Subexponential Algorithms for Rectilinear Steiner Tree and Arborescence Problems*

Fedor Fomin[1], Sudeshna Kolay[2], Daniel Lokshtanov[3], Fahad Panolan[4], and Saket Saurabh[5]

1 **University of Bergen, Norway**
`fedor.fomin@ii.uib.no`
2 **Institute of Mathematical Sciences, India**
`skolay@imsc.res.in`
3 **University of Bergen, Norway**
`daniello@ii.uib.no`
4 **University of Bergen, Norway**
`fahad.panolan@ii.uib.no`
5 **University of Bergen, Norway; and Institute of Mathematical Sciences, India**
`saket@imsc.res.in`

Abstract

A *rectilinear Steiner tree* for a set T of points in the plane is a tree which connects T using horizontal and vertical lines. In the Rectilinear Steiner Tree problem, input is a set T of n points in the Euclidean plane ($\mathbb{R}^2$) and the goal is to find an rectilinear Steiner tree for T of smallest possible total length. A *rectilinear Steiner arborecence* for a set T of points and root $r \in T$ is a rectilinear Steiner tree S for T such that the path in S from r to any point $t \in T$ is a shortest path. In the Rectilinear Steiner Arborescense problem the input is a set T of n points in $\mathbb{R}^2$, and a root $r \in T$, the task is to find an rectilinear Steiner arborescence for T, rooted at r of smallest possible total length. In this paper, we give the *first* subexponential time algorithms for both problems. Our algorithms are deterministic and run in $2^{\mathcal{O}(\sqrt{n}\log n)}$ time.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity.

Keywords and phrases Rectilinear graphs, Steiner arborescence, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.39

1 Introduction

In the Steiner Tree problem we are given as input a connected graph G, a non-negative weight function $w : E(G) \to \{1, 2, \ldots, W\}$, and a set of terminal vertices $T \subseteq V(G)$. The task is to find a minimum-weight connected subgraph of G, which is a tree, containing all terminal nodes T. Steiner Tree is one of the central and best-studied problems in Computer Science, we refer to the books of Hwang, Richards, and Winter [13] and Prömel and Steger [19] for thorough introductions to the problem.

In this paper we give the first *subexponential* algorithm for an important geometric variant of Steiner Tree, namely Rectilinear Steiner Tree. Here, for a given set of terminal points T in the Euclidean plane with ℓ_1-norm, the goal is to construct a network of minimum

* This work was partially supported by the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992.

length connecting all points in T. This variant of the problem is extremely well studied, see Chapter 3 of the recent book of Brazil and Zachariasen [2] for an extensive overview of various applications of Rectilinear Steiner Tree.

For the purposes of this paper, it is convenient to define Rectilinear Steiner Tree as the Steiner Tree problem on a special class of graphs called Hanan grids. Recall that for two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the Euclidean plane $\mathbb{R}^2$, the *rectilinear* (ℓ_1, *Manhattan* or *taxicab*) distance between p_1 and p_2 is $d_1(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$.

▶ **Definition 1** (Hanan grid [11])**.** Given a set T of n terminal points in the Euclidean plane $\mathbb{R}^2$, the *Hanan grid* G of T is defined as follows. The vertex set $V(G)$ of G is the set of intersection points obtained by drawing a horizontal line (line parallel to x-axis) and a vertical line (line parallel to y-axis) through each point of T. For every $u, v \in V(G)$, there is an edge between u and v in G, if and only if u and v are adjacent along a horizontal or vertical line; the weight of edge uv is the rectilinear distance between u and v. For a Hanan grid G we define a weight function $\mathsf{recdist}_G$ from the edge set $E(G)$ to $\mathbb{R}$ such that for an edge $uv \in E(G)$, $\mathsf{recdist}_G(uv) = d_1(u, v)$. If the graph G is clear from the context we drop the subscript from $\mathsf{recdist}_G$ and only use $\mathsf{recdist}$.

Let us note that when G is the Hanan grid of a set T of n points, then $T \subseteq V(G)$, $|V(G)| \leq n^2$, and for every $u, v \in V(G)$, the weight of a shortest path between u and v is equal to $d_1(u, v)$. For an edge $uv \in E(G)$, we say that uv is a *horizontal (vertical) edge* if both points u and v are on the same horizontal (vertical) line.

