

APPROXIMATING A MINIMUM MANHATTAN NETWORK

JOACHIM GUDMUNDSSON

*Department of Information and Computing Sciences, Utrecht University
PO Box 80.089, 3508 TB Utrecht, The Netherlands*
joachim@cs.uu.nl

CHRISTOS LEVCOPOULOS

*Department of Computer Science, Lund University
Box 118, S-221 00 Lund, Sweden*
christos@cs.lth.se

GIRI NARASIMHAN*

*Department of Mathematical Sciences, The University of Memphis
Memphis, TN 38152, USA*
giri@msci.memphis.edu

Abstract. Given a set S of n points in the plane, we define a *Manhattan Network* on S as a rectilinear network G with the property that for every pair of points in S , the network G contains the shortest rectilinear path between them. A *Minimum Manhattan Network* on S is a Manhattan network of minimum possible length. A Manhattan network can be thought of as a graph $G = (V, E)$, where the vertex set V corresponds to points from S and a set of Steiner points S' , and the edges in E correspond to horizontal or vertical line segments connecting points in $S \cup S'$. A Manhattan network can also be thought of as a 1-spanner (for the L_1 -metric) for the points in S .

Let R be an algorithm that produces a rectangulation of a staircase polygon in time $R(n)$ of weight at most r times the optimal. We design an $O(n \log n + R(n))$ time algorithm which, given a set S of n points in the plane, produces a Manhattan network on S with total weight at most $4r$ times that of a minimum Manhattan network. Using known rectangulation algorithms, this gives us an $O(n^3)$ -time algorithm with approximation factor four, and an $O(n \log n)$ -time algorithm with approximation factor eight.

CR Classification: F.2.2, I.3.5

Key words: computational geometry, approximation algorithms, spanners

1. Introduction

A *rectilinear path* connecting two points in the plane is a path consisting of only horizontal and vertical line segments. A rectilinear path of minimum possible length connecting two points will be referred to as a *Manhattan path*,

*Funded by grants from NSF (CCR-940-9752) and Cadence Design Systems, Inc.

where the length of a rectilinear path is equal to the sum of the lengths of its horizontal and vertical line segments. Manhattan paths are monotonic. The *Manhattan distance* (or L_1 -distance) between two points in the plane is the length of the Manhattan path connecting them. In this paper we introduce the concept of geometric networks that guarantee Manhattan paths between every pair of points from a given set of points.

Consider a input set S of points in the plane. A *geometric network* on S can be modeled as an undirected graph $G = (V, E)$. The vertex set V corresponds to the points in $S \cup S'$, where S' is a set of newly added Steiner points; the edge set E corresponds to line segments joining points in $S \cup S'$. If all the line segments are either horizontal or vertical, then the network is called a *rectilinear geometric network*. Each edge $e = (a, b) \in E$ has length $|e|$ that is defined as the Euclidean distance $|ab|$ between its two endpoints a and b . The total length of a set of edges is simply the sum of the lengths of the edges in that set. The *total length* of a network $G(V, E)$ is denoted by $|E|$. For $p, q \in S$, a pq -path in G is a path in G between p and q .

For a given set S of n points in the plane, we define a *Manhattan Network* on S as a rectilinear geometric network G with the property that for every pair of points $p, q \in S$, the network G contains a Manhattan pq -path connecting them. A *Minimum Manhattan Network* on S is a Manhattan network of minimum length.

The complete grid on the point set S is clearly a Manhattan network. In other words, the network obtained by drawing a horizontal line and a vertical line through every point in S and by considering only the portion of the grid inside the bounding box of S is a network that includes the Manhattan path between every pair of points in S . It is easy to show that the minimum Manhattan network on S need not be unique and that the complete grid on the point set S can have total weight $O(n)$ times that of a minimum Manhattan network on the same point set.

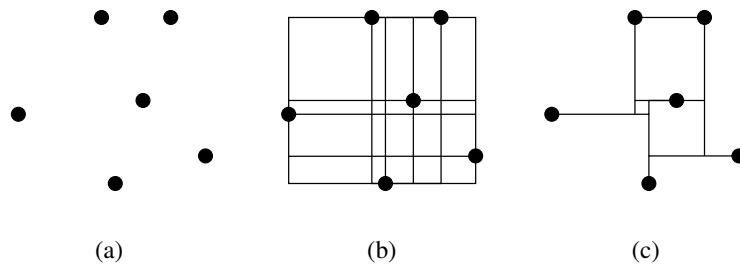


Fig. 1: (a) A set of input points, (b) A Manhattan network, and (c) A minimum Manhattan network.

