The Minimum Manhattan Network Problem: A Fast Factor-3 Approximation[*]

Marc Benkert[†] Alexander Wolff[†] Florian Widmann[‡]

Technical Report 2004-16
Fakultät für Informatik, Universität Karlsruhe

Abstract

Given a set of nodes in the plane and a constant $t \geq 1$, a Euclidean *t-spanner* is a network in which, for any pair of nodes, the ratio of the network distance and the Euclidean distance of the two nodes is at most t. Such networks have applications in transportation or communication network design and have been studied extensively.

In this paper we study 1-spanners under the Manhattan (or L_1-) metric. Such networks are called *Manhattan networks*. A Manhattan network for a set of nodes can be seen as a set of axis-parallel line segments whose union contains a Manhattan path for each pair of nodes. It is not known whether it is NP-hard to compute minimum Manhattan networks (MMN), i.e. Manhattan networks of minimum total length. In this paper we present a factor-3 approximation algorithm for this problem. Given a set P of n nodes, our algorithm computes in $O(n \log n)$ time and linear space a Manhattan network for P whose length is at most 3 times the length of an MMN of P. We have implemented our algorithm and have done a thorough experimental analysis.

1 Introduction

For many applications it is desirable to connect the nodes of a transportation or communication network by short paths within the network. In the Euclidean plane this can be achieved by connecting all pairs of nodes by straight line segments. While the complete graph minimizes node-to-node travel time, it maximizes the network-construction costs. An interesting alternative are Euclidean *t-spanners*, i.e. networks in which the ratio of the network distance and the Euclidean distance between any pair of nodes is bounded by a constant $t \geq 1$. Euclidean spanners were introduced by Chew [Che89] and have since been studied extensively—see for instance the survey by Eppstein [Epp00]. Researchers have tried to construct spanners with other desirable properties, such as small node degree, small total edge length, and small diameter. Spanners with one

[*]This work was supported by grant WO 758/4-1 of the German Science Foundation (DFG).

[†]Faculty of Computer Science, Karlsruhe University, P.O. Box 6980, D-76128 Karlsruhe, Germany. WWW: `http://i11www.ira.uka.de/algo/group`

[‡]Email: `ui72@stud.uni-karlsruhe.de`

or more of these properties can be constructed in $O(n \log n)$ time [ADM$^+$95], where n is the number of nodes.

Under the Euclidean metric, in a 1-spanner (which is the complete graph) the location of each edge is uniquely determined. This is not the case in the Manhattan (or L_1-) metric, where an edge $\{p, q\}$ of a 1-spanner is a Manhattan p–q path, i.e. a staircase path between p and q. A 1-spanner under the Manhattan metric for a finite point set $P \subset \mathbb{R}^2$ is called a *Manhattan network* and can be seen as a set of axis-parallel line segments whose union contains a Manhattan p–q path for each pair $\{p, q\} \in \binom{P}{2}$.

In this paper we investigate how the extra degree of freedom in routing edges can be used to construct Manhattan networks of minimum total length, so-called *minimum Manhattan networks* (MMN). The MMN problem may have applications in city planning or VLSI layout, but Lam et al. [LAP03] also describe an application in computational biology. For aligning gene sequences they propose a three-step approach. In the first step, they use a local-alignment algorithm like BLAST [AGM$^+$90] to identify subsequences of high similarity, so-called *high-scoring pairs* (HSP). In the second step they compute a network for certain points given by the HSPs. They do not require that each point be connected by Manhattan paths to *all* other points, but only to those that have both larger x- and y-coordinates. A Manhattan path in their setting corresponds to a sequence of insertions, deletions, and (mis)matches that are needed to transform one point representing a gene sequence into another. Lam et al. show that modifying an algorithm by Gudmundsson et al. [GLN01] yields a $O(n^3)$-time factor-2 approximation for their problem. They state that the restriction to the network they compute helps to considerably reduce the size of the search space for a good alignment, which is computed by dynamic programming in the third step of their approach.

1.1 Previous work

The MMN problem has been considered before, but until now, its complexity status is unknown. Gudmundsson et al. [GLN01] have proposed an $O(n \log n)$-time factor-8 and an $O(n^3)$-time factor-4 approximation algorithm. Later Kato et al. [KIA02] have given an $O(n^3)$-time factor-2 approximation algorithm. However, the correctness proof of Kato et al. is incomplete. Both the factor-4 approximation and the algorithm by Kato et al. use quadratic space. We now briefly sketch all three algorithms.

Gudmundsson et al. [GLN01] considered each input point p separately. From p they established Manhattan paths to those points p' where the bounding box of p and p' contained no other input point. This yields a Manhattan network. In order to establish the paths from p to all points p', they considered the points p' in each of the four quadrants relative to p simultaneously. In each of the quadrants, these points define a staircase polygon. The points p' are connected to p by rectangulating the staircase polygon, minimizing the length of the segments used for the rectangulation. Solving this subproblem by a factor-2 approximation algorithm yields the factor-8 approximation algorithm for the MMN problem while using dynamic programming to solve the subproblem optimally yields the factor-4 approximation.

Kato et al. [KIA02] observed that it is not always necessary to connect p explicitly to all points p'. Instead, they came up with the notion of a *generating* *set*, i.e. a set of pairs of points with the property that each network that contains Manhattan paths between these point pairs is already a Manhattan network. In a first step they constructed a network N' whose length is bounded from above by the length of an MMN. Kato et al. designed the network N' such that it contains Manhattan paths for as many point pairs in the generating set as possible. They claimed that in a second step, they could rectangulate the facets of N' such that the remaining unconnected point pairs are connected and the total length of the new segments is again bounded from above by the length of an MMN. Both the details of this step and the proof of its correctness are missing in [KIA02].

1.2 Our results

In this paper we present an $O(n \log n)$-time factor-3 approximation algorithm. We use the generating set of [KIA02], and we also split the generating set into two subsets for which we incrementally establish Manhattan paths. However, our algorithm is simpler, faster and uses only linear (instead of quadratic) storage. The main novelty of our approach is that we partition the plane into two regions and compare the network computed by our algorithm to an MMN in each region separately. One region of the partition is given by the union of staircase polygons that have to be pseudo-rectangulated. For this subproblem a factor-2 approximation suffices. It runs in $O(n \log n)$ time and is similar to the factor-2 approximation for rectangulating staircase polygons that Gudmundsson et al. [GLN01] proposed.

We implemented our factor-3 approximation algorithm and measured its performance on random point sets using an exact solver based on a mixed integer formulation for the MMN problem [WBS04]. Further, we make an extensive comparison with other algorithms including the factor-4 and -8 approximations of Gudmundsson et al. [GLN01]. It turns out that our algorithm usually finds Manhattan networks that are at most 50% longer than the corresponding MMN. However, for any $\varepsilon > 0$ there is a point set for which our algorithm returns a Manhattan network that is $(3 - \varepsilon)$ times as long as the corresponding MMN.

This paper is structured as follows. In Section 2 and 3 we give some basic definitions and show how helpful information for our network is computed. In Section 4 we detail how the backbone of our network is computed. We describe the algorithm precisely in Section 5 and analyze its approximation factor in Section 6. The practical performance of the algorithm is evaluated in Section 7. We conclude with some open problems in Section 8.

We have made our algorithm available via a Java applet under the URL *http://i11www.ira.uka.de/manhattan*. The applet also features the factor-4 and factor-8 approximation algorithms by Gudmundsson et al. [GLN01].

2 Basic definitions

We use $|M|$ to denote the total length of a set M of line segments. For all such sets M we assume throughout the paper that each segment of M is inclusion-maximal with respect to $\bigcup M$. It is not hard to see that for every Manhattan network M there is a Manhattan network M' with $|M'| \leq |M|$ that is contained in the grid induced by the input points, i.e. M' is a subset of the union U of the horizontal and vertical lines through the input points. Therefore we will only consider networks contained in U. It is clear that any meaningful Manhattan network of a point set P is contained in the bounding box $\mathrm{BBox}(P)$ of P. Finding a Manhattan network for given P is rather easy, e.g. the parts of U within $\mathrm{BBox}(P)$ yield a Manhattan network. However, the point set $\{(1,1),\ldots,(n,n)\}$ shows that this network is not always a good approximation, in this case it is n times longer than an MMN.

We will use the notion of a *generating set* that has been introduced in [KIA02]. A generating set is a subset of $\binom{P}{2}$ with the property that a network containing Manhattan paths for all pairs in the subset is already a Manhattan network of P.

The authors of [KIA02] defined a generating set Z with the nice property that Z consists only of a linear number of point pairs. We use the same generating set Z, but more intuitive names for the subsets of Z. We define Z to be the union of three subsets $Z_{\mathrm{hor}}, Z_{\mathrm{ver}}$ and Z_{quad}. These subsets are defined below. Our algorithm will establish Manhattan paths for all point pairs of Z—first for those in $Z_{\mathrm{hor}} \cup Z_{\mathrm{ver}}$ and then for those in Z_{quad}.

Definition 1 (Z_{ver}) *Let $P = \{p_1,\ldots,p_n\}$ be the set of input points in lexicographical order, where $p_i = (x_i, y_i)$. Let $x^1 < \cdots < x^u$ be the sequence of x-coordinates of the points in P in ascending order. For $i = 1,\ldots,u$ let $P^i = \{p_{a(i)}, p_{a(i)+1},\ldots,p_{b(i)}\}$ be the set of all $p \in P$ with x-coordinate x^i. Then*

$$\begin{aligned} Z_{\mathrm{ver}} = & \quad \{(p_i, p_{i+1}) \mid x_i = x_{i+1} \text{ and } 1 \le i < n\} \\ & \cup \ \{(p_{a(i)}, p_{b(i+1)}) \mid y_{a(i)} > y_{b(i+1)} \text{ and } 1 \le i < u\} \\ & \cup \ \{(p_{b(i)}, p_{a(i+1)}) \mid y_{b(i)} < y_{a(i+1)} \text{ and } 1 \le i < u\}. \end{aligned}$$

See Figure 3, where all pairs of Z_{ver} are connected by an edge. Note that Z_{ver} consists of at most $n-1$ point pairs. If no points have the same x-coordinate, it holds that $Z_{\mathrm{ver}} = \{(p_i, p_{i+1}) \mid 1 \le i < n\}$, i.e. Z_{ver} is the set of neighboring pairs in the lexicographical order. The definition of Z_{hor} is analogous to that of Z_{ver} with the roles of x and y exchanged. Figure 4 shows that $Z_{\mathrm{hor}} \cup Z_{\mathrm{ver}}$ is not necessarily a generating set: Since $(p,h) \in Z_{\mathrm{hor}}$ and $(p,v) \in Z_{\mathrm{ver}}$, no network that consists only of Manhattan paths between pairs in $Z_{\mathrm{hor}} \cup Z_{\mathrm{ver}}$ contains a Manhattan p–q path. This shows the necessity of a third subset Z_{quad} of Z.

Definition 2 (Z_{quad}) *For a point $r \in \mathbb{R}^2$ denote its Cartesian coordinates by (x_r, y_r). Let $Q(r,1) = \{s \in \mathbb{R}^2 \mid x_r \le x_s \text{ and } y_r \le y_s\}$ be the first quadrant of the Cartesian coordinate system with origin r. Define $Q(r,2)$, $Q(r,3)$, $Q(r,4)$ analogously and in the usual order. Then Z_{quad} is the set of all ordered pairs (p,q) with $p,q \in P$ and $q \in Q(p,t)$ for some $t \in \{1,2,3,4\}$ that fulfill*

(a) q is the point that has minimum y-distance from p among all points in $Q(p,t) \cap P$ with minimum x-distance from p, and

(b) there is no $q' \in Q(p,t) \cap P$ with $\{p,q'\} \in Z_{\mathrm{hor}} \cup Z_{\mathrm{ver}}$.

Obviously Z_{quad} consists of at most $4n$ point pairs. For the proof that Z_{quad} is in fact sufficient for $Z = Z_{\mathrm{ver}} \cup Z_{\mathrm{hor}} \cup Z_{\mathrm{quad}}$ to be a generating set, see [KIA02].

For our analysis we need the following areas of the plane. Let $\mathcal{R}_{\mathrm{hor}} = \{\mathrm{BBox}(p,q) \mid \{p,q\} \in Z_{\mathrm{hor}}\}$, where $\mathrm{BBox}(p,q)$ is the smallest axis-parallel closed rectangle that contains p and q. Note that $\mathrm{BBox}(p,q)$ is just the line segment

Seg$[p, q]$ from p to q, if p and q lie on the same horizontal or vertical line. In this case we call BBox(p, q) a *degenerate rectangle*. Define $\mathcal{R}_{\text{ver}}$ and $\mathcal{R}_{\text{quad}}$ analogously. Let $\mathcal{A}_{\text{hor}}$, $\mathcal{A}_{\text{ver}}$, and $\mathcal{A}_{\text{quad}}$ be the subsets of the plane that are defined by the union of the rectangles in $\mathcal{R}_{\text{hor}}$, $\mathcal{R}_{\text{ver}}$, and $\mathcal{R}_{\text{quad}}$, respectively.

