree for $G = C_S(G_1, G_2)$ constructed out of T_1 and T_2

For this construction to result in a union tree, we delete w_1 in every tree vertex that comes from T_2. Deleting w_1 cannot increase the number of interface vertices. The leaf $\{w_1\}$ (or $\emptyset$ by then) is now deleted completely. In case two tree vertices end up being identical, we melt them together. Note, that this can only happen if one is the parent and the other one is his only child. Since the vertices were identical, the bound c_2 is not violated.

We make sure the tree T we constructed is indeed a union tree for $G = C_S(G_1, G_2)$. The root is $V = \{v_1, \ldots, v_n, w_2, \ldots, w_m\}$ and none of the vertices is $\emptyset$. The children of any vertex V' are disjoint after w_1 is deleted and their union yields V' again. Also,

by construction, leaves are singletons. What remains to show is that the number of interface vertices is bounded by some constant $c \leq M$. For T_1, T_2 this holds already by induction. Only the root V needs to be checked. Here, the left child is V_1, the right child is $V_2 \setminus \{w_1\}$. G_1 and G_2 are only connected via one vertex which is $v_n = w_1$. Therefore, V_1 contains just 1 interface vertex, namely v_n. The interface vertices of $V_2 \setminus \{w_1\}$ consists of all neighbors of v_n in V_2. Since we assume G to be connected, v_n has at least 1 neighbor in V_1, so we have at most $M - 1$ of them in V_2. Overall the number of interface vertices is bounded by $1 + M - 1 = M$. Deleting v_n instead of w_1 might lower that bound for concrete cases, depending on whether v_n has less or more neighbors in V_1 than w_1 has in V_2. Either way, c is bounded by M and we conclude: T is a valid union tree with interface $c \leq \max\{c_1, c_2, M\}$.

Now let $G = C_P(G_1, G_2)$, where G_1, G_2 are series parallel with vertex sets

$$V_1 = \{v_1, \ldots, v_n\},$$
$$V_2 = \{w_1, \ldots, w_m\}$$

where $v_1 = s_1, v_n = t_1, w_1 = s_2$ and $w_m = t_2$. Identifying $v_1 = w_1$ and $v_n = w_m$ we have $V = \{v_1, \ldots, v_{n-1}, w_2, \ldots, w_m\}$.

By induction there exist union trees T_1, T_2 with interfaces c_1, c_2 respectively. Again we can merge them into one tree, see Figure 3.8.

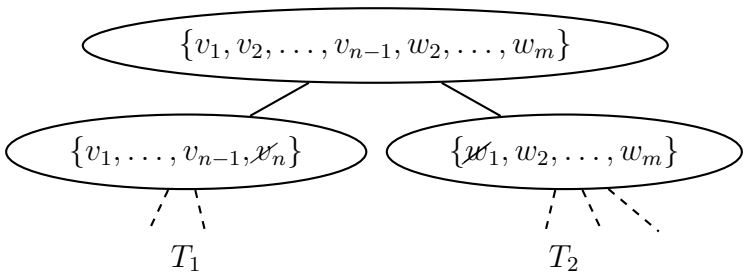

Figure 3.8: union tree for $G = C_P(G_1, G_2)$ constructed out of T_1 and T_2

Here, we delete not only the leaf $\{w_1\}$, but also $\{v_n\}$. Apart from that, we can use the same procedure and the result is a valid union tree for $G = C_P(G_1, G_2)$. The main difference is the interface, in this case we get $c = \max\{c_1, c_2, 2M\}$, since both w_1 and v_n can have $M - 1$ neighbors in the opposing vertex sets. ■

3.7 Remarks

For all the graphs mentioned above (paths, circles, trees and sp-graphs), there are in fact 2-GMMN instances having those as their intersection graphs. Paths are almost trivial and circles likewise. For any tree, a 2-GMMN instance can be constructed as follows: chose any vertex to be the root and construct a $n \times 3$ box. For the i-th child, construct a $n_i \times 3$ box, where n_i is the size of its subtree. Since $\sum n_i = n - 1 < n$,

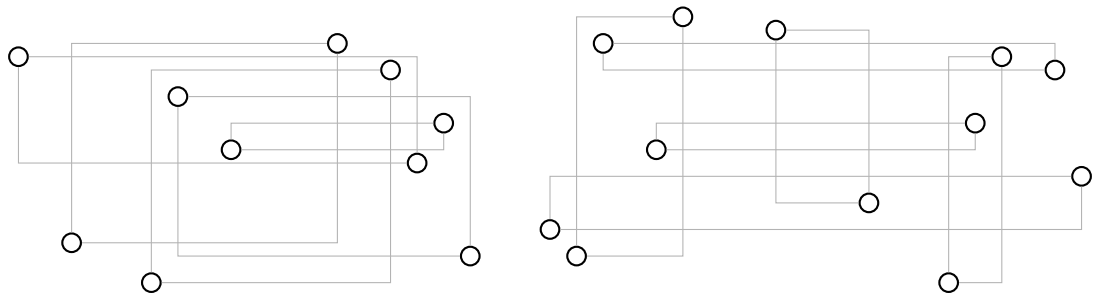

Figure 3.9: instances for intersection graphs K_5 and $K_{3,3}$ with bounding boxes

it is possible to fit those boxes next to each other without overlap. Also, shift them down by 2, then repeat the process with the subtrees. Series-parallel graphs can be constructed in a straightforward fashion, just maintain the invariant, that the box corresponding to the source (target) is always the highest (lowest) box, i.e. there is a horizontal line interesecting the source box such that every other box lies completely below (above).

Whether a graph can occur as an intersection graph of a 2-GMMN instance does not seem to depend on planarity: there are instances with non-planar intersection graphs, c.f. the complete graph K_5 and the bipartite graph $K_{3,3}$ shown in Figure 3.9.

On the other hand, we found planar graphs with no matching 2-GMMN instance, e.g. the diamond in Figure 3.10.

Label the grey vertices v_1, v_2 and their boxes respectively. Use indices $3, \ldots, 6$ (in order) for the rest. Consider the two projections of v_1 and v_2 onto the coordinate axes. If both $\pi_1(v_1) \cap \pi_1(v_2) = \emptyset$ and $\pi_2(v_1) \cap \pi_2(v_2) = \emptyset$, then every other box v_i must contain the same corner of v_1, thus $v_3 \cap v_4 \cap v_5 \cap v_6 \neq \emptyset$, a contradiction. If both of these intersections are non-empty, then $v_1 \cap v_2 \neq \emptyset$, another contradiction. So we can assume w.l.o.g. that $\pi_1(v_1) \cap \pi_1(v_2) \neq \emptyset = \pi_2(v_1) \cap \pi_2(v_2)$, i.e. the situation of the boxes looks like in Figure 3.11.