Fig. 1b and c above show examples of a Manhattan network and a minimum Manhattan network on the same set of points. In fact, the network in Fig. 1b is also a complete grid on the input points.

Many VLSI circuit design applications require that a given set of terminals in the plane must be connected by networks of small total length. Rectilinear Steiner minimum trees were studied in this context. Manhattan networks impose additional constraints on the distance between the terminals in the network. The concept of Manhattan networks seems to be a rather natural concept; it is surprising that this concept has not been previously studied.

Manhattan networks are also closely connected to the concept of spanners. Given a set S of n points in the plane, and a real number $t > 1$, we say that a geometric network G is a t -spanner for S for the L_b -norm, if for each pair of points $p, q \in S$, there exists a pq -path in G of length at most t times the L_b -distance between p and q . In this connection, a minimum Manhattan network can be thought of as a sparsest 1-spanner for S for the L_1 -norm, assuming that Steiner points are allowed to be added. The distance between two points in the plane, $p = (p_x, p_y)$ and $q = (q_x, q_y)$, in some L_b -metric is defined as $(|p_x - q_x|^b + |p_y - q_y|^b)^{1/b}$. The concept of 1-spanners are also interesting since they represent the network with the most stringent distance constraints. However, note that the sparsest 1-spanner for S in the L_b -norm (for $b \geq 2$) is the trivial complete graph on S . It is also interesting to note that a Manhattan network can be thought of as a $\sqrt{2}$ -spanner (with Steiner points) for the L_2 norm. Although complete graphs represent ideal communication networks, they are expensive to build; sparse spanners represent low cost alternatives. The weight of the spanner network is a measure of its sparseness; other sparseness measures include the number of edges, maximum degree and the number of Steiner points. Spanners have applications in network design, robotics, distributed algorithms, and many other areas, and have been a subject of considerable research [Althöfer *et al.* 1993, Arya *et al.* 1995, Chandra *et al.* 1995, Das and Narasimhan 1997, Levkopoulos *et al.* 1998]. More recently, spanners have found applications in the design of approximation algorithms for problems such as the traveling salesperson problem [Arora *et al.* 1998, Rao and Smith 1998].

In this paper we present an algorithm that produces a Manhattan network for a given set S of n points in the plane. The total weight of the network output by the algorithm is within a constant factor of the minimum Manhattan network. It is interesting to note that in this paper we reduce the problem of computing an approximate minimum Manhattan network to the problem of finding a minimum-weight rectangulation of a set of staircase polygons. If the rectangulation algorithm runs in time $O(R(n))$ and produces a rectangulation that is within a factor r of the optimal, our algorithm will produce a Manhattan network of total weight $4r$ times the weight of a minimum Manhattan network in time $O(n \log n + R(n))$. Using two known approximation algorithms for the minimum-weight rectangulation problem, we obtain two algorithms for the approximate minimum Manhattan network problem. The first algorithm runs in $O(n^3)$ time and produces a Manhattan network of total weight at most four times that of a minimum Manhattan network. The second algorithm runs in $O(n \log n)$ time and has an approximation factor of eight. It is unknown whether the problem of computing

the minimum Manhattan network is a NP-hard problem. It is also unknown whether a polynomial-time approximation scheme exists for this problem.

A noteworthy feature of our result is that unlike most of the results on t -spanners, we compare the output of our algorithm to that of minimum Manhattan networks and our results involve small constants (4 or 8). Most results on sparse t -spanners prove weight bounds that compare it to the length of a minimum spanning tree; the constants involved in those results are usually very large.

In Section 3, we present the approximation algorithm. In Section 3.1 we prove that the algorithm produces a Manhattan network; in Section 3.2 we prove that the network produced is of weight at most $4r \times |E_{opt}|$, where E_{opt} is the set of edges in a minimum Manhattan network on S .