Any Manhattan network has to bridge the vertical (horizontal) gap between the points of each pair in $Z_{\text{ver}}(Z_{\text{hor}})$. Of course this can be done such that at the same time the gaps of adjacent pairs are (partly) bridged. The corresponding minimization problem is defined as follows:

Definition 3 (cover [KIA02]) *A set of vertical line segments $\mathcal{V}$ is a* cover *of (or* covers*) $\mathcal{R}_{\text{ver}}$, if any $R \in \mathcal{R}_{\text{ver}}$ is covered, i.e. for any horizontal line ℓ with $R \cap \ell \neq \emptyset$ there is a $V \in \mathcal{V}$ with $V \cap \ell \cap R \neq \emptyset$. We say that $\mathcal{V}$ is a* minimum vertical cover (MVC) *if $\mathcal{V}$ has minimum length among all covers of $\mathcal{R}_{\text{ver}}$. The definition of a* minimum horizontal cover (MHC) *is analogous.*

Figure 5 shows an example of an MVC. Since any MMN covers $\mathcal{R}_{\text{ver}}$ and $\mathcal{R}_{\text{hor}}$, Kato et al. have the following result.

Lemma 1 ([KIA02]) *The union of an* MVC *and an* MHC *has length bounded by the length of an* MMN.

To sketch our algorithm we need the following notations. Let N be a set of line segments. We say that N *satisfies* a set of point pairs $S \subseteq \binom{P}{2}$ if N contains a Manhattan p–q path for each $\{p, q\} \in S$. We use $\bigcup N$ to denote the corresponding set of points, i.e. the union of the line segments in N. Let ∂M be the boundary of a set $M \subseteq \mathbb{R}^2$.

Our algorithm will proceed in four phases. In phase 0, we compute Z. In phase I, we construct a network N_1 that contains the union of a special MVC and a special MHC and satisfies $Z_{\text{ver}} \cup Z_{\text{hor}}$. In phase II, we identify a set $\mathcal{R}$ of open regions in $\mathcal{A}_{\text{quad}}$ that do not intersect N_1, but need to be bridged in order to satisfy Z_{quad}. The regions in $\mathcal{R}$ are staircase polygons. They give rise to two sets of segments, N_2 and N_3, which are needed to satisfy Z_{quad}. For each region $R \in \mathcal{R}$ we put the segments that form $\partial R \setminus \bigcup N_1$ into N_2, plus, if necessary, an extra segment to connect R to N_1. Finally, in phase III, we bridge the regions in $\mathcal{R}$ by computing a set N_3 of segments in the interior of the regions. This yields a Manhattan network $N = N_1 \cup N_2 \cup N_3$.

The novelty of our analysis is that we partition the plane into two areas and compare N to an MMN in each area separately. The area $\mathcal{A}_3$ consists of the interiors of the regions $R \in \mathcal{R}$ and contains N_3. The other area $\mathcal{A}_{12}$ is the complement of $\mathcal{A}_3$ and contains $N_1 \cup N_2$. For a fixed MMN N_{opt} we show that $|N \cap \mathcal{A}_{12}| \leq 3|N_{\text{opt}} \cap \mathcal{A}_{12}|$ and $|N \cap \mathcal{A}_3| \leq 2|N_{\text{opt}} \cap \mathcal{A}_3|$ and thus $|N| \leq 3|N_{\text{opt}}|$. The details will be given in Section 5.

3 Neighbors and the generating set

We now define vertical and horizontal neighbors of points in P. Knowing these neighbors helps to compute Z and $\mathcal{R}$.

Definition 4 (neighbors) *For a point $p \in P$ and $t \in \{1, 2, 3, 4\}$ let p.*xbnor$[t]$ = nil *if $Q(p, t) \cap P = \{p\}$. Otherwise p.*xbnor$[t]$ *points at the point that has minimum y-distance from p among all points in $Q(p, t) \cap P \setminus \{p\}$ with minimum*

x-distance from p. The pointer $p.\text{ynbor}[t]$ is defined by exchanging x and y in the above definition.

All pointers of types xnbor and ynbor can be computed by a simple plane sweep in $O(n \log n)$ time. The set Z_{ver} is then determined by passing the points in lexicographical order and examining the pointers of type xnbor. This works analogously for Z_{hor}. Note that by Definition 1 each point $q \in P$ is incident to at most three rectangles of $\mathcal{R}_{\text{ver}}$, at most two of which can be (non-) degenerate. We refer to points $p \in P$ with $(p, q) \in Z_{\text{ver}}$ as *vertical predecessors* of q and to points $r \in P$ with $(q, r) \in Z_{\text{ver}}$ as *vertical successors* of q. We call a predecessor or successor of q *degenerate* if it has the same x-coordinate as q. Note that each point can have at most one degenerate vertical predecessor and successor, and at most one non-degenerate vertical predecessor and successor. Horizontal predecessors and successors are defined analogously with respect to Z_{hor}. For each $t \in \{1, 2, 3, 4\}$ the pair $(q, q.\text{xnbor}[t])$ lies in Z_{quad} if and only if $q.\text{xnbor}[t] \neq$ nil and no vertical or horizontal predecessor or successor lies in $Q(q, t)$. We conclude:

Lemma 2 *All pointers of type* xnbor *and* ynbor*, and the generating set Z can be computed in $O(n \log n)$ time.*

4 Minimum covers

In general the union of an MVC and an MHC does not satisfy $Z_{\text{ver}} \cup Z_{\text{hor}}$. Additional segments must be added to achieve this. To ensure that the total length of these segments can be bounded, we need covers with a special property. We say that a cover is *nice* if each cover segment contains an input point.

Lemma 3 *For any nice* MVC $\mathcal{V}$ *and any nice* MHC $\mathcal{H}$ *there is a set $\mathcal{S}$ of line segments such that $\mathcal{V} \cup \mathcal{H} \cup \mathcal{S}$ satisfies $Z_{\text{ver}} \cup Z_{\text{hor}}$ and $|\mathcal{S}| \leq W + H$, where W and H denote width and height of* $\text{BBox}(P)$*, respectively. We can compute the set $\mathcal{S}$ in linear time if for each $R \in \mathcal{R}_{\text{ver}}$ ($\mathcal{R}_{\text{hor}}$) we have constant-time access to the segments in $\mathcal{V}$ ($\mathcal{H}$) that intersect R.*

Proof. We show that there is a set $\mathcal{S}_{\mathcal{V}}$ of horizontal segments with $|\mathcal{S}_{\mathcal{V}}| \leq W$ such that $\mathcal{V} \cup \mathcal{S}_{\mathcal{V}}$ satisfies Z_{ver}. Analogously it can be shown that there is a set $\mathcal{S}_{\mathcal{H}}$ of vertical segments with $|\mathcal{S}_{\mathcal{H}}| \leq H$ such that $\mathcal{H} \cup \mathcal{S}_{\mathcal{H}}$ satisfies Z_{hor}. This proves the lemma.

Let $(p, q) \in Z_{\text{ver}}$. If $R = \text{BBox}(p, q)$ is degenerate, then by the definition of a cover, there is a line segment $s \in \mathcal{V}$ with $R \subseteq s$, and thus $\mathcal{V}$ satisfies (p, q).

Otherwise R defines a non-empty vertical open strip $\sigma(p, q)$ bounded by p and q. Note that by the definition of Z_{ver}, R is the only rectangle in $\mathcal{R}_{\text{ver}}$ that intersects $\sigma(p, q)$. This yields that the widths of $\sigma(p, q)$ over all $(p, q) \in Z_{\text{ver}}$ sum up to at most W. Thus we are done, if we can show that there is a horizontal line segment h such that the length of h equals the width of $\sigma(p, q)$ and $\mathcal{V} \cup \{h\}$ satisfies (p, q).

Now observe that no line segment in $\mathcal{V}$ intersects $\sigma(p, q)$ since $\mathcal{V}$ is nice and $\sigma(p, q) \cap P = \emptyset$. Hence, the segments of $\mathcal{V}$ that intersect R in fact intersect only the vertical edges of R. We assume w.l.o.g. that $x_p < x_q$ and $y_p < y_q$ (otherwise rename and/or mirror P at the x-axis). This means that due to the definition of Z_{ver}, there is no input point vertically above p. Thus, if there is a segment s_p in $\mathcal{V}$ that intersects the left edge of R, then s_p must contain p. Analogously, a segment s_q in $\mathcal{V}$ that intersects the right edge of R must contain q. Since $\mathcal{V}$ covers R, s_p or s_q must exist. Let ℓ be the horizontal through the topmost point of s_p or the bottommost point of s_q. Then $h = \ell \cap R$ does the job, again due to the fact that $\mathcal{V}$ covers R, see Figure 1. Clearly h can be determined in constant time. $\square$

In order to see that every point set has in fact a nice MVC, we need the following definitions. We restrict ourselves to the vertical case, the horizontal case is analogous.

For a horizontal line ℓ consider the graph $G_\ell(V_\ell, E_\ell)$, where V_ℓ is the intersection of ℓ with the vertical edges of rectangles in $\mathcal{R}_{\text{ver}}$, and there is an edge in E_ℓ if two intersection points belong to the same rectangle. We say that a point v in V_ℓ is *odd* if v is contained in a degenerate rectangle or if the number of points to the left of v that belong to the same connected component of G_ℓ is odd, otherwise we say that v is *even*. For a vertical line g let an *odd segment* be an inclusion-maximal connected set of odd points on g. Define *even segments* accordingly. For example, the segment s (drawn bold in Figure 6) is an even segment, while $f \setminus s$ is odd. We say that *parity changes* in points where two segments of different parity touch. We refer to these points as *points of changing parity*. The MVC with the desired property will simply be the set of all odd segments. The next lemma characterizes odd segments, especially item (v) prepares their computation. Strictly speaking we have to state whether the endpoints of each odd segment are odd too, but since a closed segment has same length as the corresponding open segment, we consider odd segments closed.

Lemma 4 *Let $g : x = x_g$ be a vertical line through some point $p = (x_p, y_p) \in P$.*

(i) Let e be a vertical edge of a rectangle $R \in \mathcal{R}_{\text{ver}}$. Then either all points on e are even or the only inclusion-maximal connected set of odd points on e contains an input point.

(ii) Let $R_1, \dots, R_d$ and $R'_1, \dots, R'_{d'}$ be the degenerate and non-degenerate rectangles in $\mathcal{R}_{\text{ver}}$ that g intersects, respectively. Then $d = |g \cap P| - 1$ and $d' \leq 2$. If $d = 0$ then $d' > 0$ and each R'_i has a corner in p. Else, if $d > 0$, there are $p_1, p_2 \in P$ such that $g \cap (R_1 \cup \dots \cup R_d) = \text{Seg}[p_1, p_2]$. Then each R'_i has a corner in either p_1 or p_2.

(iii) There are $b_g < t_g \in \mathbb{R}$ such that $g \cap \mathcal{A}_{\text{ver}} = \{x_g\} \times [b_g, t_g]$.

(iv) The line g contains at most two points of changing parity and at most one odd segment. For each point c of changing parity there is an input point with the same y-coordinate.

(v) If g has no point of changing parity, there is either no odd segment on g or the odd segment is $\{x_g\} \times [b_g, t_g]$. If g has one point c of changing parity, then either $\{x_g\} \times [b_g, y_c]$ or $\{x_g\} \times [y_c, t_g]$ is the odd segment. If g has two points c and c' of changing parity, then $\{x_g\} \times [y_c, y_{c'}]$ is the odd segment.

Proof. For (i) we assume without loss of generality that e is the right vertical edge of $R = \text{BBox}(p, q)$ and that q is the topmost point of e. If R is degenerate it is clear that all points on e (including p and q) are odd, and we are done. Thus we can assume that $x_p < x_q$. Let $p_0 = q, p_1 = p, p_2 \dots, p_k$ be the input points in order of decreasing x-coordinate that span the rectangles in $\mathcal{R}_{\text{ver}}$ that are relevant for the parity of e. Let $p_i = (x_i, y_i)$. For $2 \leq i \leq k$ define recursively $\overline{y_i} = \min\{y_i, \overline{y_{i-2}}\}$ if i is even, and $\overline{y_i} = \max\{y_i, \overline{y_{i-2}}\}$ if i is odd. Let $\overline{p_i} = (x_i, \overline{y_i})$, and let $\overline{\mathcal{L}}$ be the polygonal chain through $p_0, p_1, \overline{p_2}, \overline{p_3}, \dots, \overline{p_k}$, see Figure 6. Note that the parity of a point v on e is determined by the number of segments of $\overline{\mathcal{L}}$ that the horizontal h_v through v intersects. If h_v is below $\overline{p_k}$, then it intersects a descending segment for each ascending segment of $\overline{\mathcal{L}}$, hence v is even. If on the other hand h_v is above $\overline{p_k}$, then it intersects an ascending segment for each descending segment—plus $\overline{p_1 p_0}$, hence v is odd. In other words, if $\overline{y_k} = y_0$, all points of e are even, if $\overline{y_k} = y_1$, all points of e are odd, and otherwise parity changes only in $(x_0, \overline{y_k})$ and q is odd. This settles (i).