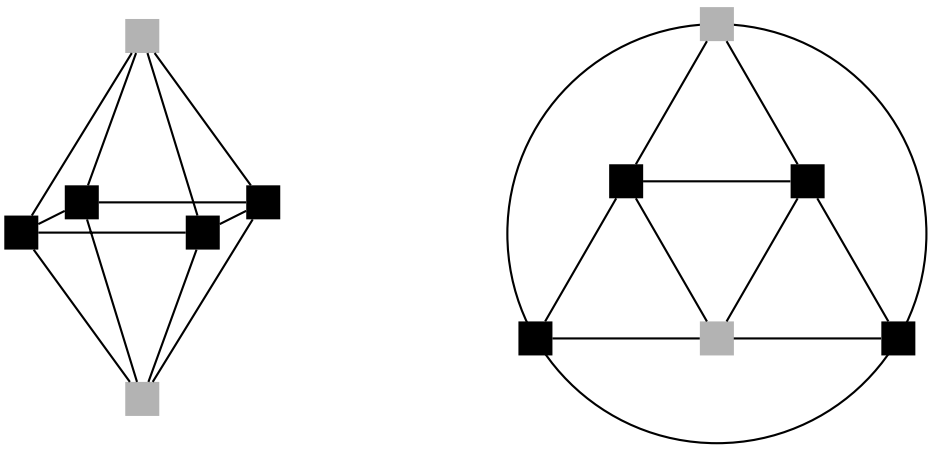

Figure 3.10: planar graph and planar embedding with no 2-GMMN instance

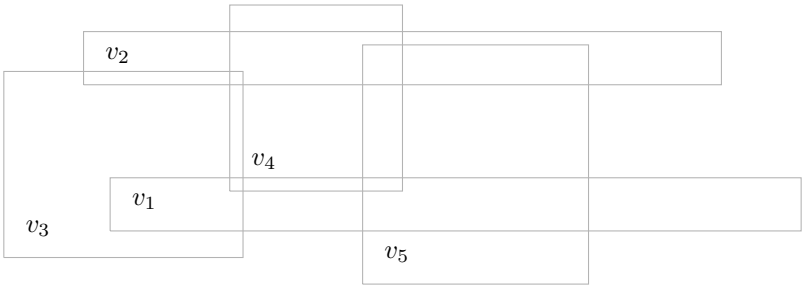

Figure 3.11: 2-GMMN almost-instance for the graph shown in Figure 3.10

This layout is forced by the fact that $(v_3, v_4), (v_4, v_5) \in E$ and $(v_3, v_5) \notin E$. By convexity it is impossible to have another box v_6 intersect v_1, v_2, v_5 and v_6 without intersecting v_4. Therefore, even though it is planar, the diamond in Figure 3.10 is not an intersection graph of any 2-GMMN instance.

4 Higher Dimensions

Now we drop the requirement $d = 2$ and consider arbitrary $d \in \mathbb{N}$. It is worth mentioning that F. S. Roberts introduced a graph parameter called *boxicity* [Rob69]. The boxicity of G is the minimum d such that there are n boxes in $\mathbb{R}^d$ with intersection graph G. Roberts proved that the boxicity of a graph with n vertices is bounded by $\lfloor \frac{n}{2} \rfloor$. This means every graph can occur as the intersection graph of some d-GMMN instance as long as d is big enough. For example, Figure 3.10 can be obtained from a 3-GMMN instance, even though it was impossible from 2-GMMN. So in general, we cannot restrict ourselves to a proper subset of graphs. Although, for fixed d it might be possible, e.g. interval graphs in the case $d = 1$.

4.1 Generalizations

We want to generalize Lemma 3.1:

Lemma 4.1
Let N be a (not necessarily optimal) Manhattan network for a d-GMMN instance R. Then there exists a Manhattan network N' which is completely contained in the Hanan Grid for R and also satisfies

$$\ell(N') \leq \ell(N).$$

Particularly, the existance of an optimal network implies the existance of an optimal network within the Hanan Grid.

The proof is analogous to the proof of Lemma 3.1, the idea is taken from [FS14].

Proof : We only proof the result for one coordinate x_1, it follows analogously for $x_2, \ldots, x_d$. Let $\pi_1(R) = \{\xi^{(1)}, \ldots, \xi^{(k)}\}$ be the projection of the terminals in R onto their x_1-coordinate and assume $\xi^{(1)} \leq \cdots \leq \xi^{(k)}$. Let H be the hyperplane perpendicular to x_1 containing $\xi^{(1)}$, i.e. $H : x_1 = \xi^{(1)}$. We then sweep H upwards (increasing its x_1-coordinate). By induction we assume the part of N below H is already contained in the Hanan Grid. Suppose at some height $\xi^* \notin \pi_1(R)$ a subset E' of the edges of N is contained in H and denote by $E_\uparrow, E_\downarrow$ the set of edges parallel to x_1 incident to H from below and above respectively.

If $|E_\uparrow| \geq |E_\downarrow|$ we can shift E' downward until we reach the next x_1-value in $\mathcal{H}(R)$, thereby extending the edges from above and shortening the edges from below. Therefore, the overall length of the network does not increase. Every M-path passing through H must start below and end above H or vice versa. Since one edge is shortened by the same length the other one is extended, the length of the path does not change, so M-paths are preserved.

If $|E_\uparrow| < |E_\downarrow|$ we can, in a similar fashion, shift E' upwards. To satisfy the induction invariant, we can only shift E' to either the next $\xi^{(j)}$ or to the next ξ^{**} containing some other subset E'', whichever comes first. ■

Lemma 4.2

The number of M-paths connecting both terminals of a box (p, q) within $\mathcal{H}(R)$ is exactly

$$\frac{(\sum a_i)!}{a_1! \cdots a_d!},$$

where a_i denotes the distance of $\pi_i(p)$ to $\pi_i(q)$ in $\pi_i(R)$.

Again, the proof is analogous to the proof of Lemma 3.2.

Proof : Any M-path for (p, q) consists of $a_1 + \cdots + a_d$ edges within $\mathcal{H}(R)$. Basically we have a bijection between the set of M-paths in $\mathcal{H}(R)$ and the set of strings

$$B = \{w \in \{x_1, \ldots, x_d\}^* \ : \ |w|_{x_i} = a_i\}$$

by identifying each path with the concatenation of the directions the path takes from p to q. Combinatorics takes the rest. ■

The following generalizes Lemma 3.3:

Lemma 4.3

Given a single box (p, q) with m neighbors, the number of possible M-paths within the Hanan Grid $\mathcal{H}(R)$ is at most

$$\frac{(d \cdot (2dm + 1))!}{((2dm + 1)!)^d} \cdot \binom{2dn}{2dm}.$$

Proof : For $m = 0$ every M-path uses exactly d edges, one in each direction $x_1, \ldots, x_d$. The order can be chosen freely, so there are $d! = \frac{d!}{1!\cdots 1!}$ different M-paths. For $m \geq 1$, any new M-path intersecting (p, q) has some entry point e_i and some exit point f_i on the boundary of the box. Alternatively, if one or both terminals of the i-th neighboring box are within (p, q), the terminal(s) substitute(s) e_i or/and f_i. In the worst case, both e_i and f_i are contained in the interior of the box. Applying Lemma 4.1 to

$$\tilde{R} = \{(e_i, f_i) \ : \ i \leq m\} \cup \{(p, q)\},$$

we can restrict our search to $\mathcal{H}(\tilde{R})$ if we know the entry and exit points. Each of the new vertices increases the length of an M-path from p to q in terms of the number of Hanan edges by at most d (1 in each dimension). So we have $a_1 = \ldots = a_d = 2m \cdot d + 1$ in Lemma 4.2, which shows that the number of M-paths for a fixed set of entry and exit points is bounded by

$$\frac{(d \cdot (2dm + 1))!}{((2dm + 1)!)^d}.$$

We have $2dn$ straight lines in $\mathcal{H}(R)$, so there are $\binom{2dn}{2dm}$ possible ways to chose a subset of size $2dm$. ■

Of course the goal is to generalize Theorem 3.7, so while the overall strategy we used for union graphs remains the same, the runtime bounds we derived from Lemma 3.2 must be replaced by bounds from Lemma 4.2.

Theorem 4.4

Let R be a d-GMMN instance and G the intersection graph of R. Furthermore, suppose G is a union graph with interface c and the maximum degree is bounded by some constant M. Then a minimum Manhattan network can be found in polynomial time.

Proof : For the algorithm: like in Theorem 3.7, we start with height $h = 1$ in the union tree T. Leaves are singletons and by Lemma 4.3, for a single vertex v with $\deg(v) = m$ the number of possible M-paths connecting the corresponding terminal pair (p, q) in $\mathcal{H}(R)$ is bounded by

$$\frac{(d \cdot (2dm + 1))!}{((2dm + 1)!)^d} \cdot \binom{2dn}{2dm}.$$

Note that for $m \leq M$ and a fixed dimension d, only the binomial coefficient depends on n, the factor is a constant. For each of these configurations we save the length of the corresponding network, i.e. we always save $d_1(p, q)$.

We want to maintain the following invariant: for each configuration of the interface vertices of $V' \subseteq V$, we want to determine the minimum length of a network for V'.

So let V' be a tree vertex of height $h > 1$. Inductively, we assume that we already know the desired lenghts for each configuration of the interface vertices of any child of V', since their height is less than h. Label the children $V_1, \ldots, V_k \subseteq V$. Let I be the union of the interface vertices of $V_1, \ldots, V_k$ and note that $|I| \leq c$. We consider all c-tuples of configurations of all the interface vertices in I, there are at most

$$\left(\frac{(d \cdot (2dM+1))!}{((2dM+1)!)^d} \cdot \binom{2dn}{2dM}\right)^c = (\mathcal{O}(n^{2dM}))^c = \mathcal{O}(n^{2cdM}) \tag{4.1}$$

of them by Lemma 4.3. For every combination of configurations of the interface vertices of any V_j we already know an optimal network. The union of these k optimal networks for $V_1, \ldots, V_k$ sure yields a valid network for V'. Now we look at the interface vertices of V', recall that there are at most c of those. For these c vertices, there are

$$\left(\frac{(d \cdot (2dm+1))!}{((2dm+1)!)^d} \cdot \binom{2dn}{2dm}\right)^c = (\mathcal{O}(n^{2dM}))^c = \mathcal{O}(n^{2cdM})$$

combinations of single configurations. So as we construct valid networks for V', we know they are not necessarily optimal, but if we always store the shortest one we have encountered so far for each configuration, we will end up with exactly what we wanted: an optimal network for V' when being restricted to a single configuration of all of its interface vertices. We iterate this process until we reach the full height of the union tree. For the root, where $V' = V$ by definition, there are no interface vertices, we can just take the minimum of all the unions of optimal solutions for the children yielding an optimal solution for R.

Optimality for R follows from optimality for individual terminal pairs (trivial, see algorithm) and inductively by the maintained invariant. Having all optimal solutions for all tree vertices V' on some height h, parametrized by the configurations of their respective interface vertices, we find the new optimal solutions for height $h+1$ by minimizing over all possible configurations and then parameterize those by the configurations of this height's interface vertices. Therefore, the network we obtain in the end, is optimal for R.

For the runtime: we need $\mathcal{O}(n^{2dM})$ for each leaf by Lemma 4.3, where $m \leq M$, so for n leaves we get a total of $\mathcal{O}(n^{2dM+1})$. For each tree vertex we have $\mathcal{O}(n^{2cdM})$ look-ups (see (4.1)) and the computation of the union within the set of edges in $\mathcal{H}(R)$ (which is linear in the number of edges, which is $d \cdot (2n)^d$ c.f. Lemma 2.5) in $\mathcal{O}(n^d)$. We assume the union tree to be redundance-free, i.e. there cannott be a vertex V' with only one child V'', implying $V' = V''$. Therefore, the number of tree vertices is at most n, so the total runtime is bounded by $\mathcal{O}(n^{2dM+1}) + \mathcal{O}(n^{2cdM+1}) + \mathcal{O}(n^{d+1}) = \mathcal{O}(n^{2cdM+1})$. ■

5 Treewidth

In Chapter 3 we showed that Simple paths, circles, trees and series parallel graphs have one thing in common: for each of these graph classes there exists an upper bound on a graph parameter called the treewidth.

We give a short introduction to treewidth, for further information see [Die12].

5.1 Introduction

Given an intersection graph G, we are interested in knowing how similar the structure of G is to a structure of a tree.

Definition 5.1

A *decomposition tree* of a graph $G = (V(G), E(G))$ is a tree $D = (V(D), E(D))$ satisfying the following conditions:

(i) Every tree node is a subset of $V(G)$.

(ii) Every vertex $v \in V(G)$ appears in at least one tree node.

(iii) For every edge $(v, w) \in E(G)$ there is a tree node $X \in V(D)$ containing both v and w.

(iv) For $v \in V(G)$, if v appears in two tree nodes X_i, X_j, then v is contained in every tree node on the path connecting X_i to X_j in D.

The *width* of a decomposition tree D is

$$\text{width}(D) = \max\{|X| - 1 \ : \ X \in V(D)\}.$$

Among all valid decomposition trees for G, the minimum width is called the *treewidth* of G. We write $\text{tw}(G)$ for short.

Following the notation in [Bod98], elements of $V(G)$ are called vertices and elements of $V(D)$ are nodes. While decomposition trees of a graph G are ambiguous, the treewidth is unique. To see this, we consider the trivial decomposition tree where $V(D) = \{V(G)\}$ and $E(D) = \emptyset$. Therefore $\text{tw}(G) \leq n - 1$.

Trees have treewidth 1 due to the fact, that we added the -1 in the definition of width. A decomposition tree of width 1 has a node for every edge in the original tree.

Remark 5.2
Condition (iv) is equivalent to the following:

For every vertex $v \in V(G)$, the set of all nodes X_i containing v forms a connected subtree D_v of D.