2. Definitions

Let u and v be two points in S , where u lies to the left of and above v . A \perp -path between u and v is a rectilinear path consisting of one vertical line segment with upper endpoint at u , and one horizontal line segment with right endpoint at v . Note that such a path is a minimum-weight, minimum-link path connecting u and v . The \neg -path is defined in a similar fashion, as shown in Fig. 2. If u lies to the left of and below v , we define a \lrcorner -path and a \ulcorner -path in a similar fashion. Note that each of the four paths described above introduces one Steiner point at the bend. We will denote by $[v, u]$ the closed region described by a rectilinear rectangle with corners in v and u . The coordinates of a vertex $v \in S$ are denoted $v.x$ respectively $v.y$.

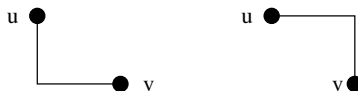


Fig. 2: (left) An \perp -path, and (right) an \neg -path.

If two different edges of the graph intersect, we will assume that their intersection defines a Steiner point.

3. The approximation algorithm

In this section we present an approximation algorithm to construct a Manhattan network $G = (V, E)$ of small total length. The algorithm will construct the edge set in four independent steps. In each step, for each vertex, the algorithm constructs a (possibly empty) local network connecting that vertex to a set of chosen “neighboring” vertices. The local network is constructed by a call to the subroutine RECTANGULATE that produces a rectangulation of a simple rectilinear polygon, i.e., it outputs a set of edges

that partitions the input polygon into rectangles. The rectangulation will be explained in more detail in Section 4. Since the four steps are almost symmetrical, the first step is explained in more detail than the later steps. We need the following definition.

DEFINITION 1. Let v_1, \dots, v_m be a set of vertices such that v_{i+1} lies above and to the right of v_i , $1 \leq i < m$. The staircase polygon obtained by adding a \sqcup -path between v_1 and v_m , and \lrcorner -paths between v_i and v_{i+1} , as shown in Fig. 3c, will be referred to as the C -hull of the set of vertices $\{v_1, \dots, v_m\}$. The C -hull is defined symmetrically if v_{i+1} lies above and to the left of v_i , below and to the right of v_i or below and to the left of v_i .

Now we are ready to state the algorithm.

- (1) Sweep the points in S from left to right. As shown in Fig. 3a, for each point $v \in S$ let $B_1(v)$ be the set of points such that for every point $v' \in B_1(v)$ it holds that v is the leftmost point below and not to the left of v' (if several, take bottommost). We say that v' 1-belongs to v . Let v_1, \dots, v_m be the vertices in $B_1(v)$ ordered from left to right, as shown in Fig. 3b. For completeness we describe the construction of the point sets. Let p_1, \dots, p_n be the points in S ordered from left to right. The algorithm processes the points from left to right. Assume that we are to process $p_j \in S$. During the sweep we maintain a list where the encountered points, not belonging to any of the sets $B_1(p_1), \dots, B_1(p_{j-1})$, are ordered according to their y -coordinate (from bottom to top). The algorithm now performs a linear search through the list until a point that does not belong to $B_1(p_j)$ is found. The points that belong to $B_1(p_j)$ are removed from the list and p_j is inserted into the list. From this it follows that the construction of these sets takes linear time plus the time needed for sorting the points. Hence, the total time for this step is $O(n \log n)$.

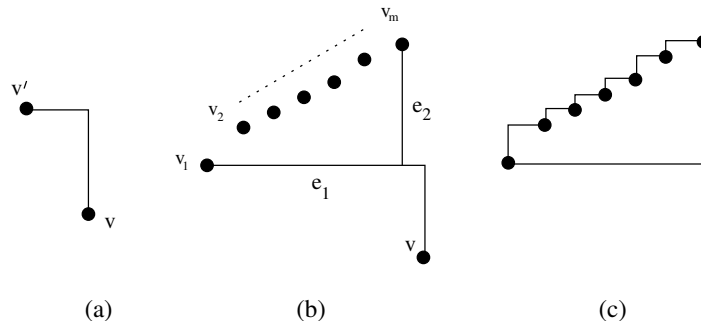


Fig. 3: (a) Vertex v' 1-belongs to v . (b) The set $B_1(v) = \{v_1, \dots, v_m\}$ 1-belongs to v . (c) The C -hull of $B_1(v)$.

Now, construct an \lrcorner -path, denoted e_1 , connecting v_1 and v . If $m > 1$, draw a vertical edge e_2 , with top endpoint at v_m and bottom endpoint on e_1 . Next a “local” Manhattan network is constructed by a call to the subroutine RECTANGULATE with the C -hull of $\{v_1, \dots, v_m\}$ as input. The set of edges constructed in this step is denoted E_1 . It should be noted that the \lrcorner -paths from v_i to v_{i+1} are not included in $E_1(v)$.