(ii) follows directly from the definition of Z_{ver}, and (iii) follows from (ii), see also Figure 2.

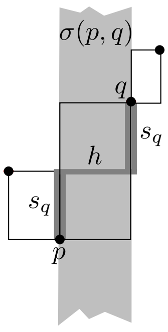


Figure 1: Illustration for Lemma 3.

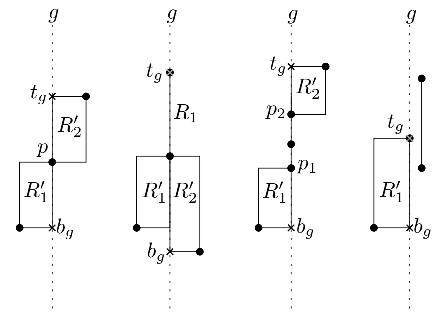


Figure 2: Illustration for Lemma 4 (ii) and Lemma 4 (iii).

For (iv) we first assume $d = 0$. Then (ii) yields $d' \in \{1, 2\}$ and $g \cap P = \{p\}$. By (i) we know that the only inclusion-maximal connected set of odd points on each vertical rectangle edge on g contains an input point, i.e. p. Thus there are at most two points of changing parity and there is at most one odd segment on g. Also according to the above proof of (ii), parity can change only in points of type $(x_0, \overline{y_k})$, and $\overline{y_k}$ is the y-coordinate of some input point in the set $\{p_0, \dots, p_k\}$.

Now if $d > 0$ note that all degenerate rectangles consist only of odd points. By (ii) we have that $g \cap (R_1 \cup \dots \cup R_d) = \text{Seg}[p_1, p_2]$ and that each of the at most two non-degenerate rectangles has a corner in either p_1 or p_2. Thus again the statement holds.

For the proof of (v) we make a case distinction depending on d'. If $d' = 0$, g intersects only degenerate rectangles and thus there is no point of changing parity on g and the odd segment is $\{x_g\} \times [b_g, t_g]$. Otherwise we assume w.l.o.g. that e is contained in g. If $e = \{x_g\} \times [b_g, t_g]$ holds, we are done. The argument of (i) shows that either e contains no point of changing parity and hence all points of e are of one parity, or $c = c_1$ is the only point of changing parity and the odd segment is $\{x_g\} \times [c_1, t_g = y_p]$. If $e \neq \{x_g\} \times [b_g, t_g]$, there is a further rectangle R_p in $\mathcal{R}_{\text{ver}}$ with $R_p = \text{BBox}(p, r)$ and $x_p \leq x_r, y_p < y_r$. If R_p is non-degenerate all points on $\{x_g\} \times [b_g, t_g] \setminus e$ are even, as there are no relevant rectangles to the right. In this case we have no odd segment on g if e is completely even, the odd segment is $\{x_g\} \times [b_g, y_p = c_1]$ if e is completely odd, and if c is a point of changing parity the odd segment is $\{x_g\} \times [c = c_1, y_p = c_2]$. If R_p is degenerate, $\{x_g\} \times [p, r]$ has to be added to the odd segments stated as before, besides the same argument holds with a possibly rectangle R_r connected to r. $\square$

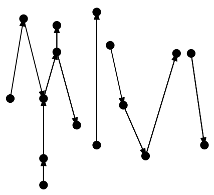


Figure 3: Point pairs in Z_{ver}.

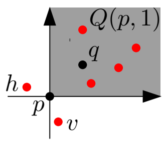


Figure 4: The pair (p,q) is in Z_{quad}.

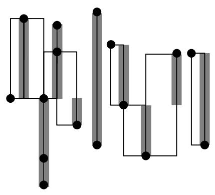


Figure 5: The odd MVC.

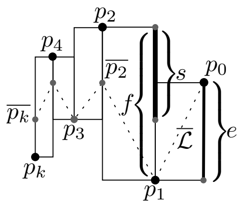


Figure 6: Proof of Lemma 4.

Lemma 5 *The set $\mathcal{V}$ of all odd segments is a nice* MVC*, the* odd MVC.

Proof. Clearly $\mathcal{V}$ covers $\mathcal{R}_{\text{ver}}$. Let ℓ be a horizontal line that intersects $\mathcal{A}_{\text{ver}}$. Consider a connected component C of G_ℓ and let k be the number of vertices in C. If k is even then any cover must contain at least $k/2$ vertices of C, and $\mathcal{V}$ contains exactly $k/2$. On the other hand, if $k > 1$ is odd then any cover must contain at least $(k-1)/2$ vertices of C, and $\mathcal{V}$ contains exactly $(k-1)/2$. If $k = 1$, any cover must contain the vertex, and so does $\mathcal{V}$ as the vertex belongs to a degenerate rectangle. Thus $\mathcal{V}$ is an MVC. Lemma 4 (i) shows that $\mathcal{V}$ is nice. $\square$

Lemma 6 *The odd* MVC *can be computed in $O(n \log n)$ time using linear space.*

Proof. We compute the odd MVC by a plane sweep. Let $x^1 < \cdots < x^u$ be the ascending sequence of all distinct x-coordinates. For each vertical line $g_i : x = x^i$ we determine in a preprocessing step the points b_i and t_i such that $g_i \cap \mathcal{A}_{\text{ver}} = [b_i, t_i]$. For this it suffices to go through the input points in lexicographical order. For each g_i we introduce numbers β_i and τ_i which we initially set to ∞. After the sweep β_i and τ_i will determine the odd MVC in the following way: If $\beta_i = \tau_i = \infty$, then there is no odd segment on g_i, otherwise g_i contains the odd segment $x^i \times [\beta_i, \tau_i]$. These two variables are sufficient since according to Lemma 4,(iv) there is at most one odd segment on g_i.

We use a sweep-line algorithm to compute the values β_i and τ_i. As usual, our sweep-line algorithm is supported by two data structures, the event-point queue and the sweep-line status. According to Lemma 4,(iv) there is an input point r with $y_c = y_r$ for each point c of changing parity and according to v we have to determine these points in order to get the odd segments. Thus, the event-point queue can be implemented as a sorted list of all y-coordinates of the input points. Note that the same y-value can occur more than once. This ensures that at each event point only one event takes place. The sweep-line status is a balanced binary tree in which each node corresponds to a connected components of G_ℓ, where ℓ is the current position of the horizontal sweep line.

Our sweep line ℓ is a horizontal line sweeping all rectangles in $\mathcal{R}_{\text{ver}}$ from bottom to top.

While the sweep line moves from one event point to the next, the sweep-line status maintains the connected components of G_ℓ in a balanced binary tree $\mathcal{T}$. Initially $\mathcal{T}$ is empty. Whenever ℓ reaches an event point, we update $\mathcal{T}$. For each component C of G_ℓ we store two indices l_C and r_C with the property that the leftmost node of C lies on g_{l_C} and the rightmost node on g_{r_C}. The tree $\mathcal{T}$ is organized such that $r_{C'} < l_C$ for two components of G_ℓ if C' is a left child of C, while $r_C < l_{C'}$ if C' is a right child.

The following component modifications can occur on an event: a component appears or disappears, one component is replaced by a new one, a component is enlarged or reduced, two or three components are joined or a component is split into two or three components. We can decide in constant time which type of event takes place, simply by evaluating b_i, t_i, and—if they exist—$b_{i\pm1}$ and $t_{i\pm1}$, where i is the index of the line which contains the input point that caused the current event. For each event we have to change the entries of at most three components and update $\mathcal{T}$ accordingly.

Each of these update operations takes $O(\log n)$ time. For example if a component is split into two, this component has to be found, its entries have to be updated and a new component has to be created and inserted to $\mathcal{T}$.

The correct values β_i, τ_i for each line g_i are computed during the sweep. At any point of time, the values β_i and τ_i indicate the information about the odd segment on g_i detected so far: $\beta_i = \infty$ means no odd segment has been found yet, while $\beta_i \neq \infty$ says that there is an odd segment on g_i with lower endpoint (x^i, β_i). If additionally $\tau_i \neq \infty$ then the upper endpoint has also been detected yet and the odd segment on g_i is $x^i \times [\beta_i, \tau_i]$. Thus, at each event we have to check whether there are odd segments that start or end at y_ℓ, the current y-value of the sweep line ℓ. According to Lemma 4,v, points of changing parity are always endpoints of odd segments, while bottom- or topmost points of $\mathcal{A}_{\text{ver}} \cap g_i$ may be endpoints. In order to find all endpoints, we have to consider the old and new entries of changing components whenever $\mathcal{T}$ is updated. Bottommost points occur, if a new components appears, a component is enlarged or components are joined. Topmost points occur, if a component disappears, is reduced or components are split. Points of changing parity can occur if the extent of a component changes, components are joined or split or one component is replaced by a new one. If we have found a bottommost point b_i, we check whether b_i is odd and hence the lower endpoint of the odd segment on g_i is b_i. We do this by examining l_C, r_C and i, where C is the component that contains b_i. If $l_C = r_C$ (degenerate rectangle) or the parities of l_C and i are different, b_i is odd and we set $\beta_i = b_i$. If we discover a point of changing parity, we check whether it is the lower or upper endpoint of the odd segment on g_i. If β_i is still ∞ the point of changing parity is the lower endpoint, otherwise the upper. We set accordingly $\beta_i = y_\ell$ or $\tau_i = y_\ell$. At a topmost point t_i we only have to check whether there is an odd segment on g_i and whether t_i is the upper endpoint of the odd segment. This is the case if $\beta_i \neq \infty$ and $\tau_i = \infty$, we then set $\tau_i = t_i$.

As there are at most $3n$ operations that change components during the sweep, we have to handle $O(n)$ of these checks. After sorting, each of the n events of our sweep takes $O(\log n)$ time. Thus, the total running time is $O(n \log n)$. $\square$

The odd MHC can be computed analogously.

5 An approximation algorithm

Our algorithm ApproxMMN proceeds in four phases, see Figure 10. In phase 0 we compute all pointers of type xnbor and ynbor and the set Z. In phase I we satisfy all pairs in $Z_{\text{ver}} \cup Z_{\text{hor}}$ by computing the network N_1, the union of a nice MVC $\mathcal{C}_{\text{ver}}$, a nice MHC $\mathcal{C}_{\text{hor}}$, and at most one additional line segment for each rectangle in $\mathcal{R}_{\text{ver}} \cup \mathcal{R}_{\text{hor}}$. In phase II we compute the staircase polygons that were mentioned in Section 2. The union of their interiors is area $\mathcal{A}_3$. Network N_2 consists of the boundaries of these polygons and segments that connect the boundaries to N_1. In phase III we compute a network N_3 of segments in $\mathcal{A}_3$. The resulting network $N_1 \cup N_2 \cup N_3$ satisfies Z.

Phase 0. In phase 0 we compute all pointers of type xnbor and ynbor and the set Z. We organize our data structures such that from now on we have constant-time access to all relevant information such as xnbor, ynbor, vertical and horizontal predecessors and successors from each point $p \in P$.

Phase I. First we compute the nice odd MVC and the nice odd MHC, denoted by $\mathcal{C}_{\text{ver}}$ and $\mathcal{C}_{\text{hor}}$, respectively. Then we compute the set $\mathcal{S}$ of additional segments according to Lemma 3. We compute $\mathcal{C}_{\text{ver}}$, $\mathcal{C}_{\text{hor}}$ and $\mathcal{S}$ such that from each point $p \in P$ we have constant-time access to the at most two *cover segments* (i.e. segments in $\mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}}$) that contain p and to the additional segments in the at most four rectangles incident to p.

Lemmas 1, 3, and 6 show that $N_1 = \mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}} \cup \mathcal{S}$ can be computed in $O(n \log n)$ time and that $|N_1| \leq |N_{\text{opt}}| + H + W$ holds.