5.2 Main Results

Theorem 5.3
Let G be an intersection graph of a d-GMMN instance R. Suppose there exist constants $M, k \in \mathbb{N}$ bounding the maximum degree and the treewidth of G respectively.

Then G is a union graph.

Proof : Using a decomposition tree D of G, we constuct a union tree T. Let

$$V(D) = \{X_i \ : \ i \in I\}$$

for some finite index set I. Then we set

$$V(T) = \{Y_i \ : \ i \in I\} \cup \{\{v\} \ : \ v \in V(G)\}\,.$$

We connect Y_i to $Y_{i'}$ in T whenever X_i and $X_{i'}$ are neighbors in D, so we get the same basic tree structure. Since D is a tree, we proceed as follows: for each vertex $v \in V(G)$ we chose one node $X_i \in V(D)$ of maximal height such that $v \in X_i$. Here, we introduce an edge between $\{v\}$ and Y_i. All we have to do now, is to define the vertices Y_i of T. We start with $Y_i = \emptyset$ for all $i \in I$. In ascending height we then add to each Y_i the vertices contained in all of its children. We delete empty vertices and, as before, if two vertices are identical, we merge them into one.

Every vertex Y_i of T is now a subset of V, the root is V since every vertex $v \in V(G)$ was inserted at some point. Leafs are singletons, since empty Y_i's are deleted. The union of the children yields the parent and the children are disjoint because each $v \in V(G)$ was inserted exactly once. Now consider some vertex Y_i with child Y_j and suppose $v \in Y_j$ is an interface vertex. We have 2 cases:

$v \in X_j$: This case can occur $k+1$ times, since $\text{tw}(G) = k$.

$v \notin X_j$: Then the entire subtree D_v must lie below X_j in D. Since v is an interface vertex, there exists $w \in V(G) \setminus Y_j$ such that $(v, w) \in E(G)$. By definition, D has a node $X_{i'}$ containing both v and w. Of course $X_{i'}$ belongs to D_v and therefore

$$\text{height}(X_{i'}) < \text{height}(X_j) < \text{height}(X_i).$$

Note that $w \notin Y_j$ implies $w \in X_{i''}$ for some $X_{i''}$ with $\text{height}(X_{i''}) \geq \text{height}(X_i)$. D is a decomposition tree, therefore we also have $w \in X_i$. Since $\text{tw}(G) = k$, at most $k+1$ different w's are possible, each of them having at most M neighbors, so the number of interface vertices $v \notin X_j$ is bounded by $M \cdot (k+1)$.

Overall, we have $c = k + 1 + M(k+1) = (M+1)(k+1)$, so T is indeed a union tree for G which completes the proof. ■

Theorem 5.4

Let G be a graph and suppose its maximum degree is bounded by some constant $M \in \mathbb{N}$.

Then G is a union graph if and only if the treewidth of G is bounded by a constant.

Proof : The reverse implication follows from Theorem 5.3, so we only proof the following: given a union graph G with interface c, there exists a constant $k \in \mathbb{N}$ such that $\text{tw}(G) \leq k$.

W.l.o.g. we assume G to be connected. Using the union tree T we construct a decomposition tree D.

For every vertex V_i in T we define $X_i \subseteq V(G)$ to contain all the interface vertices of V_i's children (at most c) and all of V_i's interface vertices (at most c). Therefore $|X_i| \leq 2c$. We connect X_i to X_j in D, when V_i and V_j are connected in T.

It remains to show that D is a decomposition tree. Condition (iii) can be seen as follows: consider some edge $(v, w) \in E(G)$. Starting from the leaves $\{v\}$ and $\{w\}$, v and w are propagated upwards by construction until they lose their interface status, i.e. they encountered all of their neighbors. Particularly, v and w are both contained in their lowest common ancestor. We remark, that on its way from leaf to root, a vertex v can lose the status of an interface vertex, but that status can never be regained. One of the consequences of this fact is condition (iv), because once a vertex v loses interface status on its way towards the root, v never appears in any other X_i. So D is indeed a valid decomposition tree of G and $\text{width}(D) \leq k := 2c - 1$. It follows that $\text{tw}(G) \leq k$. ■

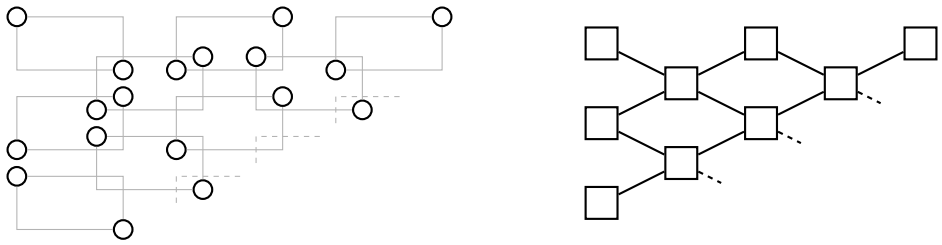

Figure 5.1: instance with boxes and grid intersection graph, maximum degree $M = 4$

5.3 Remarks

It is not possible to drop either one of the two requirements of Theorem 5.3. The intersection graph G in Figure 5.1 is a grid graph, each vertex has at most 4 neighbors. However, there is no constant k bounding the treewidth of G, this follows from Lemma 88 in [Bod98]. If we apply it to this $\sqrt{n} \times \sqrt{n}$ grid graph G, it states that $\mathrm{tw}(G) \geq \sqrt{n}$. Likewise, we can easily construct an intersection graph G in such a way that $\mathrm{tw}(G) = 1$, yet the maximum degree is $n - 1$, see Figure 5.2 for an example.

This clearly shows that none of the requirements implies the other. Furthermore, it was vital to have an upper bound for the degree in (3.1). Recall that the upper bound for the runtime was a consequence of the minimum in (3.1) being taken over a polynomial number of Hanan line sets. We used the same idea for the general case in (3.3) and (4.1), so all of our dynamic programs require the maximum degree of the intersection graph to be bounded. Finally, the upper bound on the treewidth is essential as well. This is a consequence of Theorem 5.4 since the most general dynamic program of this thesis is the one for union graphs. Intuitively, the decomposition tree provides the structure we need for the dynamic program.

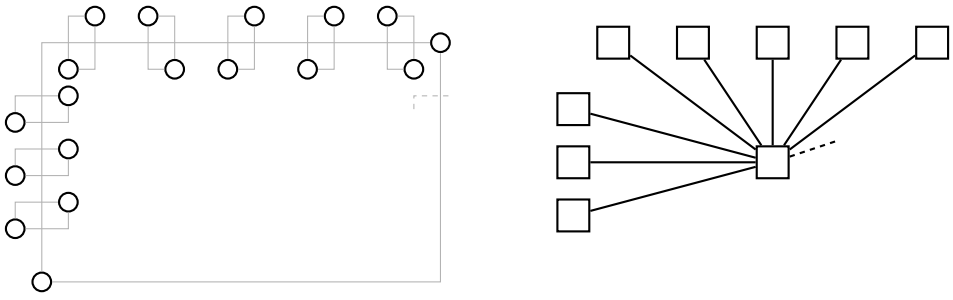

Figure 5.2: instance with boxes and tree intersection graph, treewidth $k = 1$

6 Implemented Algorithm

In this chapter we desribe a greedy algorithm for 2-GMMN. Unlike the dynamic programs we presented in the previous chapters, the algorithm will not always find a minimal network, but (in contrast to the dynamic programs) will actually be applicable in practice.