- (2) Sweep the points in S from left to right. For each point $v \in S$ let $B_2(v)$ be the set of points such that for every point $v' \in B_2(v)$ it holds that v is the leftmost point above and not to the left of v' (if several, take bottommost). We say that v' 2-belongs to v . For each vertex v let $B_2(v)$ denote the set of vertices in S that 2-belong to v , as shown in Fig. 4b. Perform the symmetrical procedure as performed in step 1 on the set $B_2(v)$ for every $v \in S$, to obtain the set of edges E_2 .
- (3) Sweep the points in S from bottom to top. For each point $v \in S$ let $B_3(v)$ be the set of points such that for every point $v' \in B_3(v)$ it holds that v is the bottommost point to the left and not below v' (if several, take leftmost). We say that v' 3-belongs to v . For each vertex v let $B_3(v)$ denote the set of vertices in S that 3-belong to v , see Fig. 4c. Perform the symmetrical procedure as performed in step 1 on the set $B_3(v)$ for every $v \in S$, to obtain the set of edges E_3 .
- (4) Sweep the points in S from top to bottom. For each point $v \in S$ let $B_4(v)$ be the set of points such that for every point $v' \in B_4(v)$ it holds that v is the topmost point to the left and not above v' (if several, take leftmost). We say that v' 4-belongs to v . For each vertex v let $B_4(v)$ denote the set of vertices in S that 4-belong to v , as shown in Fig. 4d. Perform the symmetrical procedure as performed in step 1 on the set $B_4(v)$ for every $v \in S$, to obtain the set of edges E_4 .

After building the appropriate “local” networks, we say that every vertex v is *directly connected* to the vertices in $B_1(v) \cup \dots \cup B_4(v)$. From these four sweeps we get four edge sets, E_1, \dots, E_4 . The Manhattan network is now defined as $G = (V, E)$, where $E = E_1 \cup \dots \cup E_4$, and V includes the points in S and all the Steiner points that are generated when adding the edges in E .

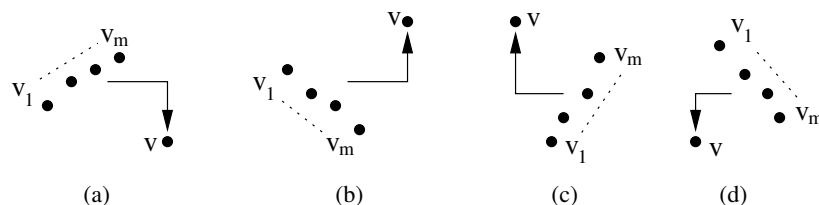


Fig. 4: The set $\{v_1, \dots, v_m\}$ (a) 1-belongs to v , (b) 2-belongs to v , (c) 3-belongs to v , and (d) 4-belongs to v .

Constructing the sets in each step of the algorithm takes $O(n \log n)$ time, hence, the time-complexity of the algorithm is $O(n \log n + R(n))$, where $R(n)$ is the time-complexity of the subroutine `RECTANGULATE`. This is straightforward since for every point $v \in S$ there is at most one point $v' \in S$, for each step of the algorithm, such that $v \in B_i(v')$.

Also note that there is some asymmetry in the above construction. This is deliberate; the asymmetric cases are required in the proof of the correctness of the algorithm (see Lemma 2).

3.1 The algorithm outputs a Manhattan network

We will show that the algorithm presented above produces a Manhattan network (no matter which subroutine `RECTANGULATE` is used by the algorithm).

LEMMA 1. *The set of edges in $E_i(v)$ is a rectilinear network that guarantees Manhattan paths from v to every vertex in $B_i(v)$.*

PROOF. Since the four steps of the algorithm are similar we may w.l.o.g. prove the lemma for only one of the directions, say $i = 1$.

Let v be an arbitrary vertex of S and let v_1, \dots, v_m be the vertices in $B_1(v)$. By step 1 of the algorithm, we note that v_j lies below and to the left of v_{j+1} , $1 \leq j < m$. Recall that v_1 and v are connected by an \sqsupset -path, e_1 , and that v_m is connected to this edge by a vertical segment corresponding to edge e_2 , see Fig. 3b. Let e'_1 be the horizontal part of e_1 between v_1 and e_2 . Our aim is to show that there is a Manhattan path between v_j and v , $1 \leq j \leq m$. This already holds for v_1 and v_m .