It was shown by Hanan [11] that the Rectilinear Steiner Tree problem can be defined as the following variant of Steiner Tree.

Rectilinear Steiner Tree
Input: A set T of n terminal points, the Hanan grid G of T and $\mathsf{recdist}_G$.
Output: A minimum Steiner tree for T in G.

Previous work on Rectilinear Steiner Tree. Though the Rectilinear Steiner Tree problem is a very special case of the Steiner Tree problem, the decision version of the problem is known to be NP-complete [9]. A detailed account of various algorithmic approaches applied to this problem can be found in books of Brazil and Zachariasen [2] and Hwang, Richards, and Winter [13]. In particular, several exact algorithms for this problem can be found in the literature. The classic algorithm of Dreyfus and Wagner [5] from 1971 solves Steiner Tree on general graphs in time $3^n \cdot \log W \cdot |V(G)|^{\mathcal{O}(1)}$, where W is the maximum edge weight in G. For Rectilinear Steiner Tree, an adaptation of Dreyfus-Wagner algorithm provides an algorithm of running time $\mathcal{O}(n^2 \cdot 3^n)$. The survey of Ganley [7] summarizes the chain of improvements based on this approach concluding with the $\mathcal{O}(n^2 \cdot 2.62^n)$-time algorithm of Ganley and Cohoon [8].

Thobmorson et al. [21] and Deneen et al. in [4] gave randomized algorithms with running time $2^{\mathcal{O}(\sqrt{n} \log n)}$ for the special case of Rectilinear Steiner Tree when the terminal points T are drawn from a uniform distribution on a rectangle.

It is also worth mentioning relevant parameterized algorithms for Steiner Tree on general graphs. Fuchs et al. [6] provide an algorithm with run time $\mathcal{O}((2+\varepsilon)^n |V(G)|^{f(1/\varepsilon)} \log W)$. Björklund et al. [1] and Nederlof [16] gave $2^n |V(G)|^{\mathcal{O}(1)} \cdot W$ time algorithms for Steiner Tree. Let us remark that, since the distances between adjacent vertices in Hanan grid can be exponential in n, the algorithms of Björklund et al. and of Nederlof do not outperform the Dreyfus-Wagner algorithm for the Rectilinear Steiner Tree problem. Interesting recent developments also concern Steiner Tree on planar graphs, and more generally, on graphs

of bounded genus. While the existence of algorithms running in time subexponential in the number of terminals on these graph classes is still open, Pilipczuk et al. [17, 18] showed that Steiner Tree can be solved in time subexponential in the size of the Steiner tree on graphs of bounded genus.

In spite of the long history of research on Rectilinear Steiner Tree and Steiner Tree, whether Rectilinear Steiner Tree can be solved in time subexponential in the number of terminals remained open. In this paper we give the first such algorithm. The running time of our algorithm is $2^{\mathcal{O}(\sqrt{n}\log n)}$. Further, our techniques also yield the first subexponential algorithm for the related Rectilinear Steiner Arborescence problem.

▶ **Definition 2.** Let G be a graph, $T \subseteq V(G)$ a set of terminals, and $r \in T$ be a root vertex. A *Steiner arborescence of T in G* is a subtree $H \subseteq G$ rooted at r with the following properties:

- H contains all vertices of T, and
- For every vertex $t \in T \setminus \{r\}$, the unique path in H connecting r and t is also the shortest r–t path in G.

Let us note that if H is a Steiner arborescence of T in G, then for every vertex $v \in V(H)$, the unique path connecting r and v in H is also a shortest r–v path in G. The Rectilinear Steiner Arborescence problem is defined as follows.

Rectilinear Steiner Arborescence
Input: A set T of n terminal points, the Hanan grid G of T, a root $r \in T$ and $\mathsf{recdist}_G$.
Output: A minimum length Steiner arborescence of T.

Rectilinear Steiner Arborescence was introduced by Nastansky, Selkow, and Stewart [15] in 1974. Interestingly, the complexity of the problem was open until 2005, when Shu and Su [20] proved that the decision version of Rectilinear Steiner Arborescence is NP-complete. No subexponential algorithm for this problem was known prior to our work.