6.1 Greedy Algorithm

We only present the algorithm in pseudo code. More detail is provided below.

Algorithm 1 (Greedy Algorithm)

```
sort R
for all e ∈ ℋ(R) do
  w_e ← ℓ(e) / n_e
end for

N ← ∅
for all (p, q) ∈ R do
  find shortest M-path π_{p,q} from p to q (w.r.t. w)
  N ← N ∪ {π_{p,q}}
  for all e ∈ π_{p,q} do
    w_e ← 0
  end for
end for
return N
```

Like before, R denotes the problem instance containing pairs of terminals. First, the set R is sorted. Multiple sortings were implemented for the terminal pairs: from left to right (or vice versa), by L^1 distance, by area, in the order of input or random. Others can be added easily. The choice of sorting will typically affect the length of the network N.

Next, each Hanan edge e in $\mathcal{H}(R)$ is assigned a weight, based on its length and the number of bounding boxes containing e. In this algorithm we consider Hanan edges (see Definition 2.3), not complete Hanan lines (as we did in Chapters 3 and 4). Edges contained in many boxes are given a lower weight, as they are more likely to be useful for a short network.

For each pair of terminals (p, q) we apply a shortest path algorithm using these weights instead of the original edge lengths. We then add this M-path $\pi_{p,q}$ to the current network. The weights of the added edges are set to 0 so all the following shortest path computations can use these edges 'for free'. By construction, the output network N is always a valid Manhattan network as it contains an M-path for each pair $(p, q) \in R$.

Note that there exists a topological ordering on the edges, so the shortest path algorithm runs in $\mathcal{O}(n^2)$ since the overall number of Hanan Edges is quadratic (see Lemma 2.5). Altogether we end up with a runtime in $\mathcal{O}(n^3)$.

However, even if the runtime is easy to determine, the quality of the solution is not. There is a trivial upper bound on the length of the network N. Let (p^*, q^*) be the box with maximal L^1 distance. Then

$$\ell(N) \leq \sum_{i=1}^{n} d_1(p_i, q_i) \leq n \cdot d_1(p^*, q^*) \leq n \cdot \ell(N_{\text{opt}}), \tag{6.1}$$

where N_{opt} denotes a minimal network. Even an algorithm completely ignoring the overlap of the bounding boxes would easily match this upper bound by just connecting each pair of terminals by any M-path. Since GMMN is NP-hard, there is no efficient way of computing $\ell(N_{\text{opt}})$ (unless P = NP), so we cannot compare our solution to the optimum. Instead, we want to give an overview over the input sortings we used and their influence on the length of the output network N.

6.2 Input Sortings

The following sortings were implemented:

- $L^1_\uparrow$: sort pairs by $d_1(p, q)$ in ascending order
- $L^1_\downarrow$: sort pairs by $d_1(p, q)$ in descending order
- $A_\uparrow$: sort pairs by area of the bounding box in ascending order
- $A_\downarrow$: sort pairs by area of the bounding box in descending order

Furthermore, we added a random sorting for comparison and a sorting based on the order the terminal pairs are inserted. This allows the user to manually specify the order. These two will be ignored throughout the following analysis.

The concept of both of the ascending sortings $L^1_\uparrow$ and $A_\uparrow$ is based on the scenario shown in Figure 6.1. For this given instance, the sortings by area and L^1 distance are equivalent. If we stretch the instance vertically, we can ignore the horizontal edges and just compare the numbers of 'long' vertical edges. Then the upper solution is approximately 1.5 times longer than the bottom one.

Intuitively, it is much easier for a large box to adapt its M-path to some already existing small M-path along the way than vice versa.

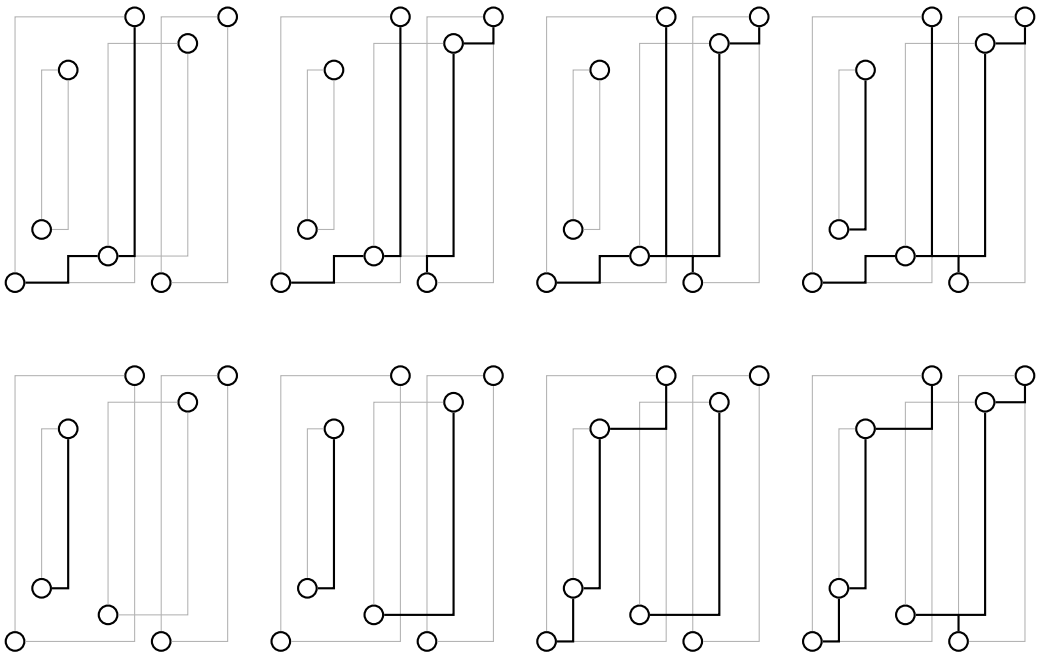

Figure 6.1: top: step-by-step view of the algorithm using sorting $L^1_\downarrow$ or $A_\downarrow$; bottom: step-by-step view of the algorithm using sorting $L^1_\uparrow$ or $A_\uparrow$

However, sorting the instance in ascending order (by area or L^1 distance) does not always yield a minimal network either. For example, consider Figure 6.2.