The lemma is easily proved by observing that every vertex $v_j \in B_1(v)$, must lie on the perimeter of a distinct rectangle (hence one should be able to proceed either down or to the right from v_j), and there always exists a monotonic rectilinear path from v_i to v that follows the borders of the rectangles encountered along the way. The set of edges constructed in the rectangulation of the C -hull plus the two edges e_1 and e_2 is the set $E_1(v)$. The lemma follows. \square

It remains to prove that the graph $G = (V, E)$ is a Manhattan network for the points in S and that $|E| \leq 4r \times |E_{opt}|$, where r is the approximation factor of the rectangulation algorithm.

To show that the algorithm outputs a Manhattan network, it suffices to prove the following lemma:

LEMMA 2. *For each pair of points $s, t \in S$ there is a Manhattan path in G connecting s and t .*

PROOF. Without loss of generality, either t lies to the right and above s , or t lies to the right and below s (if not, switch s and t). We first assume that t lies to the right and above s . Without loss of generality, we may assume

that no two points have the same x - or y -coordinate, since the algorithm always connects two such points by a line segment.

Consider $[s, t]$, the rectilinear rectangle with corners at s and t . If $s \in B_2(t)$ or $t \in B_4(s)$, then s is directly connected to t and we are done. Otherwise, we know from the algorithm that s is directly connected to a point $s_1 \in S$ above and to the right of s and to the left of t (Case 2 of construction), and that t is directly connected to a point $t_1 \in S$ below and to the left of t and above s (Case 4 of construction). We consider the following two cases:

Case 1 [s_1 or t_1 lies within $[s, t]$]: Without loss of generality, assume that s_1 lies within $[s, t]$, as shown in Fig. 5a. In this case, let s_1 be the new s and continue recursively.

Case 2 [s_1 and t_1 lie outside $[s, t]$]: Then we know that s_1 lies above and to the left of t and that t_1 lies above and to the left of s , see Fig. 5b. The Manhattan path connecting s and s_1 must intersect the Manhattan path connecting t and t_1 within $[s, t]$. Hence there is a Manhattan path connecting s and t .

The case when t lies to the right and below s uses Cases 1 and 3 of the construction, and is otherwise similar to the proof above. Hence the lemma. \square

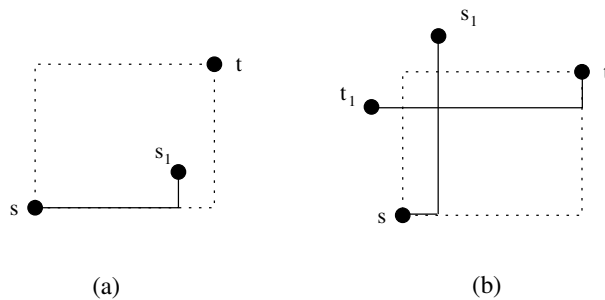


Fig. 5: There exists a Manhattan path between s and t .

3.2 Bounding the length of the network

For the length analysis, once again, it is sufficient to consider only one sweep, *i.e.*, step 1 of the algorithm. The approximation factor for this sweep is then multiplied by four to obtain the approximation factor of the algorithm. Let v be an arbitrary vertex of S . Let $B_1(v) = \{v_1, \dots, v_m\}$ and, let $V = B_1(v) \cup \{v\}$. We define a *charging area* of v (with respect to this sweep) as the region $\cup_{i=1}^m [v, v_i]$, as shown by the shaded region in Fig. 6a. The charging area is denoted by $C_1(v)$. Note that the interior of the charging area for any vertex must be empty of input points. We start with the following observation:

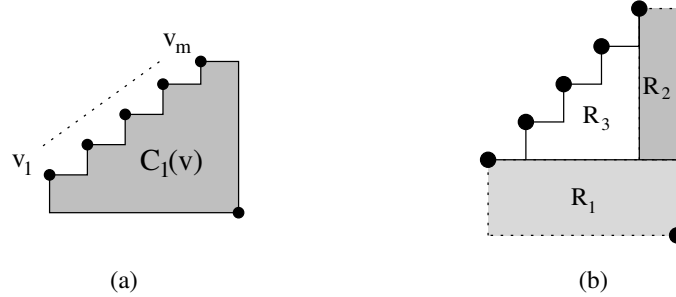


Fig. 6: (a) The charging area. (b) Partitioning the charging area into three regions.