Phase II. In general N_1 does not satisfy Z_{quad}; further segments are needed. In order to be able to bound the length of these new segments, we partition the plane into two areas $\mathcal{A}_{12}$ and $\mathcal{A}_3$ as indicated in Section 2. We wanted to define $\mathcal{A}_3$ such that $|N_{\text{opt}} \cap \mathcal{A}_3|$ were large enough for us to bound the length of the new segments. However, we were not able to define $\mathcal{A}_3$ such that we could at the same time (a) satisfy Z_{quad} by adding new segments exclusively in $\mathcal{A}_3$ and (b) bound their length. Therefore we put the new segments into two disjoint sets, N_2 and N_3, such that $N_1 \cup N_2 \subseteq \mathcal{A}_{12}$ and $N_3 \subseteq \mathcal{A}_3$. This enabled us to bound $|N_1 \cup N_2|$ by $3|N_{\text{opt}} \cap \mathcal{A}_{12}|$ and $|N_3|$ by $2|N_{\text{opt}} \cap \mathcal{A}_3|$.

We now prepare our definition of $\mathcal{A}_3$. Recall that $Q(q,1), \ldots, Q(q,4)$ are the four quadrants of the Cartesian coordinate system with origin q. Let $P(q,t) = \{p \in P \cap Q(q,t) \mid (p,q) \in Z_{\text{quad}}\}$ for $t = 1,2,3,4$. For example, in Figure 12, $P(q,1) = \{p_1, \ldots, p_5\}$. Due to the definition of Z_{quad} we have $Q(p,t) \cap P(q,t) = \{p\}$ for each $p \in P(q,t)$. Thus the area $\mathcal{A}_{\text{quad}}(q,t) = \bigcup_{p \in P(q,t)} \text{BBox}(p,q)$ is a staircase polygon. The points in $P(q,t)$ are the "stairs" of the polygon and q is the corner opposite the stairs. In Figure 12, $\mathcal{A}_{\text{quad}}(q,1)$ is the union of the shaded areas. In order to arrive at a definition of the area $\mathcal{A}_3$, we will start from polygons of type $\mathcal{A}_{\text{quad}}(q,t)$ and then subtract areas that can contain segments of N_1 or are not needed to satisfy Z_{quad}.

Let $\Delta(q,t) = \mathrm{int}\big(\mathcal{A}_{\mathrm{quad}}(q,t) \setminus (\mathcal{A}_{\mathrm{hor}} \cup \mathcal{A}_{\mathrm{ver}})\big)$, where $\mathrm{int}(M)$ denotes the interior of a set $M \subseteq \mathbb{R}^2$. In Figure 12, $\Delta(q,1)$ is the union of the three areas with dotted boundary. Let $\delta(q,t)$ be the union of those connected components A of $\Delta(q,t)$, such that $\partial A \cap P(q,t) \neq \emptyset$. In Figure 12, $\delta(q,1)$ is the union of the two dark shaded areas A and $\overline{A}$.

Due to the way we derived $\delta(q,t)$ from $\mathcal{A}_{\mathrm{quad}}(q,t)$, it is clear that each connected component A of $\delta(q,t)$ is a staircase polygon, too. The stairs of A correspond to the input points on ∂A, i.e. $P(q,t) \cap \partial A$. Let q_A be the point on ∂A that is closest to q. This is the corner of A opposite the stairs. The next lemma is very technical, but it is crucial for the estimation of our network within the $\delta(q,t)$ regions.

Lemma 7 *Areas of type $\delta(q,t)$ are pairwise disjoint.*

Proof. For each pair $(p,q) \in Z_{\mathrm{quad}}$ we define its *forbidden area* F_{pq} to be the union of $\mathrm{BBox}(p,q)$ and the intersection of (a) the halfplane not containing p that is bounded by the horizontal through q and (b) the open strip between the verticals through p and q, see Figure 7. We have $F_{pq} \cap (P \setminus \{p,q\}) = \emptyset$ since the existence of a point $r \in F_{pq} \cap (P \setminus \{p,q\})$ would contradict $(p,q) \in Z_{\mathrm{quad}}$.

Suppose there is a point $s \in \delta(q,t) \cap \delta(q',t')$ with $(q,t) \neq (q',t')$. Clearly $q \neq q'$ since $\delta(q,t) \subset \mathrm{int}(Q(q,t))$ and $\delta(q,t') \subset \mathrm{int}(Q(q,t'))$ and $\mathrm{int}(Q(q,t)) \cap \mathrm{int}(Q(q,t')) = \emptyset$ for $t \neq t'$. Since $\delta(q,t), \delta(q',t') \subseteq \mathcal{A}_{\mathrm{quad}}$ we know that there are points p and p' with $(p,q),(p',q') \in Z_{\mathrm{quad}}$ such that $s \in \mathrm{BBox}(p,q) \cap \mathrm{BBox}(p',q')$. Let $B = \mathrm{BBox}(p,q)$ and $B' = \mathrm{BBox}(p',q')$. Without loss of generality, we assume that p is to the right and above q.

We know that $p', q' \notin B$ since $B \subset F_{pq}$. Analogously $p, q \notin B'$. Let $\ell(x_q, y_p)$ and $r(x_p, y_q)$ be the other two corners of B, see Figure 8. There are three cases:

Case I: $B' \cap \{\ell, r\} = \emptyset$.

Recall that $B' \cap \{p,q\} = \emptyset$ and that $B \cap B' \neq \emptyset$. Thus B' lies in the vertically unbounded open strip $S_1 = (x_q, x_p) \times (-\infty, \infty)$ or in the horizontally unbounded open strip $S_2 = (-\infty, \infty) \times (y_q, y_p)$ determined by two opposite edges of B, see Figure 8. (Note that p' and q' cannot lie on the boundary of S_1 or S_2, otherwise (p,q) or (p',q') would not be in Z_{quad}.) Now if $B' \subset S_1$ (see the dashed rectangle in Figure 8), then p' or q' lies in F_{pq} contradicting $(p,q) \in Z_{\mathrm{quad}}$. If on the other hand $B' \subset S_2$ (see the dotted rectangle in Figure 8) then p or q lies in $F_{p'q'}$ contradicting $(p',q') \in Z_{\mathrm{quad}}$.

Case II: $B' \cap \{\ell, r\} = \{r\}$.

Now the upper left corner of B' lies in B since again $B' \cap \{p,q\} = \emptyset$. Thus the lower left corner of B' is an input point (p' or q') but lies in F_{pq} contradicting $(p,q) \in Z_{\mathrm{quad}}$.

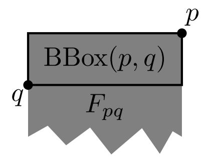


Figure 7: The forbidden area F_{pq} is shaded.

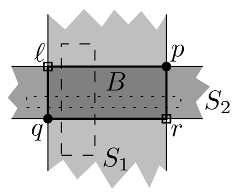


Figure 8: Case I.

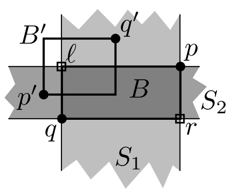


Figure 9: Case III.

Case III: $B' \cap \{\ell, r\} = \{\ell\}$.

In this case the lower right corner of B' lies in B and the upper right corner of B' lies above B in S_2. If p' was the upper right corner of B', we would have $q \in F_{p'q'}$, which contradicts $(p', q') \in Z_{\text{quad}}$. Thus p' lies in S_2 to the left of B and q' in S_1 above B, see Figure 9. Such a constellation is indeed possible. Note, however, that $B \cap B' \subset \text{BBox}(q, q')$. Furthermore $\{q, q'\} \in Z_{\text{ver}}$ since $\text{BBox}(q, q')$ and the open strip bounded by the verticals through q and q' are completely contained in $F_{pq} \cup F_{p'q'}$ and thus do not contain any input points except q and q'. These observations yield $s \in B \cap B' \subset \text{BBox}(q, q') \subset \mathcal{A}_{\text{ver}}$, which contradicts $s \in \delta(q, t)$ since $\delta(q, t)$ is contained in the complement of $\mathcal{A}_{\text{ver}}$.

□

We are now sure that we can treat each connected component A of $\delta(q, t)$ independently. Finally we define $\mathcal{A}_3 = \bigcup_{t \in \{1,2,3,4\}} \bigcup_{q \in P} \delta(q, t)$ and $\mathcal{A}_{12} = \mathbb{R}^2 \setminus \mathcal{A}_3$. This definition ensures that $N_1 \subset \mathcal{A}_{12}$ as desired. The set N_2 will be constructed as follows: for each connected component A of $\mathcal{A}_3$, we put $\partial A \setminus \bigcup N_1$ into N_2 and test whether N_1 contains a Manhattan path from q_A to q. If not, we add a further segment to N_2. This segment lies in $\mathcal{A}_{\text{hor}}$ and will be defined below. Since $\mathcal{A}_{\text{hor}}$ as well as ∂A are contained in $\mathcal{A}_{12}$, we have $N_2 \subset \mathcal{A}_{12}$. The set N_3 will be defined in phase III and will be arranged such that $N_3 \subset \mathcal{A}_3$.

We now describe how to compute $P(q, t)$ and how to find the connected components of $\delta(q, t)$. We compute all sets $P(q, t)$ by going through the input points and checking their Z_{quad}-partners. This takes linear time since $|Z_{\text{quad}}| = O(n)$. We sort the points in each set $P(q, t)$ according to their x-distance from q. This takes $O(n \log n)$ total time. The remaining difficulty is to decide which points in $P(q, t)$ are incident to the same connected component of $\delta(q, t)$. In Figure 12, $\{p_1, p_2\} \subset \partial A$ and $\{p_3, p_4, p_5\} \subset \partial \overline{A}$. For our description how to figure this out we assume $t = 1$ and $P(q, 1) = (p_1, \ldots, p_m)$. Note that each connected component of $\delta(q, 1)$ corresponds to a sequence of consecutive points in $P(q, 1)$. By definition, for each connected component A of $\delta(q, 1)$ and all $p_i, p_j \in A$ we have $p_i.\text{ynbor}[3] = p_j.\text{ynbor}[3]$.

We detect these sequences by going through $p_1, \ldots, p_m$. Let p_i be the current point and let A be the current connected component. If and only if $p_i.\text{ynbor}[3] \neq p_{i+1}.\text{ynbor}[3]$ there is a rectangle $R_A \in \mathcal{R}_{\text{hor}}$ that separates A from the next connected component of $\delta(q, 1)$. The rectangle R_A is defined by the point $v_A = p_i.\text{ynbor}[3]$ and its horizontal successor w_A, which in this case is unique, see Figure 12. It remains to specify the coordinates of the corner point q_A of A. Let p_0 be the (unique) vertical successor of q. Then $x_{q_A} = x_{p_0}$ and $y_{q_A} = y_{w_A}$.

At last, we want to make sure that $N_1 \cup N_2$ contains a Manhattan q–q_A path. The reason for this is that in phase III we will only compute Manhattan paths from each $p_i \in \partial A$ to q_A. Concatenating these paths with the q–q_A path yields Manhattan p_i–q paths since $q_A \in \text{BBox}(q, p_i)$. Note that segments in N_3 lie in $\mathcal{A}_3$ and thus cannot help to establish a q–q_A path within $\text{BBox}(q, q_A) \subset \mathcal{A}_{12}$.

The set N_1 contains a Manhattan q–p_0 path $\mathcal{P}_{\text{ver}}$ and a Manhattan v_A–w_A path $\mathcal{P}_{\text{hor}}$, since $(q, p_0) \in Z_{\text{ver}}$ and $(v_A, w_A) \in Z_{\text{hor}}$. If $q_A \in \mathcal{P}_{\text{ver}}$, then clearly N_1 contains a Manhattan q–q_A path. However, N_1 also contains a Manhattan q–q_A path if $q_A \in \mathcal{P}_{\text{hor}}$. This is due to the fact that $\mathcal{P}_{\text{ver}}$ and $\mathcal{P}_{\text{hor}}$ intersect. If $q_A \notin \mathcal{P}_{\text{ver}} \cup \mathcal{P}_{\text{hor}}$, then $\mathcal{P}_{\text{hor}}$ contains the point $c_A = (x_{q_A}, y_{v_A})$, which lies on the vertical through q_A on the opposite edge of R_A. Thus, to ensure a Manhattan q–q_A path in $N_1 \cup N_2$, it is enough to add the segment $s_A = \mathrm{Seg}[q_A, c_A]$ to N_2. We refer to such segments as *connecting segments*.