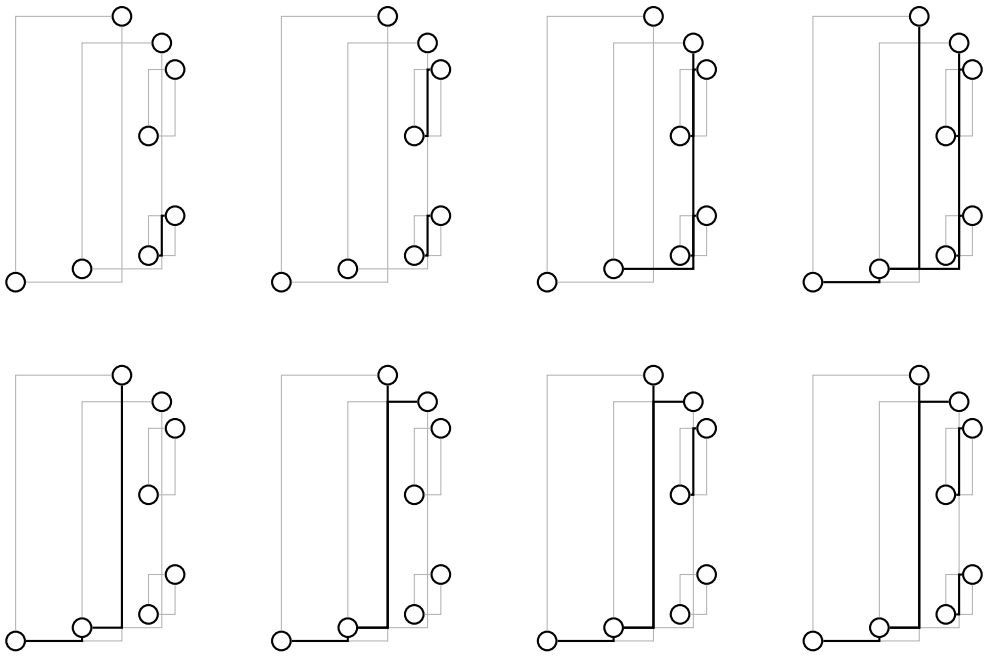

Figure 6.2: top: step-by-step view of the algorithm using sorting $L^1_\uparrow$ or $A_\uparrow$; bottom: step-by-step view of the algorithm using sorting $L^1_\downarrow$ or $A_\downarrow$

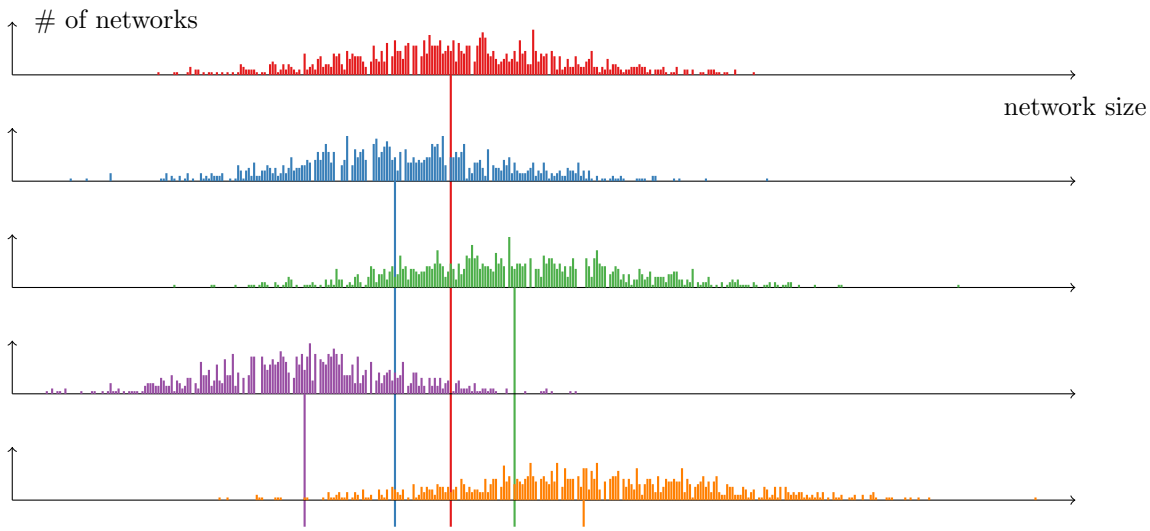

Figure 6.3: histogram for 1000 instances of size $n = 100$ for different input sortings; red: random, blue: $L^1_\uparrow$, green: $L^1_\downarrow$, purple: $A_\uparrow$, orange: $A_\downarrow$

Again, sortings $L^1_\uparrow$ and $A_\uparrow$ are equivalent. If the instance is stretched horizontally, we only need to consider horizontal egdes. Suppose the two small boxes have combined length $\frac{1}{2}\ell(a) + \varepsilon$, where a is the vertical segment shared by the two larger boxes and $\varepsilon > 0$. Hence, the upper solution (vertical length $\approx 2a$) is approximately $\frac{4}{3}$ times longer than the bottom one (vertical length $\approx 1.5a$).

So none of the sortings can guarantee to always find an optimal solution. But there is an advantage to the sorting $A_\uparrow$ in comparison to $L^1_\uparrow$. Suppose there is a vertex pair (p, q) with $\pi_x(p) = \pi_x(q)$, i.e. p and q share the same x-coordinate. Then p and q must be connected by a straight vertical line in every valid Manhattan network. In this case, chosing (p, q) first provides some edges with weight 0, which would have been added later anyways. Adding them early makes it possible to use these edges for the rest of the algorithm and therefore to lower the length of the output network.

Figure 6.3 shows a histogram of 1000 instances. We added $n = 100$ pairs of terminals at random (independent identically distributed) into a 500×500 integer grid. Then, all five sorting algorithms were tested on these instances and the resulting networks were sorted by length. The histogram also shows the median, plotted as a long vertical line. While the sorting $L^1_\uparrow$ produces significantly shorter networks than the random sorting, the sorting $A_\uparrow$ performs even better. The histograms for $n = 20$ and $n = 50$ show the same overall behavior. Overall, the sortings $A_\downarrow$ and $L^1_\downarrow$ are least sucessful. We did not provide any numbers for the histogram, as it is only intended to show the differences between the input sortings. To get an idea of the quality of the solutions, we used a tool martin Seybold (FMI, University of Stuttgart) provided to calculate lower bounds for those 1000 instances. For the calculation of these lower bounds see [FS14].