Our method. Most of the previous exact algorithms for Rectilinear Steiner Tree exploit Hwang's theorem [12], which describes the topology of so-called full rectilinear trees. Our approach here is entirely different. The main idea behind our algorithms is inspired by the work of Klein and Marx [14], who obtained a subexponential algorithm for Subset Traveling Salesman Problem on planar graphs. The approach of Klein and Marx was based on the following two steps: (1) find a locally optimal solution such that its union with some optimal solution is of bounded treewidth, and (2) use the first step to guide a dynamic program. While our algorithm follows this general scheme, the implementations of both steps for our problems are entirely different from [14].

We give a high level description of the algorithm for Rectilinear Steiner Tree. The algorithm for Rectilinear Steiner Arborescence is similar. In the first step we build in polynomial time a (possibly non-optimal) solution. The constructed tree $\widehat{S}$ can be seen as a "shortest path" rectilinear Steiner tree. The property of $\widehat{S}$ which is crucial for the algorithm is that there is an optimal Steiner tree S_{opt} such that graph $S = \widehat{S} \cup S_{\mathsf{opt}}$ is of treewidth $\mathcal{O}(\sqrt{n})$. For the second step we have $\widehat{S}$ at hand and know that there exists a subgraph S of G of treewidth $\mathcal{O}(\sqrt{n})$, which contains an optimal Steiner tree S_{opt} and $\widehat{S}$. Note that we only know that such a subgraph S exists, albeit with the extra information that $\widehat{S} \cup S_{\mathsf{opt}} \subseteq S$. It turns out that this is sufficient in order to mimic the dynamic programming algorithm for bounded treewidth, to solve Rectilinear Steiner Tree in time $2^{\mathcal{O}(\sqrt{n}\log n)}$.

Recall the dynamic programming algorithm for Steiner Tree on a rooted tree decomposition $\mathcal{T} = (\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ of the input graph, see e.g. [3, Theorem 7.8]. For each

node $t \in V(\mathbb{T})$, let V_t be the union of vertices contained in all bags corresponding to nodes of the subtree of $\mathbb{T}$ rooted at t and let S_t be the subgraph induced by V_t. Then, in the dynamic programming algorithm, for each t we store a set of states, capturing the interaction between a minimal Steiner tree and subgraph S_t; in particular the weight of a tree and the information about its connected components in S_t. It is possible to ensure that all the information carried out in each state is "locally" defined, i.e., the information can be encoded by the elements of the bag X_t only. Therefore, at the root node, there is a state that corresponds to an optimal Steiner tree.

In our algorithm, we define *types*, which are analogous to the states stored at a node of a tree decomposition. A type stores all the information of its corresponding state. Since we do not know the tree decomposition $\mathcal{T}$, a type stores more "local" information, to take care of the lack of definite information about S. We guess some structural information about the virtual tree decomposition $\mathcal{T}$ of S. For example, we guess the height h of the rooted tree $\mathbb{T}$. In a rooted tree decomposition, the level of a node t is defined by the height of the subtree rooted at t. In our algorithm, we generate types over h levels. The intuition is that, for a node $t \in \mathbb{T}$ of level h', for each state, of t, that was required for the dynamic programming over $\mathcal{T}$, there is an equivalent type generated in level h' of our algorithm. This implies that, at level h, there is a type equivalent to a state that corresponds to an optimal Steiner tree in S. In fact, we show that any Steiner tree corresponds to exactly one type $\widehat{D}$. During the iterative generation of types, the type $\widehat{D}$ may be generated many times. One such generation corresponds to an optimal solution. So the final step of the algorithm involves investigating all the occurrences of type $\widehat{D}$ in the iterative generation, and finding the weight of a minimum Steiner tree. As in dynamic programming, a backtracking step will enable us to retrieve a minimum Steiner tree of S and therefore of G. Due to paucity of space, many proofs have been ommitted from this version.

2 Preliminaries

For a positive integer n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For point u in the Euclidean plane $\mathbb{R}^2$, its position is denoted by the coordinates (u_x, u_y). For a set V, a partition $\mathcal{P}$ of V is a family of subsets of V, such that the subsets are pairwise disjoint and the union of the subsets is V. Each subset of a partition is called a block. Given two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$, the join operation results in the partition $\mathcal{P}$, that is the most refined partition of V such that each block of $\mathcal{P}_1$ and $\mathcal{P}_2$ is contained in a single block of $\mathcal{P}$. The resultant partition $\mathcal{P}$ is often denoted as $\mathcal{P}_1 \sqcup \mathcal{P}_2$. Given a block B of $\mathcal{P}$, by $\mathcal{P} \setminus B$ denotes removing the block B from $\mathcal{P}$.