LEMMA 3. *For every pair of vertices $v_i, v_j \in S$, the charging areas $C_1(v_i)$ and $C_1(v_j)$ are disjoint, except possibly for the point v_i or v_j .*

PROOF. Since no vertex can belong to more than one vertex, the staircase parts of the two charging areas cannot share any vertices. Thus either v_i is on the staircase part of $C_1(v_j)$ or vice versa. But then, the rest of the charging areas cannot overlap because of its shape and orientation. \square

It is important to point out that the charging areas may share a point, but cannot share an edge of the boundary. Also note that all edges of $E_1(v)$ that were added in step 1 must lie entirely within $C_1(v)$. Hence, the edges produced in step 1 connecting vertices in S cannot be used to connect vertices in any other charging areas. We will prove that $|E_1(v)|$ is at most equal to the length of the edges in E_{opt} lying within the charging area of v . Since the charging areas are disjoint this implies that the total length of the edges produced in step 1 of the algorithm, $|E_1| = \sum_{v \in S} |E_1(v)|$, is at most $|E_{opt}|$.

First, partition the charging area $C_1(v)$ into three regions, where $R_1 = [v, v_1]$, $R_2 = [v, v_m] \setminus R_1$ and R_3 is the remaining region of the charging area. The three regions are shown in Fig. 6b. Before we continue, recall that $E_1(v)$ consists of a \sqsupset -path, e_1 , connecting v_1 and v , a vertical edge e_2 connecting v_m with e_1 , and a rectangulation of the C -hull of $B_1(v)$ (which is meant to connect v_i , $2 \leq i < m$, with e_1 or e_2).

Consider a minimum Manhattan network connecting the vertices in S . What do we know about E_{opt} of such a network?

- (1) E_{opt} must include a Manhattan path between v_1 and v within R_1 . Note that this path has the same length as e_1 .
- (2) E_{opt} must include a Manhattan path between v_m and v within $[v, v_m]$. Within R_2 this path has at least the same length as e_2 .
- (3) If $m > 2$ then there must be a network N_{opt} connecting v_2, \dots, v_{m-1} with e_1 or e_2 within R_3 . Hence, it remains to prove that a minimum weight network connecting v_2, \dots, v_{m-1} to the right or bottom side of R_3 has weight equal to a minimum weight rectangulation of R_3 .

LEMMA 4. *A minimum-weight network connecting v_i , $2 \leq i < m$, with e_1 or e_2 has length at least equal to the length of a minimum-weight rectangulation of the C -hull.*

Lemma 4 is a direct consequence of the following two lemmas. Consider a grid induced by the vertices of S . Let N_{opt} be an optimal network within R_3 connecting v_2, \dots, v_{m-1} to the right or bottom side of R_3 . Let N_{rect} be a minimum-weight network, connecting v_2, \dots, v_{m-1} to the right or bottom side of R_3 , whose segments lie (only) on the grid induced by the vertices in S .

LEMMA 5. $|N_{rect}| = |N_{opt}|$.

PROOF. Let P be a Manhattan path between vertices v_i and v . Assume that the path is moving right on the grid and that it changes direction downwards without reaching an intersection of the grid. Denote by t the last intersection of the grid that the path passed. First, it is obvious that any monotone path from v_i to v lying on the grid is of equal weight. Hence, the only reason to change direction “outside” the grid is that other paths may use this segment. Since all paths that may use this segment start at vertices to the left of t , they must start from vertices v_j , where $j < i$, which cannot intersect till the next horizontal grid line below it. Thus the path can be “straightened” within that grid cell to follow the grid lines without decreasing its length. Going step by step, all paths can be modified to follow grid lines. Hence, the observation follows. \square

LEMMA 6. *There exists a minimum-weight rectangulation of the interior of the C -hull of length $|N_{rect}|$.*

PROOF. Every path between an interior point v_i and v moves (seen from v_i) only in two directions, right and down. The only case when a path would induce a non-rectangular network of the C -hull is when it turns right or down without meeting another path. Assume we follow a path from v_i going down and then turning right, without meeting a horizontal segment. In this case the path could have been shortened by not changing direction, see Fig. 7b, or by starting going right from the beginning. This is easy to see since the horizontal distance between a vertical segment and the turning point of the path is equal to the horizontal distance to the vertical segment and v_i . Hence, we do not gain anything by going downwards if the path is not meeting a horizontal segment. This means that there exists a rectangulation of the interior of the C -hull of weight $|N_{rect}|$. \square

Putting these results together we obtain the following lemma.