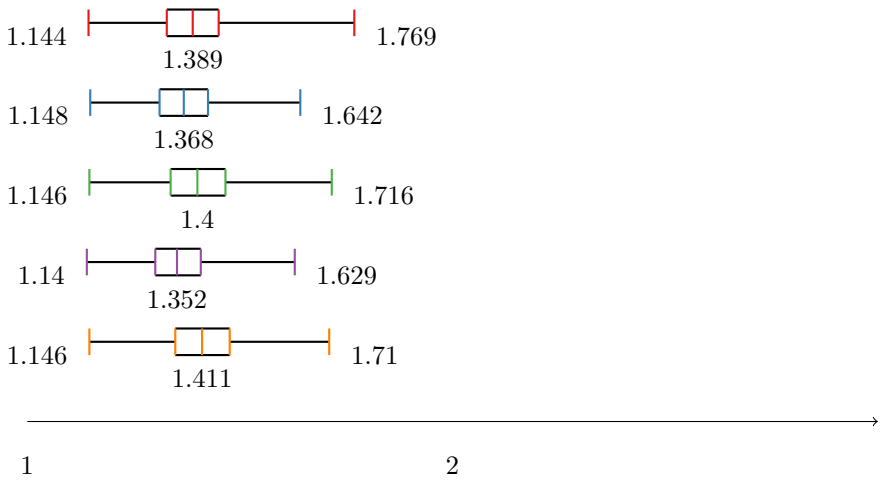

Figure 6.4: box plot of approximation factors for 1000 instances of size $n = 20$ for different input sortings; red: random, blue: $L^1_\uparrow$, green: $L^1_\downarrow$, purple: $A_\uparrow$, orange: $A_\downarrow$

Since we only have lower bounds for the length of a minimal network, we get upper bounds for the approximation factors. The box plots for these approximation factors are shown in Figure 6.4, 6.5 and 6.6 for $n = 20$, $n = 50$ and $n = 100$ respectively. They feature quantiles at 0%, 25%, 50% (median), 75% and 100%.

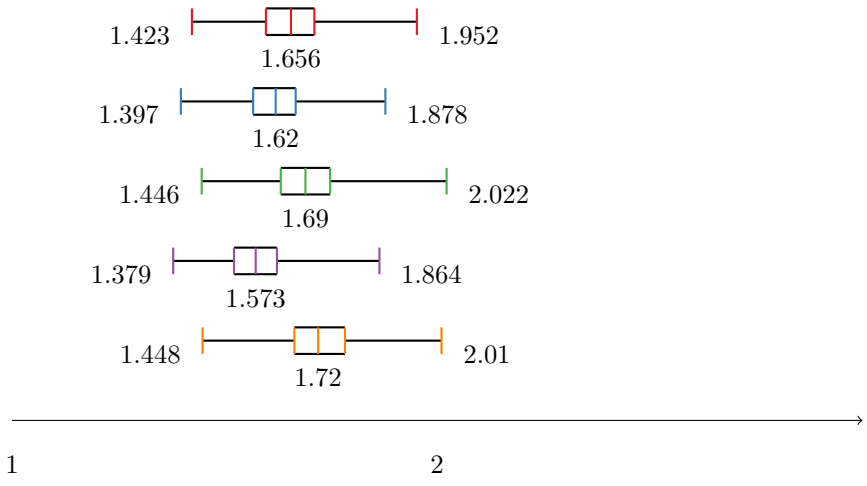

Figure 6.5: box plot of approximation factors for 1000 instances of size $n = 50$ for different input sortings; red: random, blue: $L^1_\uparrow$, green: $L^1_\downarrow$, purple: $A_\uparrow$, orange: $A_\downarrow$

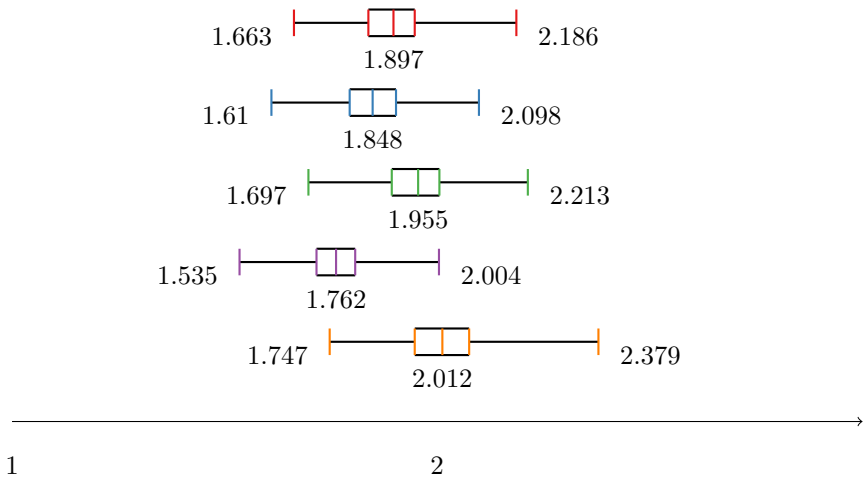

Figure 6.6: box plot of approximation factors for 1000 instances of size $n = 100$ for different input sortings; red: random, blue: $L^1_\uparrow$, green: $L^1_\downarrow$, purple: $A_\uparrow$, orange: $A_\downarrow$

We end this chapter with a short example, where the difference between sorting in ascending order ($L^1_\uparrow$,$A_\uparrow$) and sorting in descending order ($L^1_\downarrow$,$A_\downarrow$) is linear in n.

Consider the instance shown in Figure 6.7: a symmetric GMMN instance, each pair of terminals contains one point in the lower left and the point directly opposite. Here, the sortings $L^1_\uparrow$ and, equivalently, $A_\uparrow$ produce an output network of constant size $8a$, where a denotes half the side length of the bounding square, independent of n. On the other hand, the sortings $L^1_\downarrow$ and, equivalently, $A_\downarrow$ produce a network of size $4a + \frac{n-2}{2}a$, which is linear in n. In (6.1) we already mentioned that the factor by which such networks differ is at most $\mathcal{O}(n)$, so this is one of the biggest differences we might suspect between $L^1_\uparrow$ and $L^1_\downarrow$ for this greedy algorithm.