The algorithm ApproxMMN does not compute $\mathcal{P}_{\mathrm{ver}}$ and $\mathcal{P}_{\mathrm{hor}}$ explicitly, but simply tests whether $q_A \notin \bigcup N_1$. This is equivalent to $q_A \notin \mathcal{P}_{\mathrm{ver}} \cup \mathcal{P}_{\mathrm{hor}}$ since our covers are minimum and the bounding boxes of $\mathcal{P}_{\mathrm{ver}}$ and $\mathcal{P}_{\mathrm{hor}}$ are the only rectangles in $\mathcal{R}_{\mathrm{ver}} \cup \mathcal{R}_{\mathrm{hor}}$ that contain s_A. Due to the same reason and to the fact that cover edges are always contained in (the union of) edges of rectangles in $\mathcal{R}_{\mathrm{ver}} \cup \mathcal{R}_{\mathrm{hor}}$, we have that $s_A \cap \bigcup N_1 = \{c_A\}$. This shows that connecting segments intersect N_1 at most in endpoints. The same holds for segments in N_2 that lie on $\partial \mathcal{A}_3$. This is important as later on, in Section 6 we need that a segment in N_1 and a segment in N_2 intersect at most in their endpoints. We summarize:

Lemma 8 *In $O(n \log n)$ time we can compute the set N_2, which has the following properties: (i) $N_2 \subset \mathcal{A}_{12}$, (ii) a segment in N_1 and a segment in N_2 intersect at most in their endpoints, and (iii) for each region $\delta(q,t)$ and each connected component A of $\delta(q,t)$, $N_1 \cup N_2$ contains ∂A and a Manhattan q–q_A path.*

Proof. The properties of N_2 and the time bound for computing the connected components of $\mathcal{A}_3$ follow from the description above. For each connected component A of $\mathcal{A}_3$ the connecting segment s_A and the set $\partial A \setminus \bigcup N_1$ can be computed in $O(m)$ time, where $m = |P \cap \partial A|$. This is due to the fact that we have constant-time access to each of the $O(m)$ rectangles in $\mathcal{R}_{\mathrm{hor}} \cup \mathcal{R}_{\mathrm{ver}}$ that intersect ∂A and to the $O(m)$ segments of N_1 that lie in these rectangles. □

Phase III. Now, we finally satisfy the pairs in Z_{quad}. Due to Lemma 8 for each connected component A of $\mathcal{A}_3$ it is enough to compute a set of segments $B(A)$ such that the union of $B(A)$ and ∂A contains Manhattan paths from any input point on ∂A to q_A. We say that such a set $B(A)$ *bridges* A. The set N_3 will be the union over all sets of type $B(A)$. The algorithm Bridge(A) that we use to compute $B(A)$ is similar to the "thickest-first" greedy algorithm for rectangulating staircase polygons, see [GLN01]. However, we cannot use that algorithm since the segments that it computes do not lie entirely in $\mathcal{A}_3$.

For our description of Bridge(A) we assume that A lies in a region of type $\delta(q,1)$. Let again $(p_1, \ldots, p_m)$ denote the sorted sequence of points on ∂A. Note that ∂A already contains Manhattan paths that connect p_1 and p_m to q_A. Thus we are done if $m \le 2$. Otherwise let $p'_j = (x_{p_j}, y_{p_{j+1}})$, $a_j = \mathrm{Seg}[(x_{q_A}, y_{p'_j}), p'_j]$ and $b_j = \mathrm{Seg}[(x_{p'_j}, y_{q_A}), p'_j]$ for $j \in \{1, \ldots, m-1\}$, see Figure 13. We denote $|a_j|$ by α_j and $|b_j|$ by β_j. From now on we identify staircase polygon A with the tuple $(q_A, p_1, \ldots, p_m)$. Let B be the set of segments that algorithm Bridge(A) computes. Initially is $B = \emptyset$. The algorithm chooses an $i \in \{1, \ldots, m-1\}$ and adds—if they exist—a_{i-1} and b_{i+1} to B. This satisfies $\{(p_i, q), (p_{i+1}, q)\}$. In order to satisfy $\{(p_2, q), \ldots, (p_{i-1}, q)\}$ and $\{(p_{i+2}, q), \ldots, (p_{m-1}, q)\}$, we solve the problem recursively for the two staircase polygons $((x_{q_A}, y_{p_i}), p_1, \ldots, p_{i-1})$ and $((x_{p_{i+1}}, y_{q_A}), p_{i+2}, \ldots, p_m)$.

Our choice of i is as follows. Note that $\alpha_1 < \cdots < \alpha_{m-1}$ and $\beta_1 > \cdots > \beta_{m-1}$. Let $\Lambda = \{j \in \{1, \ldots, m-1\} \mid \alpha_j \le \beta_j\}$. If $\Lambda = \emptyset$, we have $\alpha_1 > \beta_1$, i.e.

```
ApproxMMN($P$)

Phase 0:   Neighbors and generat. set
for each $p \in P$ and $t \in \{1,2,3,4\}$ do
    compute $p$.xnbor$[t]$ and $p$.ynbor$[t]$
compute $Z = Z_{\text{ver}} \cup Z_{\text{hor}} \cup Z_{\text{quad}}$.

Phase I:   Compute $N_1$

compute odd MVC $\mathcal{C}_{\text{ver}}$ and MHC $\mathcal{C}_{\text{hor}}$
compute set $\mathcal{S}$ of additional segments
$N_1 \leftarrow \mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}} \cup \mathcal{S}$, $N_2 \leftarrow \emptyset$, $N_3 \leftarrow \emptyset$

Phase II:   Compute $N_2$

compute $\mathcal{A}_3$
for each connected comp. $A$ of $\mathcal{A}_3$ do
    $N_2 \leftarrow N_2 \cup (\partial A \setminus \bigcup N_1)$
    if $q_A \notin \bigcup N_1$ then
        $N_2 \leftarrow N_2 \cup \{s_A\}$

Phase III: Compute $N_3$

for each connected comp. $A$ of $\mathcal{A}_3$ do
    $N_3 \leftarrow N_3 \cup$ Bridge$(A)$

return $N = N_1 \cup N_2 \cup N_3$
```

Figure 10:

```
Bridge$\big(A = (q_A, p_1, \dots, p_m)\big)$

for $i = 1$ to $m - 1$ do
    compute $\alpha_i$ and $\beta_i$
return SubBridge$\big(1, m, 0, 0\big)$

SubBridge$\big(k, l, x_{\text{off}}, y_{\text{off}}\big)$

$A_{\text{curr}} = (q_A + (x_{\text{off}}, y_{\text{off}}), p_k, \dots, p_l)$
if $l - k < 2$ return $\emptyset$
$\Lambda = \big\{j \in \{k, \dots, l-1\}$ :
        $\alpha_j - x_{\text{off}} \le \beta_j - y_{\text{off}}\big\}$
$i = \max \Lambda \cup \{k\}$
if $i < l-1$ and $\alpha_i - x_{\text{off}} \le \beta_{i+1} - y_{\text{off}}$
    then $i = i + 1$
$B = \emptyset$
if $i > 1$ then
    $B = B \cup \{a_{i-1} \cap A_{\text{curr}}\}$
if $i < l - 1$ then
    $B = B \cup \{b_{i+1} \cap A_{\text{curr}}\}$
$x_{\text{new}} = x_{p_{i+1}} - x_{q_A}$
$y_{\text{new}} = y_{p_i} - y_{q_A}$
return $B \cup$
  $\cup$ SubBridge$(l, i-1, x_{\text{off}}, y_{\text{new}})$
  $\cup$ SubBridge$(i+2, l, x_{\text{new}}, y_{\text{off}})$
```

Figure 11:

A is flat and broad. In this case we choose $i = 1$, which means that only b_2 is put into B. Otherwise let $i' = \max \Lambda$. Now if $i' < m - 1$ and $\alpha_{i'} \le \beta_{i'+1}$, then let $i = i' + 1$. In all other cases let $i = i'$. The idea behind this choice of i is that it yields a way to balance α_{i-1} and β_{i+1}, which in turn helps to compare $\alpha_{i-1} + \beta_{i+1}$ to $\min\{\alpha_i, \beta_i, \alpha_{i-1} + \beta_{i+1}\}$, i.e. the length of the segments needed by any Manhattan network in order to connect p_i and p_{i+1} to q, see also the proof of Theorem 1.

To avoid expensive updates of the α- and β-values of the staircase polygons in the recursion, we introduce offset values x_{off} and y_{off} that denote the x-respectively y-distance from the corner of the current staircase polygon to the corner q_A of A. In order to find the index i in a recursion, we compare $\alpha_j - x_{\text{off}}$ to $\beta_j - y_{\text{off}}$ instead of α_j to β_j as in the definition of Λ above. Figure 11 shows the pseudo code of algorithm Bridge(A) for a staircase polygon A of type $\delta(q, 1)$.

Running time and performance of algorithm Bridge(A) are as follows:

Theorem 1 *Given a connected component A of $\mathcal{A}_3$ with $|P \cap \partial A| = m$, algorithm* Bridge *computes in $O(m \log m)$ time a set B of line segments with $|B| \le 2|N_{\text{opt}} \cap A|$ and $\bigcup B \subset A$ that bridges A.*

Proof. As for the running time, note that the monotone orders of $\alpha_1, \dots, \alpha_{m-1}$ and $\beta_1, \dots, \beta_{m-1}$ permit to find i by binary search in $O(\log m)$ time. The recursion tree has $O(m)$ nodes. Thus the algorithm runs in $O(m \log m)$ time.

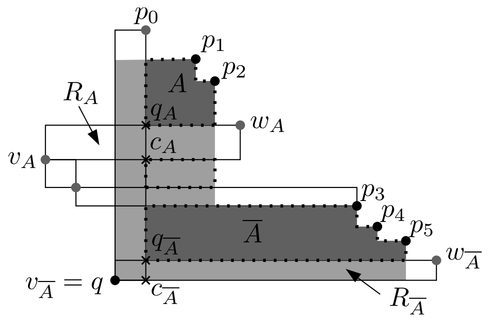


Figure 12: Notation: $\mathcal{A}_{\text{quad}}(q, 1)$ shaded, $\Delta(q, 1)$ with dotted boundary, and $\delta(q, 1) = A \cup A'$ dark shaded.

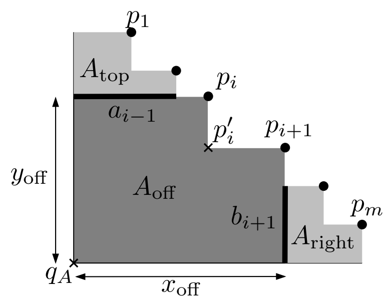


Figure 13: Notation for algorithm BRIDGE.

As for the performance, note that according to Lemma 7, A does not intersect any other connected component of $\mathcal{A}_3$. The performance proof is similar to the analysis of the greedy algorithm for rectangulation, see Theorem 10 in [GLN01].

Let i be the index determined in the first call to algorithm SUBBRIDGE, see Figure 11. If $i > 1$, let A_{top} be the part of A properly above a_{i-1}, otherwise let $A_{\text{top}} = \emptyset$. If $i < m - 1$, let A_{right} be the part of A properly to the right of b_{i+1}, otherwise let $A_{\text{right}} = \emptyset$. Now let $A_{\text{off}} = A \setminus (A_{\text{top}} \cup A_{\text{right}})$. Note that $a_{i-1} \cup b_{i+1} \subset A_{\text{off}}$.

By induction we can assume that $|B(A_{\text{top}})| \le 2|N_{\text{opt}} \cap A_{\text{top}}|$ and $|B(A_{\text{right}})| \le 2|N_{\text{opt}} \cap A_{\text{right}}|$. Thus, we are done if we can show that $\alpha_{i-1} + \beta_{i+1} \le 2|N_{\text{opt}} \cap A_{\text{off}}|$ (*). The network N_{opt} has to contain segments in A_{off} in order to satisfy $\{(p_i, q), (p_{i+1}, q)\}$, more precisely $|N_{\text{opt}} \cap A_{\text{off}}| \ge \min\{\alpha_i, \beta_i, \alpha_{i-1} + \beta_{i+1}\}$. Obviously (*) holds if N_{opt} contains segments of length at least $\alpha_{i-1} + \beta_{i+1}$ in A_{off}. Therefore, it remains to show that $\alpha_{i-1} + \beta_{i+1} \le 2 \min\{\alpha_i, \beta_i\}$. We make a case distinction depending on how i was derived. If $\Lambda = \emptyset$, then $i = 1$, $\alpha_1 > \beta_1$ and $A_{\text{top}} = \emptyset$. In this case only b_2 is added to B and $\beta_2 < \min\{\alpha_1, \beta_1\} = \beta_1$. If $i' = \max \Lambda = m - 1$, an analogous argument holds. Next we analyze the case $i' < m - 1$ and $\alpha_{i'} > \beta_{i'+1}$, where i is set to i'. This yields that $\beta_{i+1} < \alpha_i$ and thus $\alpha_{i-1} + \beta_{i+1} < 2\alpha_i$. On the other hand, by the definition of Λ, we have $\alpha_i \le \beta_i$. Hence $2\alpha_i \le 2 \min\{\alpha_i, \beta_i\}$. It remains to analyze the case $i' < m - 1$ and $\alpha_{i'} \le \beta_{i'+1}$, where i is set to $i' + 1$. This yields $\alpha_{i-1} \le \beta_i$ and thus $\alpha_{i-1} + \beta_{i+1} < 2\beta_i$. On the other hand, by the definition of Λ, we now have $\alpha_i > \beta_i$. Hence $2\beta_i \le 2 \min\{\alpha_i, \beta_i\}$. □

We conclude this section by analyzing the running time of APPROXMMN.