Given a graph G, its vertex and edge sets are denoted by $V(G)$ and $E(G)$ respectively. For a vertex $v \in V(G)$, the degree of a vertex, denoted as $\mathsf{degree}_G(v)$, is the number of edges of $E(G)$ incident with v. Given a vertex subset $V' \subseteq V(G)$, the induced subgraph of V', denoted by $G[V']$, is the subgraph of G, with V' as the vertex set and the edge set defined by the set of edges that have both endpoints in V'. An edge between two vertices u, v is denoted as uv. A path, where vertices $\{u_1, u_2, \ldots, u_\ell\}$ appear in sequence, is denoted by $u_1u_2 \cdots u_\ell$. Similarly, a path, where vertices u and v are the endpoints, is called a u–v path. Given a path P its non endpoint vertices are referred to as internal vertices. For an edge uv, an edge contraction in G results in a graph G' defined as follows. The vertex set $V(G') = V(G) \setminus \{u, v\} \cup v_{new}$, where v_{new} is a new vertex. The edge set $E(G') = E(G[V(G) \setminus \{u, v\}]) \cup \{wv_{new} | wu \in E(G) \vee wv \in E(G)\}$. By $G' \leq_s G$, we mean that the graph G' is a subgraph of G. Given a weight function $w : E(G) \to \mathbb{R}$, for a subgraph

H of G, we use $w(H)$ to denote the number $\sum_{e \in E(H)} w(e)$. Furthermore, for two vertices s and t in $V(G)$, by the term *shortest path* between s and t we mean the shortest path with respect to the weight function w. Given two subgraphs G_1, G_2 of G, a shortest path between G_1 and G_2 is a path P between a vertex $u \in V(G_1)$ and a vertex $v \in V(G_2)$ such that, among the shortest paths for each possible pair $\{u' \in V(G_1), v' \in V(G_2)\}$, P has minimum length. Given two graphs G_1 and G_2, the union graph $G_1 \cup G_2$ has $V(G_1 \cup G_2) = V(G_1) \cup V(G_2)$, while the edge set $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$.

Treewidth. Let G be a graph. A *tree-decomposition* of a graph G is a pair $(\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$, where $\mathbb{T}$ is a tree whose every node $t \in V(\mathbb{T})$ is assigned a subset $X_t \subseteq V(G)$, called a bag, such that the following conditions hold.

- $\bigcup_{t \in V(\mathbb{T})} X_t = V(G)$,
- for every edge $xy \in E(G)$ there is a $t \in V(\mathbb{T})$ such that $\{x, y\} \subseteq X_t$, and
- for any $v \in V(G)$ the subgraph of $\mathbb{T}$ induced by the set $\{t \mid v \in X_t\}$ is connected.

The *width* of a tree decomposition is $\max_{t \in V(\mathbb{T})} |X_t| - 1$. The *treewidth* of G is the minimum width over all tree decompositions of G and is denoted by $\mathsf{tw}(G)$.

A tree decomposition $(\mathbb{T}, \mathcal{X})$ is called a *nice tree decomposition* if $\mathbb{T}$ is a tree rooted at some node r where $X_r = \emptyset$, each node of $\mathbb{T}$ has at most two children, and each node is of one of the following kinds:

- **Introduce node**: a node t that has only one child t' where $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$.
- **Introduce edge node** a node t labeled with an edge uv, with only one child t' such that $\{u, v\} \subseteq X_{t'} = X_t$. This bag is said to introduce uv.
- **Forget vertex node**: a node t that has only one child t' where $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$.
- **Join node**: a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.
- **Leaf node**: a node t that is a leaf of $\mathbb{T}$, and $X_t = \emptyset$.

We additionally require that every edge is introduced exactly once. One can show that a tree decomposition of width t can be transformed into a nice tree decomposition of the same width t and with $\mathcal{O}(t|V(G)|)$ nodes, see e.g. [3].