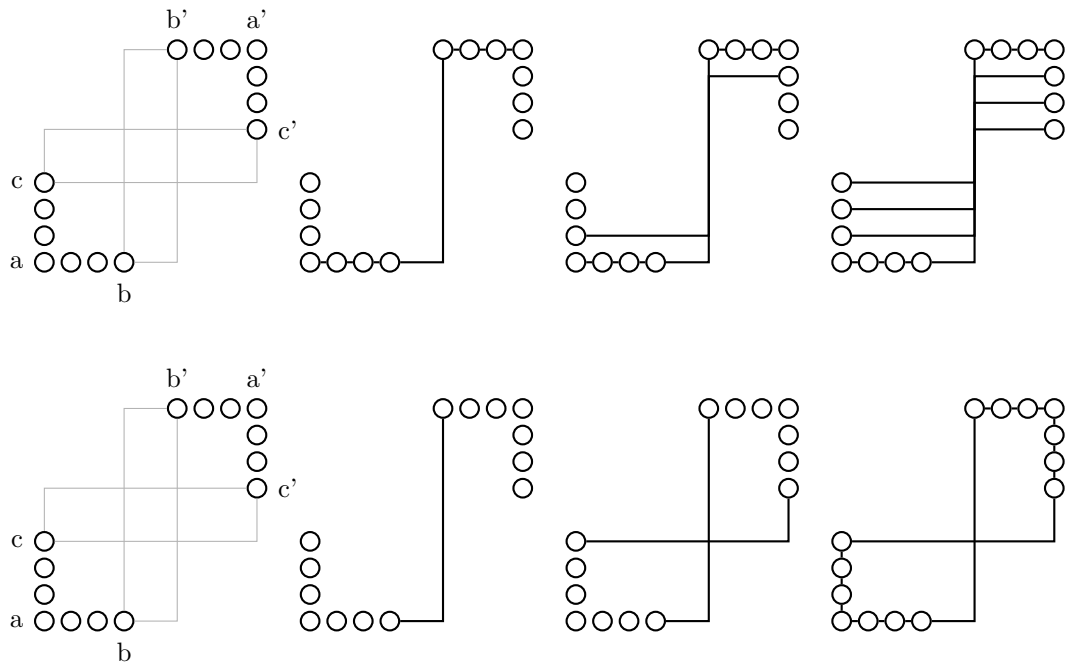

Figure 6.7: top: selected steps of the algorithm using sorting $L^1_{\downarrow}$ or $A_{\downarrow}$; bottom: selected steps of the algorithm using sorting $L^1_{\uparrow}$ or $A_{\uparrow}$

7 Conclusion

We have successfully shown that a dynamic program can be used to compute optimal solutions for d-GMMN in polynomial time, given that the instance's intersection graph is a union graph with an upper bound for the maximum degree. Also, a proof has been given that such union graphs can be characterized in terms of treewidth. Graph classes with bounded treewidth include (but are not limited to):

- circles ($\text{tw} \leq 2$)
- trees ($\text{tw} \leq 1$)
- series-parallel graphs ($\text{tw} \leq 2$, c.f. [Bod98], Theorem 41)
- outerplanar graphs ($\text{tw} \leq 2$, c.f. [Bod88], Theorem 4.5)
- k-outerplanar graphs where k is fixed ($\text{tw} \leq 3k-1$, c.f. [Bod88], Theorem 4.5)

For more examples we recommend [Bod86].

Furthermore, we introduced and implemented a greedy algorithm for practical purposes with a runtime in $\mathcal{O}(n^3)$. The algorithm relies on a shortest path algorithm and processes the instance in a certain order. The choice of this initial sorting affects the length of the output network and both in theoretical reasoning and in heuristic testing the sorting by area of the bounding box performed best.

Bibliography

[Bod86] Hans L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report RUU-CS-86-22, Department of Information and Computing Sciences, Utrecht University, 1986.

[Bod88] Hans L. Bodlaender. Planar graphs with bounded treewidth. Technical Report RUU-CS-88-14, 1988.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209, 1998.

[BWW04] Marc Benkert, Alexander Wolff, and Florian Widmann. The minimum Manhattan network problem: A fast factor-3 approximation. In *Discrete and Computational Geometry, Japanese Conference, JCDCG 2004*, volume 3742 of *Lecture Notes in Computer Science*. Springer, 2004.

[BWWS06] Marc Benkert, Alexander Wolff, Florian Widmann, and Takeshi Shirabe. The minimum Manhattan network problem: Approximations and exact solutions. *Comput. Geom.*, 35, 2006.

[CGS11] Francis Y. L. Chin, Zeyo Guo, and He Sun. Minimum Manhattan network is NP-complete. *Discrete and Computational Geometry*, 45, 2011.

[CNV08] Victor Chepoi, Karim Nouioua, and Yann Vaxès. A rounding algorithm for approximating minimum Manhattan networks. *Theoretical Computer Science*, 390, 2008.

[DFK+13] Aparna Das, Krzysztof Fleszar, Stephen G. Kobourov, Joachim Spoerhase, Sankar Veeramoni, and Alexander Wolff. Approximating the generalized minimum Manhattan network problem. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*. Springer, 2013.

[DGK+11] Aparna Das, Emden R. Gansner, Michael Kaufmann, Stephen G. Kobourov, Joachim Spoerhase, and Alexander Wolff. Approximating minimum Manhattan networks in higher dimensions. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*. Springer, 2011.

[DGK+15] Aparna Das, Emden R. Gansner, Michael Kaufmann, Stephen G. Kobourov, Joachim Spoerhase, and Alexander Wolff. Approximating

minimum Manhattan networks in higher dimensions. *Algorithmica*, 71, 2015.

[Die12] Reinhard Diestel. *Graph Theory*. Springer-Verlag, 2012.

[FS08] Bernhard Fuchs and Anna Schulze. A simple 3-approximation of minimum Manhattan networks. In *Seventh Cologne Twente Workshop on Graphs and Combinatorial Optimization, gargano, Italy, 13-15 May, 2008*. University of Milan, 2008.

[FS14] Stefan Funke and Martin P. Seybold. The generalized minimum Manhattan network problem (GMMN) - scale-diversity-aware approximation and a primal-dual algorithm. In *The Canadian Conference on Computational Geometry*, volume 26, 2014.

[GLN01] Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Approximating a minimum Manhattan network. *Nordic Journal of Computing*, 8, 2001.

[GSZ08a] Zeyu Guo, He Sun, and Hong Zhu. A fast 2-approximation algorithm for the minimum Manhattan network problem. In *Algorithmic Aspects in Information and Management, 4th International Conference, AAIM 2008, Shanghai, China, June 23-25, 2008. Proceedings*, volume 5034 of *Lecture Notes in Computer Science*. Springer, 2008.

[GSZ08b] Zeyu Guo, He Sun, and Hong Zhu. Greedy construction of 2-approximation minimum Manhattan network. In *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*. Springer, 2008.

[Rob69] Fred S. Roberts. On the boxicity and cubicity of a graph. *Recent Progress in Combinatorics*, 1969.

[SS05] Weiping Shi and Chen Su. The rectilinear steiner arborescence problem is NP-complete. *SIAM J. Comput.*, 35, 2005.

Declaration

I hereby declare that I, Michael Schnizler, wrote this thesis myself, that I did not use any source outside of the bibliography and that every adoped statement has been noted as such. This thesis has never been presented (in whole or in part) to any other examination committee. Furthermore, I declare that the electronic copy matches the printed copies.

Erklärung

Ich versichere hiermit, dass ich, Michael Schnizler, die vorliegende Arbeit selbst verfasst, keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe. Diese Arbeit ist weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen. Weiterhin versichere ich, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

(Michael Schnizler, Stuttgart, May 4$^{\text{th}}$, 2015)

I want to thank Volker Diekert for supervising this thesis and Stefan Funke for selecting this topic and his help over the last months. Also, their lectures helped a lot. Special thanks goes to Martin Seybold, who really took a lot of time helping me and provided the lower bounds for the results in Chapter 6. Big thanks to Maria Wiebe for many useful hints and to everyone who supported me during my studies. Especially I want to thank my good friend Hansjörg Schmauder for supporting me and helping me with everything concerning code and implementation!