LEMMA 7. $\sum_{v \in S} |E_1(v)| \leq r \times |E_{opt}|$.

To obtain the approximation factor for the algorithm just multiply the approximation factor for each step by four, since there are four sweeps. We summarize this paper by giving the main theorem.

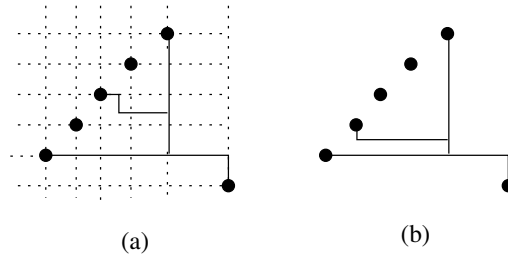


Fig. 7: Following the grid induced by the set of points does not increase the length of the network.

THEOREM 1. *Given a set of n points in the plane, and given an r -approximate, $R(n)$ -time algorithm to compute a minimum-weight rectangulation of a staircase polygon, there exists an $O(n \log n + R(n))$ -time algorithm that outputs a Manhattan network of length at most $4r$ times that of a minimum Manhattan network.*

4. The rectangulation subroutine

In the previous sections we showed that it is possible to compute a Manhattan network in time $O(n \log n + R(n))$ that outputs a Manhattan network of length at most $4r$ times that of a minimum Manhattan network, given an r -approximate, $R(n)$ -time algorithm to compute a minimum-weight rectangulation of a staircase polygon. For completeness we present two rectangulation algorithms, one optimal algorithm and one approximation algorithm.

Subroutine A: An optimal rectangulation.

An optimal rectangulation of a staircase polygon can be computed by using dynamic programming [see Lingas *et al.* 1982]. Assume that we are given a staircase polygon P . We may assume w.l.o.g. that the base of P is at the bottom and the long side of P is to the right, Fig. 8.

The key idea is that in every optimal rectangulation there exists one rectangle whose lower right corner coincides with P 's lower right corner, and whose upper left corner coincides with a step of the staircase of P . The algorithm then computes the optimal rectangulation for each step, then for each pair of steps, and so on, until the optimal solution for P has been found. The following result was shown by Lingas *et al.* [1982].

FACT 1. A minimum weight rectangulation of a staircase polygon can be computed in time $O(n^3)$ using $O(n^2)$ space.

We obtain the following corollary.

COROLLARY 1. *Given a set of n points in the plane there exists an $O(n^3)$ -time algorithm that outputs a Manhattan network of length at most 4 times that of a minimum Manhattan network.*

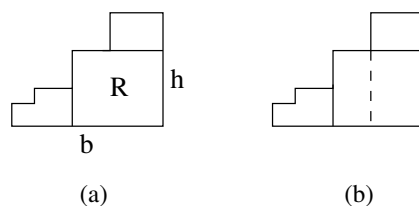


Fig. 8: (a) A thickest rectangle R within the polygon. (b) A thickest-first rectangulation is within a factor two of the optimal.

Subroutine B: A thickest-first rectangulation.

A thickest-first rectangulation algorithm is a greedy like algorithm that recursively “cuts” off a largest possible rectangle from the staircase polygon. Levcopoulos and Östlin [1996] showed that a thickest-first partition of a histogram can be computed in linear time with an approximation factor of 2.42. We will below show, that it is possible to improve the approximation factor for staircase polygons. We need the following definition.