Planar Graph Embeddings and Minors. A graph is *planar* if can be embedded in the plane. That is, it can be drawn on the plane in such a way that its edges intersect only at their endpoints. Formally, a planar embedding Π of a graph G consists of an injective mapping $\Pi : V(G) \to \mathbb{R}^2$ and a mapping Π of edges $uv \in E(G)$ to simple curves in $\mathbb{R}^2$ that join $\Pi(u)$ and $\Pi(v)$. Also, for $e, f \in E(G)$, $\Pi(e) \cap \Pi(f)$ contains only the images of common end vertices and for $e \in E(G)$ and $v \in V(G)$, $\Pi(v)$ is not an internal point of $\Pi(e)$. Now we define the notion of a minor of a graph G.

▶ **Definition 3.** A graph H is a minor of a graph G, denoted as $H \leq_m G$, if it can be obtained from a subgraph of G by a sequence of edge contractions.

Notice that this implies that H can be obtained from G by a sequence of vertex deletions, followed by a sequence of edge deletions and finally a sequence of edge contractions. We will need the following folklore observation.

▶ **Observation 4.** Suppose G, H are connected graphs such that H is a minor of G. Then H can be obtained from G only by edge deletions and contractions.

We also will be using the notion of a minor model.

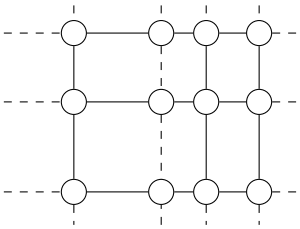

Figure 1 The union of solid edges define a subgrid of a grid.

▶ **Definition 5.** Let G and H be two connected graphs, and $H \leq_m G$. A *minor model* or simply a *model* of a graph H is a collection of pairwise disjoint vertex subsets $\mathcal{P}(H) = \{C_v \subseteq V(G) \mid v \in V(H)\}$ such that,

(a) $V(G) = \biguplus_{v \in V(H)} C_v$,

(b) for each $v \in V(H)$, $G[C_v]$ is connected, and

(c) for any $uv \in E(H)$, there exists $w \in C_u$ and $w' \in C_v$ such that $ww' \in E(G)$.

▶ Remark. It is important to point out that in general the definition of minor model does not demand that the vertex sets in $\mathcal{P}(H) = \{C_v \subseteq V(G) \mid v \in V(H)\}$ form a partition of $V(G)$. However, when both G and H are connected one can easily show that even this extra property can be assumed.

Grids and subgrids play an important role in this article. For a subset $W \subseteq [n]$, by $\max W$ ($\min W$) we denote the maximum (minimum) element of W.

▶ **Definition 6.** Let n, m be two positive integers. An $n \times m$ grid is a graph G such that $V(G) = \{v_{i,j} \mid i \in [n], j \in [m]\}$ and $E(G) = \{v_{ij}v_{i'j'} \mid |i - i'| + |j - j'| = 1\}$. For any $i \in [n]$, we call $\{v_{i1}, \ldots, v_{im}\}$ to be the i-th row of the grid G and for any $j \in [m]$, we call $\{v_{1j}, \ldots, v_{nj}\}$ to be the j-th a column of the grid G. The vertices in the first row, n-th row, the first column and m-th columns are called the *boundary vertices* of the grid. The vertices that are not boundary vertices are called *internal vertices*.

The graph H is called a subgrid of G, if there exist subsets $R \subseteq [n], C \subseteq [m]$ such that $V(H) = \{v_{ij} \in V(G) \ : \ (\min R \leq i \leq \max R) \wedge (\min C \leq j \leq \max C) \wedge (i \in R \vee j \in C)\}$ and $E(H) = \{v_{ij}v_{i'j'} \in E(G) \ : \ v_{ij}, v_{i'j'} \in V(H) \wedge (i = i' \in R \vee j = j' \in C)\}$. The set of vertices $\{v_{ij} \in V(H) \ : \ i \notin \{\min R, \max R\} \vee j \notin \{\min C, \max C\}\}$ are called the internal vertices of H. The set of vertices $\{v_{ij} \in V(H) \ : \ i \in R \wedge j \in C\}$ are called cross vertices. Finally, the set of vertices $\{v_{ij} \in V(H) \ : \ i \notin R \vee j \notin C\}$ are called subdivision vertices of H. (See Figure 1).