Theorem 2 APPROXMMN *runs in* $O(n \log n)$ *time and uses* $O(n)$ *space.*

Proof. Each of the four phases of our algorithm takes $O(n \log n)$ time: for phase 0 refer to Lemma 2, for phase I to Lemmas 3 and 6, for phase II to Lemma 8 and for phase III to Theorem 1. APPROXMMN outputs $O(n)$ line segments. □

6 The approximation factor

As desired we can now bound the length of N in $\mathcal{A}_{12}$ and $\mathcal{A}_3$ separately. Theorem 1 and Lemma 7 directly imply that $|N \cap \mathcal{A}_3| = |N_3| \leq 2|N_{\text{opt}} \cap \mathcal{A}_3|$. Note that by $|N_{\text{opt}} \cap \mathcal{A}_3|$ we actually mean $|\{s \cap \mathcal{A}_3 : s \in N_{\text{opt}}\}|$. It remains to show that $|N \cap \mathcal{A}_{12}| = |N_1 \cup N_2|$ is bounded by $3|N_{\text{opt}} \cap \mathcal{A}_{12}|$.

Recall that by Lemmas 1 and 3, $|N_1| \leq |N_{\text{opt}}| + H + W$. Since the segments of N_{opt} that were used to derive the estimation of Lemma 1 lie in $\mathcal{A}_{\text{ver}} \cup \mathcal{A}_{\text{hor}} \subset \mathcal{A}_{12}$, even the stronger bound $|N_1| \leq |N_{\text{opt}} \cap \mathcal{A}_{12}| + H + W$ holds. It remains to analyze the length of N_2 segments. Let N_2^{ver} (N_2^{hor}) denote the set of all vertical (horizontal) segments in N_2. We will compare the length of N_2^{ver} to the length of $\mathcal{C}_{\text{ver}}$ and the length of N_2^{hor} to the length of $\mathcal{C}_{\text{hor}}$. Lemma 11 will yield the desired length bounds. In the following we show how the length bound for N_2^{ver} is obtained, this is the more complicated case as the connecting segments are vertical. First, we need to distinct the connecting segments and all other segments of N_2. We call the non-connecting segments in N_2 *boundary segments* as they lie on $\partial\mathcal{A}_3$. Due to Lemma 8, segments in N_2^{ver} and segments in $\mathcal{C}_{\text{ver}}$ intersect at most in segment endpoints. Thus, a horizontal line ℓ with $\ell \cap P = \emptyset$ does not contain any point that lies at the same time in $\bigcup \mathcal{C}_{\text{ver}}$ and in $\bigcup N_2^{\text{ver}}$. We restrict ourselves to such lines, this makes no difference in terms of overall length as we exclude only a finite number of lines. In order to obtain Lemma 11, we will characterize the sequences that are obtained by the intersection of such a line ℓ with cover and boundary segments and cover and connecting segments, see Lemma 9 and Lemma 10, respectively.

Lemma 9 *Let ℓ be a horizontal line with $\ell \cap P = \emptyset$ and $\ell \cap \text{BBox}(P) \neq \emptyset$. Consider the sequence of boundary and cover segments intersected by ℓ. Then*

(i) No more than two boundary segments are consecutive.

(ii) The left- and the rightmost segments are cover segments.

Proof. We show that each boundary segment s is (directly) preceeded or succeeded by a cover segment. This implies immediately (i). The kind of cover segment (predecessor or successor) that is assigned to a boundary segment shows that no boundary segments can be left- or rightmost and thus (ii) follows.

We show the above statement for boundary segments that lie on the boundary of a connected component A of $\mathcal{A}_3$. W.l.o.g. we assume that A is part of a region of type $\delta(q, 1)$. Let $p_1, \ldots, p_m$ be the input points on ∂A ordered according to x-distance from q. As earlier, let $v_A = p_1.\text{ynbor}[3]$ and let w_A be the horizontal successor of v_A. Let R denote the rectangle in $\mathcal{R}_{\text{ver}}$ defined by the point q and its vertical successor p_0, see Figure 14. Let $p_\ell = s \cap \ell$ and let y_ℓ be the y-coordinate of ℓ. Note that $p_\ell \in \bigcup N_2^{\text{ver}}$ and thus $p_\ell \notin \bigcup \mathcal{C}_{\text{ver}}$. There are two cases for the type of s.

First, s could be the boundary segment to the left of A. In this case s lies on the right vertical edge of R. Let $q_\ell = (x_q, y_\ell)$ be the point opposite of p_ℓ on the left vertical edge of R. Then $q_\ell \in \bigcup \mathcal{C}_{\text{ver}}$ since $R \in \mathcal{R}_{\text{ver}}$ and $p_\ell \notin \bigcup \mathcal{C}_{\text{ver}}$. Due to $\text{int}(R) \subset \text{int}(\mathcal{A}_{12})$, no boundary segments intersects the relative interior of $\text{Seg}[p_\ell, q_\ell]$, and thus p_ℓ is preceeded by $q_\ell \in \bigcup \mathcal{C}_{\text{ver}}$ on ℓ.

Second, s could be a vertical "staircase segment" to the right of A. In this case we show that s is succeeded by a segment in $\mathcal{C}_{\text{ver}}$. There are two subcases:

either s is the left edge of $\mathrm{BBox}(p_i, p_{i+1})$ for some $i \in \{1, \ldots, m-1\}$ or the left edge of $\mathrm{BBox}(p_m, w_A)$. For the first subcase let β denote $\mathrm{BBox}(p_i, p_{i+1})$. We show that ℓ intersects a vertical cover segment in β. At the same time we show that $\beta \cap \mathcal{A}_3 = \emptyset$, and hence there is no boundary segment in the interior of β. This is done by characterizing the point pairs $(p', q') \in Z_{\mathrm{quad}}$ with $\mathrm{BBox}(p', q') \cap \mathrm{int}(\beta) \neq \emptyset$ and showing that the connected component of $\mathcal{A}_3$ that is incident to p' does not intersect β. Let σ and τ be the vertical and horizontal strips, respectively, that are induced by β, see Figure 14. The strip τ does not contain any input point to the left of β since this would contradict p_i and p_{i+1} lying in the same connected component of $\delta(q, 1)$. The strip σ does not contain any input point below β since this would contradict $(p_{i+1}, q) \in Z_{\mathrm{quad}}$. Let β' be β minus its right and top edge. There is no input point in β', otherwise there would be a point $p \in \beta'$ with $(p, q) \in Z_{\mathrm{quad}}$ contradicting p_i and p_{i+1} being consecutive. Let r be the rightmost input point on the top edge of β and let t be the topmost input point on the right edge of β. (Possibly $p_i = r$ and $p_{i+1} = t$.) Since there is a point $r' \in Q(r, 4)$ with $(r', r) \in Z_{\mathrm{hor}}$ and a point $t' \in Q(t, 2)$ with $(t', t) \in Z_{\mathrm{ver}}$, we must have that $q' = t$ and $p' \in Q(q', 2)$, otherwise $\mathrm{BBox}(p', q')$ would not intersect $\mathrm{int}(\beta)$. Observe that the rectangle $\mathrm{BBox}(r, r') \in \mathcal{R}_{\mathrm{hor}}$ splits $\mathrm{BBox}(p', q')$ into two connected components. However, the component incident to p' does not intersect $\mathrm{int}(\beta)$, and thus $\beta \cap \mathcal{A}_3 = \emptyset$. Since $\mathrm{BBox}(t, t') \in \mathcal{R}_{\mathrm{ver}}$ and $y_{t'} \geq y_{p_i}$, it is clear that ℓ intersects a vertical cover segment in β, either the one that is induced by the non-degenerate rectangle $\mathrm{BBox}(t, t')$ if $y_\ell \geq y_t$ or by the degenerate rectangle $\mathrm{BBox}(p_{i+1}, t)$ itself if $y_\ell < y_t$.

Last, we examine the subcase that ℓ intersects $\beta = \mathrm{BBox}(p_m, w_A)$. We have to proceed differently as we lose the property that no input point lies in the vertical strip below β. Consider $b = p_m.\mathrm{xnbor}[4]$ (allowing $b = w_A$). We assume w.l.o.g. $y_{p_m} > y_b$ otherwise let $b = b.\mathrm{xnbor}[4]$ until this is the case. Now, b could lie in $\mathrm{int}(\beta)$, but only if there is a point b' with $x_{b'} = x_b$ and $y_{b'} < y_{w_A}$ otherwise there would be a point $p \in \mathrm{int}(\beta)$ with $(p, q) \in Z_{\mathrm{quad}}$. We discard this case for a moment and assume that already $y_b < y_{w_A}$ holds. Now, there is a point $p' \in Q(b, 2)$ with $(p', b) \in Z_{\mathrm{ver}}$. By the construction, it is clear that $y_{p'} \geq y_{p_m}$ and thus the vertical line through b splits β into two connected components. For the component β' incident to p_m we can use the same argument as above to show that $\beta' \cap \mathcal{A}_3 = \emptyset$ since the vertical strip below β' does not contain any input points by construction. Hence, s is succeeded by a vertical cover segment in $\mathrm{BBox}(p', b)$. Now, back to the discarded case: if $y_\ell < y_b$, s is succeeded by the degenerate rectangle $\mathrm{BBox}(b, b')$, otherwise the same argument holds with $p' \in Q(b, 2)$ and $(p', b) \in Z_{\mathrm{ver}}$. □

For the following characterization of connecting segments note that such segments lie only in non-degenerate rectangles of $\mathcal{R}_{\mathrm{hor}}$.

Lemma 10 *Let ℓ be a horizontal line that intersects the interior of a rectangle $R_\ell \in \mathcal{R}_{\mathrm{hor}}$. Consider the sequence of connecting and cover segments in R_ℓ. Then*

(i) No connecting segment lies on a vertical edge of R_ℓ.

(ii) No more than two connecting segments are consecutive.

(iii) At least one of the two leftmost segments is a cover segment.

(iv) At least one of the two rightmost segments is a cover segment.

(v) The left- or rightmost segment is a cover segment.

Proof. In order to show (i), we show that no connecting segment is incident to an input point. By construction, each connecting segment $s_A = \mathrm{Seg}[q_A, c_A]$ lies on a vertical edge of a rectangle $R = \mathrm{BBox}(q, p_0) \in \mathcal{R}_{\mathrm{ver}}$ and in a rectangle $R_A = \mathrm{BBox}(v_A, w_A) \in \mathcal{R}_{\mathrm{hor}}$. By construction must R be non-degenerate, otherwise $q_A \in \bigcup \mathcal{C}_{\mathrm{ver}}$. Thus, $c_A \neq q$. Clearly $q_A \neq q$. Now $\{c_A, q_A\} \cap P \setminus \{q\} \neq \emptyset$ would contradict $(p, q) \in Z_{\mathrm{quad}}$ for any point $p \in \partial A \cap P$. Hence, s_A is not incident to an input point.

Now, since a connecting segment s_A is not in $\mathcal{C}_{\mathrm{ver}}$ and lies on a vertical edge of a rectangle $R \in \mathcal{R}_{\mathrm{ver}}$ it is pre- or succeeded by the cover segment on the opposite edge of R. This directly shows (ii), (iii) and (iv).

Our proof for (v) is by contradiction: we assume that the leftmost segment s and the rightmost segment s' in R_ℓ are connecting segments. Let $R_\ell = \mathrm{BBox}(v, w)$. Let w.l.o.g. v be the lower left point and w be the upper right point of R_ℓ, see Figure 15. Let A and A' be the connected components of $\mathcal{A}_3$ with $s = s_A$ and $s' = s_{A'}$. Note that $R_A = R_{A'} = R_\ell$. Let R and R' be the rectangles in $\mathcal{R}_{\mathrm{ver}}$ whose vertical edges contain s and s', respectively. Clearly s must lie on the left edge of R and s' on the right edge of R'. Thus, A must be a region of type $\delta(q, 2)$ or $\delta(q, 3)$. First, assume $A \subseteq \delta(q, 2)$ for some $q \in P$. Then would A lie above R and q below R, see Figure 15. However, this is impossible. Let p be the leftmost point in $P(q, t) \cap \partial A$. Then p has a Z_{hor} partner in $Q(p, 4)$ which contradicts (p, q) being in Z_{quad}. Thus, $A \subseteq \delta(q, 3)$ and analogously $A' \subseteq \delta(q', 1)$ for some $q' \in P$, see Figure 16. Now, the Manhattan v–w path in N_1 contains at least one of the corner points q_A or $q_{A'}$. This contradicts s and s' both being connecting segments. $\square$

Combining Lemma 9 and Lemma 10 yields:

Lemma 11 $|N_2^{\mathrm{ver}}| \leq 2|\mathcal{C}_{\mathrm{ver}}| - H$ *and* $|N_2^{\mathrm{hor}}| \leq 2|\mathcal{C}_{\mathrm{hor}}| - W$.