DEFINITION 2. *We say that a rectangle R is thicker than a rectangle R' if and only if the shortest side of R is longer than the shortest side of R' . A rectangle within a rectilinear polygon P is called maximal if it is not properly included in any other rectangle lying within P . Finally, a rectangle is said to be thickest within P if there is no thicker rectangle lying within P .*

THEOREM 2. *A thickest-first rectangulation of a staircase polygon can be computed in linear time such that the weight of the added edges in the rectangulation is at most twice the weight of a minimum weight rectangulation.*

PROOF. Let P be a staircase polygon. We assume for simplicity that the base, b , of P is horizontal and at the bottom, and the longest vertical edge of P , denoted h , is the right side of P , Fig. 8a. Let R be a thickest rectangle within P and let l_R be the set of segments of R 's perimeter which are disjoint from the boundary of P . The proof is by induction. If P is a rectangle then l_R is empty and we are finished. Otherwise l_R contains one or two segments. Assume that the observation holds for the staircase polygon of P above R 's top side and the staircase polygon of P to the left of R 's left side, if they exist. To be able to use induction let S be the open region defined by the interior of R plus the segment or segments in l_R . Let OPT be the segments in an optimal rectangulation of P . By the induction hypothesis it holds that the total length of the segments produced by the thickest-first rectangulation minus the segment or segments in l_R is at most of length $2 \cdot (|OPT| - |OPT(S)|)$, where $OPT(S)$ is the set of segments in OPT intersecting S . Hence, it is enough to prove that $2|l_R| \leq |OPT(S)|$.

We will have two cases, either there are segments of $OPT(S)$ within the open region of R , or there are not.

1. No segments intersect the open region of R . In this case there must be segments of $OPT(S)$ that are equal to the segments in l_R , otherwise OPT would not be a rectangulation of P . Hence, the total length of the segments in l_R is equal to the length of $OPT(S)$.

2. Otherwise, if there are segments of $OPT(S)$ within the open region of R , we know that there must be segments in $OPT(S)$ of length at least equal to the shortest side of R . It is easy to see that every segment in l_r is shorter than the shortest side of R , otherwise R would not have been a thickest rectangle. It follows that the total length of the segments in l_r is less than two times the total length of the segments in $OPT(S)$. \square

We obtain the following corollary.

COROLLARY 2. *Given a set of n points in the plane there exists an algorithm that in $O(n \log n)$ -time outputs a Manhattan network of length at most 8 times that of a minimum Manhattan network.*

5. Open problems

The following problems remain open:

- (1) Determine the complexity of the problem of computing minimum Manhattan networks.
- (2) Design a 2-approximate algorithm for the problem of computing minimum Manhattan networks.
- (3) Design a PTAS for the problem of computing minimum Manhattan networks.

Acknowledgements

The authors would like to thank the referees for the Nordic Journal of Computing for their helpful comments and suggestions. The last author would like to thank Dr. Lubomir Soltes for useful discussions and an implementation of the algorithm presented in this paper.

References

- ALTHÖFER, I., DAS, G., DOBKIN, D. P., JOSEPH, D., AND SOARES, J. 1993. On sparse spanners of weighted graphs. *Discrete Computational Geometry*, 9, 81–100.
- ARORA, S., GRIGNI, M., KARGER, D., KLEIN, P., AND WOLOSZYN, A. 1998. A Polynomial-Time Approximation Scheme for Weighted Planar Graph TSP. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 33–41.
- ARYA, S., DAS, G., MOUNT, D. M., SALOWE, J. S., AND SMID, M. 1995. Euclidean spanners: short, thin, and lanky. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, 489–498.
- CHANDRA, B., DAS, G., NARASIMHAN, G., AND SOARES, J. 1995. New sparseness results on graph spanners. *International Journal of Computational Geometry & Applications* 5, 1, 125–144.

- DAS, G. AND NARASIMHAN, G. 1997. A Fast Algorithm for Constructing Sparse Euclidean Spanners. *International Journal of Computational Geometry & Applications* 7, 4, 297–315.
- LEVCOPOULOS, C., NARASIMHAN, G., AND SMID, M. 1998. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proceedings of the 30th Annual ACM Symposium on Theory Computing*, 186–195.
- LEVCOPOULOS, C. AND ÖSTLIN, A. 1996. Linear-Time Heuristics for Minimum Weight Rectangulation. In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT '96)*, Volume 1097 of Lecture Notes in Computer Science. Springer-Verlag, 296–308.
- LINGAS, A., PINTER, R., RIVEST, R., AND SHAMIR, A. 1982. Minimum edge length partitioning of rectilinear polygons. In *Proceedings of the 20th Annual Allerton Conference on Communication, Control and Computation*, 53–63.
- RAO, S. B. AND SMITH, W. D. 1998. Approximating Geometrical Graphs via “Spanners” and “Banyans”. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 540–550.