Given a planar graph G with an embedding Π, we call the vertices of the outer face as boundary vertices and all other vertices as internal vertices.

▶ **Definition 7.** Let G be a planar graph with a planar embedding Π, and C be a simple cycle of G. Let p_∞ be a point in the outer face of G in the embedding Π. Then removal of C from $\mathbb{R}^2$ divides the plane into two regions. The region that does not contain the point p_∞ is called the *internal region* of C, and the region containing p_∞ is called the *outer/external region* of C. A vertex in $V(G)$ is called internal if it lies in the internal region of C, and external if it lies in the external region of C. An edge in $E(G)$ is called an external edge if there is at least one point on its curve that lies in the external region. It is called an internal edge if there is at least one point on its curve that lies in the internal region.

By definition of a planar embedding, an edge of $V(G)$ can be exactly one of the three kinds: an edge of C, and external edge or an internal edge. Similarly a vertex can be exactly one of the three kinds: a vertex of C, an external vertex or an internal vertex.

▶ **Observation 8.** Let G be a planar graph with a planar embedding Π in $\mathbb{R}^2$. Let p_∞ be a point in the outer face of Π. Let H be a minor of G, and $\mathcal{P}(H) = \{C_v | v \in V(H)\}$ be a minor model of H. Then H is a planar graph. Furthermore, a planar embedding Π' of H can be obtained from Π that satisfies the following properties:

- Every vertex $v \in H$ is positioned in the place of a vertex in C_v.
- The point p_∞ is on the outer face of Π'.

We call such a planar embedding Π' as the *embedding derived* from Π.

On the proof of the first step of the algorithm, we will use the following auxiliary lemma.

▶ **Lemma 9.** *Let G and H be two connected planar graphs such that $H \leq_m G$ and let $\mathcal{P}(H) = \{C_v | v \in V(H)\}$ be a minor model of H in G. Let also Π' be an embedding of H derived from a planar embedding Π of G. Suppose that H contains an induced subgraph H' isomorphic to a 3×3 grid. Let C' be the cycle formed by boundary vertices of H' and let v be the vertex of H' in the internal region of C'. Then there is a simple cycle C in G, such that:*

1. $V(C) \subseteq \bigcup_{u \in V(H'); u \neq v} C_u$.
2. *For each vertex $w \in G$ that is contained in the internal region of C in Π, there is a vertex $u \in H'$ with $w \in C_u$.*
3. *All vertices of C_v are completely contained in the internal region of C in Π.*
4. *There is a vertex $w \in C_v$ such that $\mathsf{degree}_G(w) \geq 3$.*

Our proof will also need the following Planar grid-minor theorem.

▶ **Proposition 10** ([10])**.** *Let t be a nonnegative integer. Then every planar graph G of treewidth at least $9t/2$ contains a $t \times t$ grid as a minor.*

Properties of shortest paths in the Hanan grid. Let G be the Hanan grid of a set of n points P. For a subgraph H of G and $v \in V(H)$, we say that v is a *bend vertex* if there exists at least one horizontal edge and at least one vertical edge from $E(H)$ incident with the vertex v. A path $R = u_1 \cdots u_\ell$, between u_1 and u_ℓ in G, is called a *monotone path* if there exists $i \in [\ell]$ such that the points $u_1, \ldots, u_i$ belong to a horizontal line and $u_i, \ldots, u_\ell$ belong to a vertical line or vice-versa. In other words, all the horizontal edges as well as all the vertical edges in R are contiguous.

The following observation contains some simple facts about monotone paths.

▶ **Observation 11.** Let u and v be two vertices of a Hanan grid G. Then,

(a) There is at least one and at most 2 monotone u–v paths,

(b) If the x-coordinates of u and v are equal, then there is only one monotone u–v path and all the edges in this path are vertical. Similarly, if the y-coordinates of u and v matches, the unique monotone u–v path consists of horizontal edges only.

(c) If there are two monotone paths between u and v, then one path has a horizontal edge incident with u and the other path has a vertical edge incident with u.