Proof. For a horizontal line ℓ with $\ell \cap P = \emptyset$ we want to compare the numbers $\#N_2^{\mathrm{ver}}$ and $\#\mathcal{C}_{\mathrm{ver}}$ of segments in N_2^{ver} and $\mathcal{C}_{\mathrm{ver}}$ intersected by ℓ, respectively. If we show that $\#N_2^{\mathrm{ver}} \leq 2\#\mathcal{C}_{\mathrm{ver}} - 1$, $|N_2^{\mathrm{ver}}| \leq 2|\mathcal{C}_{\mathrm{ver}}| - H$ follows. (Sweep $\mathrm{BBox}(P)$ from bottom to top. The at most n lines that we have to exclude draw no distinction in terms of length.) It remains to show that $\#N_2^{\mathrm{ver}} \leq 2\#\mathcal{C}_{\mathrm{ver}} - 1$. Observe that due to Lemma 10 (i), ℓ intersects connecting segments at most within the interior of a rectangle in $\mathcal{R}_{\mathrm{hor}}$. On the other hand, due to the definition of $\mathcal{A}_3$, ℓ does not intersect any boundary segments within the interior of such a rectangle. We investigate three cases.

First, consider the case that ℓ intersects no connecting segment. Thus, only cover and boundary segments are intersected. By Lemma 9 at most two boundary segments are consecutive and both the left- and rightmost intersected segments are cover segments. By a simple counting argument, this even yields $\#N_2^{\mathrm{ver}} \leq 2\#\mathcal{C}_{\mathrm{ver}} - 2$.

Second, consider the case that ℓ intersects no boundary segments. Then, by Lemma 10 (ii) and (v), at most two connecting segments are consecutive and the left- or rightmost segment is a cover segment. Now, further using Lemma 10 (iii) and (iv) yields $\#N_2^{\mathrm{ver}} \leq 2\#\mathcal{C}_{\mathrm{ver}} - 1$ as desired.

Third, consider the case that ℓ intersects both boundary and connecting segments. Lemmas 9 (ii) and 10 (v) yield that the left- or rightmost intersected segment is a cover segment. Thus if in the sequence of segments intersected by ℓ at most two segments in N_2^{ver} are consecutive, we are in the same situation as in the second case. Hence $\#N_2^{\text{ver}} \leq 2\#\mathcal{C}_{\text{ver}} - 1$.

However, there is a case in which more than two N_2^{ver} segments are consecutive: two consecutive boundary segments are succeeded (or preceeded) by a rectangle $R \in \mathcal{R}_{\text{hor}}$. Due to Lemma 10 (iii) and (iv) at most one of the following two segments within R is a connecting segment. Hence, no more than three segments in N_2^{ver} are consecutive. If there are three consecutive segments in N_2^{ver}, then one of them is a connecting segment that is left- or rightmost in R. W.l.o.g. we assume that the connecting segment is leftmost in R. Then by Lemma 10 (v) the rightmost segment in R is a cover segment. From this we deduce two things: (a) since ℓ intersects at most one rectangle in $\mathcal{R}_{\text{hor}}$, three consecutive segments in N_2^{ver} occur at most once. (b) If there are three such segments, then by Lemma 9 (ii) both the left- and rightmost segments intersected by ℓ are cover segments. Hence, we again have $\#N_2^{\text{ver}} \leq 2\#\mathcal{C}_{\text{ver}} - 1$.

To bound the length of N_2^{hor} segments is easier since connecting segments are vertical. An analogous, simpler argument holds. ◻

This finally settles the approximation factor of ApproxMMN.

Theorem 3 $|N| \leq 3|N_{\text{opt}}|$.

Proof. By Lemma 11 and $|\mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}}| \leq |N_{\text{opt}} \cap \mathcal{A}_{12}|$ we have $|N_2| \leq 2|N_{\text{opt}} \cap \mathcal{A}_{12}| - H - W$. Together with $|N_1| \leq |N_{\text{opt}}| + H + W$ this yields $|N_1 \cup N_2|/|N_{\text{opt}} \cap \mathcal{A}_{12}| \leq 3$. Theorem 1 and Lemma 7 show that $|N_3|/|N_{\text{opt}} \cap \mathcal{A}_3| \leq 2$. Then, the disjointness of $\mathcal{A}_{12}$ and $\mathcal{A}_3$ yields $|N|/|N_{\text{opt}}| \leq \max\{|N_1 \cup N_2|/|N_{\text{opt}} \cap \mathcal{A}_{12}|, |N_3|/|N_{\text{opt}} \cap \mathcal{A}_3|\} \leq 3$. ◻

7 Experiments

In Figure 18 a network computed by ApproxMMN and an MMN of the same point set are depicted. The example indicates that there are point sets P for which the ratio $|N|/|N_{\text{opt}}|$ is arbitrarily close to 3, where N is the Manhattan network that ApproxMMN computes for P. The reason for the particularly bad performance of ApproxMMN on this point set is that neither the w_A–q path nor the p_0–q path (bold solid line segments) contain the point q_A. This forces ApproxMMN to use the connecting segment s_A. To show that our algorithm performs better on average instances, we implemented ApproxMMN and an exact solver and ran them on two classes of randomly generated point sets.

7.1 Experimental set-up

We implemented ApproxMMN in C++ using the compiler gcc-3.3. The exact solver is based on a mixed integer programming (MIP) formulation [WBS04] for the MMN problem. We used the MIP solver *Xpress-Optimizer* (2003) [Das03] by Dash Optimization with the C++ interface of the BCL library to compute optimal solutions at least for small instances. The two classes of random point sets, Square and HalfCircle, were generated as follows.

Square-k instances were generated by drawing n different points with uniform distribution from a $kn \times kn$ integer grid. We wanted to see the effects of having more (k small) or less (k large) points with the same x- or y-coordinate. If a pair of points shares a coordinate, the Manhattan path connecting them is uniquely determined. We used $k \in \{1, 2, 5, 10\}$. For an example of a Square-01 instance see Figure 19.

HalfCircle-k instances consist of a point p_1 at the origin o and $n-1$ points on the upper half of the unit circle. The points are distributed as follows. The angular range $I = [0, \pi/4]$ is split into k subranges $I_1, \ldots, I_k$ of equal length. We used $k \in \{1, 2, 5, 10, 99\}$. Then $n-1$ random numbers $r_2, \ldots, r_n$ are drawn from I. If the number r_i falls into a subinterval of even index, it is mapped to the point $p_i = (\sin r_i, \cos r_i)$, otherwise to $p_i = (-\sin r_i, \cos r_i)$. The resulting points p_i (except for the topmost point in each quadrant and the "bottommost" point in each subinterval) form pairs (p_i, p_1) that lie in Z_{quad}. This makes HalfCircle instances very different from Square instances where usually only very few point pairs belong to Z_{quad}. For an example of a HalfCircle-02 instance, see Figure 20.

We generated instances of the above types and solved them with ApproxMMN and with the Xpress-Optimizer using the MIP formulation from [WBS04]. The results of our experiments can be found in Figures 21–23. In all graphs the sample size, i.e. the number of points per instance, is shown on the x-axis. For each sample size we generated 50 instances and averaged the results over those. In Figure 21(a) the y-axis shows the performance ratio of ApproxMMN, i.e. $|N|/|N_{\text{opt}}|$. In Figures 21(b) and 22 we compared the performance ratios of ApproxMMN, a slightly modified variant of ApproxMMN, and the $O(n^3)$-time factor-4 approximation algorithm of Gudmundsson et al. [GLN01]. In the graphs we skipped the factor-8 approximation algorithm [GLN01] because its results were only slightly worse than those of the factor-4 approximation: the difference was below 5%. We also tested the performance of the following simple method to which we will refer as LPsolver+rounding. In the MIP formulation there is a 0–1 variable v_e for each edge e of the grid induced by the input points. If $v_e = 1$ then e is part of the solution, otherwise not. Our method LPsolver+rounding solves the relaxation of the MIP and returns a network which consists of all edges e with $v_e > 0$. By the construction of the MIP it is clear that this network is a Manhattan network.

In Figure 23 the y-axis measures the ratio between the running times of the corresponding algorithms over the running time of ApproxMMN. The asymptotic runtime of our implementation is $\Theta(n^2)$, the CPU time consumption was measured on an Intel Xeon machine with 2.6 GHz and 2 GB RAM under the operating system Linux-2.4.20. The Xpress-Optimizer was run on the same machine.

7.2 Results

The MIP solver required an unacceptable amount of time (i.e. at least several hours) on HalfCircle-01 instances of more than 50 points and on Square-01 instances of more than 250 points. The performance ratio of ApproxMMN seems to approach 1.1–1.2 on Square instances of increasing size, and 1.3–1.5

on HalfCircle instances, see Figure 21(a). On HalfCircle instances we observed that with an increasing number k of subranges the performance of ApproxMMN degrades. The reason for this is that each subrange induces a connected component of type $\delta(o, 1)$ or $\delta(o, 2)$. Thus, the length of the network N_2 increases with an increasing number of subranges. Indeed, the length of N_2 seems to be the bottleneck of our algorithm.

To reduce this effect we implemented a slightly modified variant of ApproxMMN, to which we will refer as ApproxMMN-var. This variant changes only the networks N_2 and N_3. We explain the approach exemplarily for a connected component A of type $\delta(q, 1)$. Let again $p_1, \dots, p_m$ be the input points on ∂A ordered according to x-distance from the input point q in the lower left corner of A. Let R_A be the rectangle in $\mathcal{R}_{\text{hor}}$ that touches the bottom edge of A, see Figure 14. Let v_A and w_A be the input points that span R_A.

In phase II ApproxMMN-var adds only the segments $\text{Seg}[p_1, (x_{p_1}, y_{w_A})]$ and $\text{Seg}[p_m, (x_{p_1}, y_{p_m})]$ instead of the whole boundary of A to N_2. Accordingly, the connecting segment is now $\text{Seg}[(x_{p_1}, y_{w_a}), (x_{p_1}, y_{v_A})]$. As before, the connecting segment is inserted only if necessary. In phase III, a similar algorithm to algorithm Bridge is used to establish connections from $p_2, \dots, p_{m-1}$ to (x_{p_1}, y_{w_a}). Here we use the thickest-first algorithm introduced in [GLN01]. Now the parts of ∂A that represent the staircase between p_1 and p_m are only inserted if the thickest-first algorithm requires this. However, the segments that lie on ∂A are now inserted in N_3, and there is the rub. We were not able to prove $|N_3 \cap \mathcal{A}_3| \leq 2|N_{\text{opt}} \cap \mathcal{A}_3|$ for ApproxMMN-var.

However, as we had hoped, the performance of ApproxMMN-var was better than that of ApproxMMN. Figure 21(b) shows the performance of ApproxMMN, ApproxMMN-var, the factor-4 approximation algorithm by Gudmundsson et al. [GLN01] and LPsolver+rounding. Exemplarily for the Square instances, we included the graphs for the Square-10 instances. The behavior of the algorithms was similar on the other Square instances, with slightly better results. On Square instances ApproxMMN performed only slightly worse than ApproxMMN-var. This is different on HalfCircle instances as Figure 22 shows. Especially with an increasing number of subranges the influence of N_2 on the total length of the network increases. The performance of LPsolver+rounding was amazingly good. The worst performance ratio of this method was 1.078. It occurred on a Square-10 instance with 25 points. Moreover, LPsolver+rounding solved all Circle instances optimally.

The CPU time of ApproxMMN depends neither on the value of k nor on the instance type. Solving instances with 3000 points took only about 5–6 seconds. In contrast to that, the runtime of the exact solver heavily depended on the value of k and even more on the instance type. Square instances were solved the faster the smaller k, because then the probability for two points having the same x- or y-coordinate is higher, which predetermines a larger number of segments to be in the network. The average CPU time of the exact solver on Square-10 instances with 250 points was about 170 seconds, compared to 0.1–0.2 seconds for ApproxMMN. HalfCircle instances were solved slower the smaller k, because then more grid points and grid segments lie in more rectangles of $\mathcal{R}_{\text{quad}}$, which means that the MIP formulation has more constraints and variables. Generally Square instances were solved much faster than HalfCircle instances. This is due to the number of Z_{quad} pairs, which is significant higher in HalfCircle instances. The MIP formulation requires $O(n^2)$ variables and constraints for a point pair in Z_{quad}, while it requires only $O(n)$ variables and constraints for point pairs in $Z_{\text{ver}} \cup Z_{\text{hor}}$. (There are Z_{quad} pairs that require $\Theta(n^2)$ variables and constraints.)

We wanted to see how fast the MIP solver becomes if it only has to compute a solution as good as the one computed by ApproxMMN. The Xpress-Optimizer allows to specify a bound that stops the branch-and-bound process as soon as the target function is at least as good as the bound. We refer to this version of the MIP solver as MIPsolver-approx and to the original exact version as MIPsolver-opt. The results are shown in Figure 23, where the average ratio between the running times of MIPsolver-approx, MIPsolver-opt and LPsolver+rounding over the running time of ApproxMMN is shown. For Square-10 instances, MIPsolver-approx is not much faster than MIPsolver-opt. This changes with decreasing k: for Square-01 instances MIPsolver-approx takes only about half the time of MIPsolver-opt. This is due to the fact that the smaller k the more segments in a Manhattan network are predetermined. The method LPsolver+rounding turned out to run only slightly faster than MIPsolver-opt. HalfCircle instances were solved relatively fast by MIPsolver-approx. Solving HalfCircle-01 instances with 45 points took MIPsolver-opt on average 2200 seconds CPU time as compared to 1.2 seconds for MIPsolver-approx (and 0.01 seconds for ApproxMMN).

Finally we compared the values of the objective function of the MIP and its LP relaxation. We found out that there are only few instances where the relaxation yields a smaller value of the objective function than the MIP. For an example of an instance with a gap between the two values, see [CNV05]. We found gaps in only three Square-10 instances. Moreover, in all of these cases the gap was very small, namely less than 0.011% of the value of the objective function in the MIP formulation.

Note that the existence of a gap means that the face of the solution polyhedron with maximum objective function value has *only* fractional corners, while the existence of a fractional corner does not imply a gap. If, however, the LP solver finds such a fractional corner and there is an integral corner, then our rounding scheme returns a non-optimal network. We conjecture that there are only few point configurations that cause a gap and that these configurations cannot occur in HalfCircle instances.

7.3 Conclusion

For time-critical applications clearly ApproxMMN or ApproxMMN-var are the methods of choice. They solve instances with 3000 points within 5–6 seconds CPU time. On average point sets the networks they compute are usually not more than 50% longer than an MMN. Within a threshold of 100 seconds CPU time we were only able to compute optimal networks of the following sizes: HalfCircle instances of at most 25 points and Square instances of at most 175 points. The (polynomial-time!) method LPsolver+rounding returns amazingly good results, but it is only slightly faster than MIPsolver-opt.

8 Open problems

The main open question is the complexity status of the MMN problem. Until now there are not even hints whether it is polynomially solvable, it is NP-hard, or has intermediate status. In the latter cases it would be of interest to find out whether a polynomial-time approximation scheme exists or whether the MMN problem cannot be approximated arbitrarily well. Very recently Chepoi et al. [CNV05] gave a factor-2 approximation algorithm for the MMN problem that is based on cleverly rounding the solution of a linear program, namely the relaxation of the MIP [WBS04] that we used as a benchmark for our experiments. However, the running time of their algorithm is in $\Omega(n^8)$ as their linear program uses $O(n^2)$ variables and constraints. So it would be interesting to see whether there is also a factor-2 approximation algorithm that runs in $O(n \log n)$ time. In order to solve larger instances optimally, it would be of interest to design a fixed-parameter algorithm. However, it is unclear to us what to choose as parameter.

We conclude with two variants of the problem. The first variant is the *real MMN problem* where apart from the point set an underlying rectilinear grid G is given, e.g. the streets of Manhattan, on which the network has to lie. Again each pair of points must be connected by a shortest possible rectilinear path and the length of the network is to be minimized. However, the shortest rectilinear path connecting two points can now be longer than a usual Manhattan path, see Figure 17. The real MMN problem is at least as hard as the MMN problem since G can be set to the grid induced by the input points.

Chepoi et al. [CNV05] suggest another variant of the MMN problem, the *F-restricted MMN problem*. Given a point set P and a set F of pairs of points in P, find a network of minimum length that connects the point pairs in F with Manhattan paths. This variant also generalizes the MMN problem, which is an F-restricted MMN problem where F is a generating set. The F-restricted MMN problem is NP-hard, since it also generalizes the rectilinear Steiner arborescence problem, which is NP-hard [SS00]. In the rectilinear Steiner arborescence problem only point sets P in the first quadrant are considered. The aim is to find a rectilinear network of minimum length that connects all points in P to the origin o. This is equivalent to solving the F-restricted MMN problem for $P' = P \cup \{o\}$ and $F = \{o\} \times P$.

Acknowledgments

We would like to thank Hui Ma for implementing large parts of ApproxMMN, and Anita Schöbel, Marc van Kreveld, Leon Peeters, Karin Höthker, Gautam Appa, Dorit Hochbaum, and Raghavan Dhandapani for interesting discussions.

References

[ADM+95] Sunil Arya, Gautam Das, David M. Mount, Jeffrey S. Salowe, and Michiel Smid. Euclidean spanners: Short, thin, and lanky. In *Proc. 27th Annu. ACM Sympos. Theory Comput. (STOC'95)*, pages 489–498, Las Vegas, 29 May–1 June 1995.

[AGM$^+$90] Stephen F. Altschul, Warran Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[Che89] L. Paul Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39:205–219, 1989.

[CNV05] Victor Chepoi, Karim Nouioua, and Yann Vaxés. A rounding algorithm for approximating minimum Manhattan networks. `http://www.lif-sud.univ-mrs.fr/~chepoi/papers.html`, accessed on March 2, 2005.

[Das03] Dash Optimization Inc. *Xpress-Optimizer Reference Manual.* Warwickshire, U.K., 2003.

[Epp00] David Eppstein. Spanning trees and spanners. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[GLN01] Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Approximating a minimum Manhattan network. *Nordic J. Comput.*, 8:219–232, 2001.

[KIA02] Ryo Kato, Keiko Imai, and Takao Asano. An improved algorithm for the minimum Manhattan network problem. In Prosenjit Bose and Pat Morin, editors, *Proc. 13th Annual International Symposium on Algorithms and Computation (ISAAC'02)*, volume 2518 of *Lecture Notes in Computer Science*, pages 344–356, Vancouver, 20–23 November 2002. Springer-Verlag.

[LAP03] Fumei Lam, Marina Alexandersson, and Lior Pachter. Picking alignments from (Steiner) trees. *Journal of Computational Biology*, 10:509–520, 2003.

[SS00] Weiping Shi and Chen Su. The rectilinear Steiner arborescence problem is NP-complete. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 780–787, New York, 9–11 January 2000. ACM Press.

[WBS04] Alexander Wolff, Marc Benkert, and Takeshi Shirabe. The minimum Manhattan network problem: Approximations and exact solutions. In *Proc. 20th European Workshop on Computational Geometry (EWCG'04)*, pages 209–212, Sevilla, 24–26 March 2004.

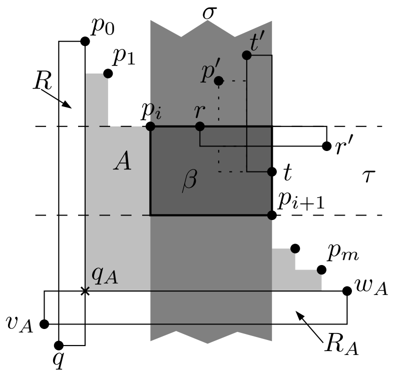


Figure 14: The area $\mathrm{int}(\tau \cap \beta)$ does not intersect any boundary segment, but a segment in $\mathcal{C}_{\text{ver}}$.

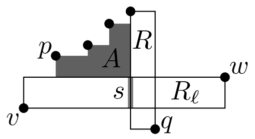


Figure 15: An impossible constellation: $(w,p) \in Z_{\text{hor}}$ excludes $(p,q) \in Z_{\text{quad}}$.

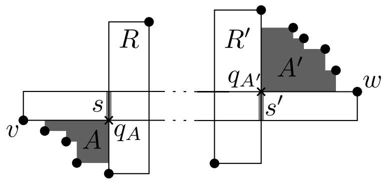


Figure 16: Not both s and s' lie in N_2.

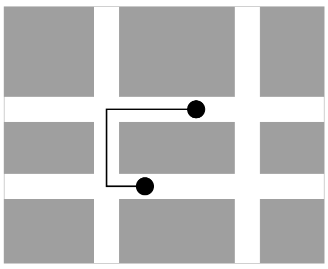


Figure 17: The *Real* Manhattan problem: a shortest path connecting two sites.

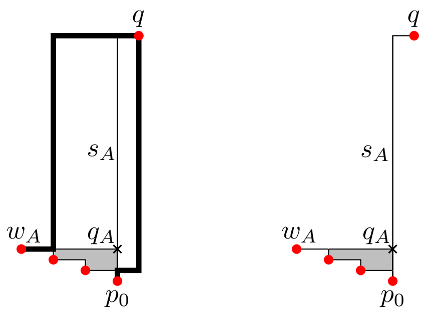


Figure 18: Left: network computed by ApproxMMN, right: an MMN of the same point set.

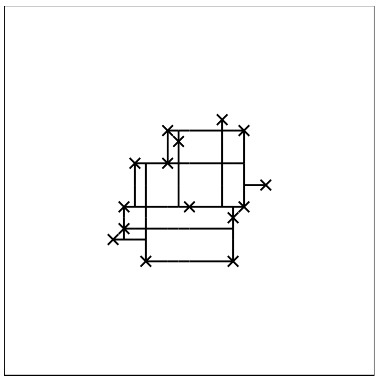


Figure 19: An MMN for a Square-01 instance with 15 points.

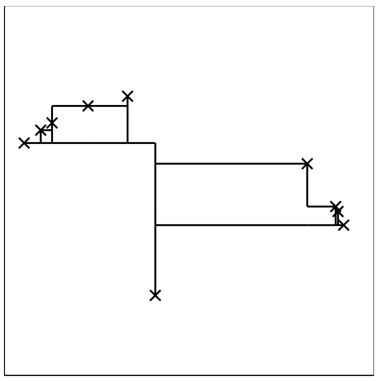
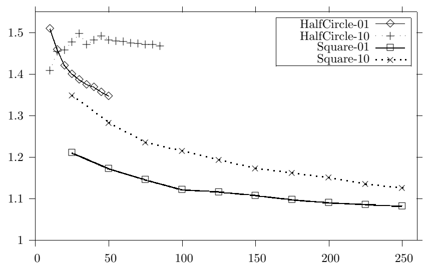
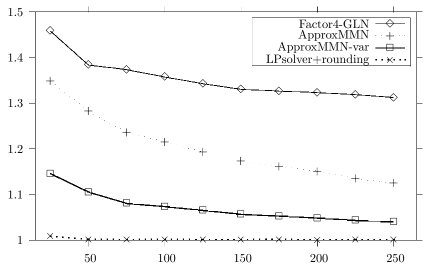


Figure 20: An MMN for a HalfCircle-02 instance with 10 points.

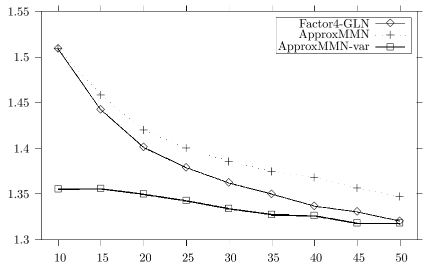


(a) ApproxMMN on various instance classes

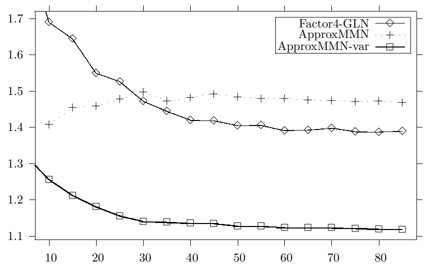


(b) Various algorithms on Square-10 instances

Figure 21: Performance of various algorithms.

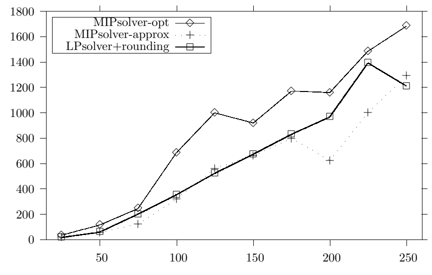


(a) HalfCircle-01 instances

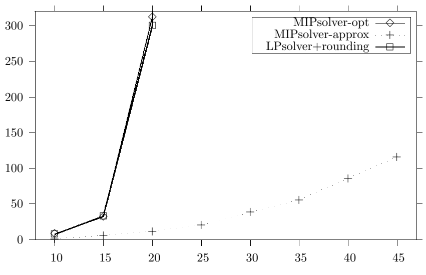


(b) HalfCircle-10 instances

Figure 22: Performance of various algorithms on HalfCircle instances.



(a) Square-10 instances



(b) HalfCircle-01 instances

Figure 23: Ratios of the running times of MIPsolver-opt, MIPsolver-approx, and LP-solver+rounding over the running time of ApproxMMN.