▶ **Definition 12.** Suppose we are given a Hanan grid G of a set of terminals T and two vertices $u, v \in V(G)$. Let $x_1 = \min\{u_x, v_x\}$, $x_2 = \max\{u_x, v_x\}$, $y_1 = \min\{u_y, v_y\}$, and $y_2 = \max\{u_y, v_y\}$. Let $V' = \{w \in V(G) | w_x \in [x_1, x_2], w_y \in [y_1, y_2]\}$. Then $G' = G[V']$, the subgraph of G induced by V', is called a grid defined by the two vertices u, v as its diagonal points.

▶ **Observation 13.** Given a Hanan grid G, a shortest path between any two vertices u, v has the property that the sequence of the x-coordinates of the vertices of the path is a monotone sequence and the sequence of their y coordinates is also a monotone sequence.

▶ **Observation 14.** Given a grid G, all shortest paths between two vertices u, v are contained in the grid $G' \leq_s G$ that is defined by u, v as its diagonal points. In fact, any path, with the property that the sequence of the x-coordinates of the vertices of the path is a monotone sequence and the sequence of their y coordinates is also a monotone sequence, and which is fully contained inside G', is a shortest path between u and v.

3 Subexponential Algorithm for Rectilinear Steiner Tree

In this section we give a subexponential algorithm for Rectilinear Steiner Tree. Let T be an input set of terminals (points in $\mathbb{R}^2$), $|T| = n$, and G be the Hanan grid of T. Furthermore, let $\mathsf{recdist}_G$ denote the weight function on the edge set $E(G)$. For brevity we will use $\mathsf{recdist}$ for $\mathsf{recdist}_G$. Our algorithm is based on a dynamic programming over vertex subsets of size $\mathcal{O}(\sqrt{n})$ of G. To reach the stage where we can apply the dynamic programming algorithm we do as follows. First, we define a rectilinear Steiner tree, called *shortest path RST*, and describe some of its properties. Next, we show that for a shortest path RST $\widehat{S}$, there is an optimal Steiner tree S_{opt} such that $\widehat{S} \cup S_{\mathsf{opt}}$ has bounded treewidth. Finally, keeping a hypothetical tree decomposition of $\widehat{S} \cup S_{\mathsf{opt}}$ in mind, we design a dynamic programming algorithm to obtain the size of a minimum rectilinear Steiner tree of G.

3.1 Shortest Path RST and its properties

In this part, we define a shortest path RST for a set $T = \{t_1, \ldots, t_n\}$ of input terminals and prove some useful properties of such a Steiner tree. We define a *shortest path RST* as follows.

Let G be the Hanan grid of T. We define a shortest path RST $\widehat{S}$ through the following constructive greedy process. Initially, we set S_1 to the graph $(\{t_1\}, \emptyset)$, which is a rectilinear Steiner tree of $\{t_1\}$. In the i^{th} step, we compute a rectilinear Steiner tree S_{i+1} of $\{t_1, \ldots t_{i+1}\}$ from S_i as follows. If $t_{i+1} \in V(S_i)$, then we set $S_{i+1} = S_i$. Otherwise, let v be a vertex in S_i such that $\mathsf{recdist}(v, t_{i+1}) = \min_{u \in V(S_i)} \mathsf{recdist}(u, t_{i+1})$. If there is only one monotone t–v path, then let Q be this path. Otherwise there are two monotone t–v paths such that one path has a horizontal edge incident with v and the other has a vertical edge incident with v. If there is a horizontal edge in S_i which is incident with v, then we choose Q to be the monotone t–v path such that the edge in Q incident with v is a horizontal edge. Otherwise we choose Q to be the monotone t–v path such that the edge in Q incident with v is a vertical edge. Then we construct S_{i+1} by adding a monotone path Q to S_i. After $n-1$ iterations, we construct a tree $\widehat{S} = S_n$ of G, which is a Steiner tree of T. This is our shortest path RST. It easy to see that one can construct a shortest path RST in polynomial time.

▶ **Lemma 15.** *Given a set T of terminal points and the Hanan grid G of T, a shortest path RST $\widehat{S}$ of T can be constructed in polynomial time.*

Next we give an upper bound on the number of bend vertices in a shortest path RST.

▶ **Lemma 16.** *The number of bend vertices in $\widehat{S}$ is at most n.*

3.2 Supergraph of an optimal RST with bounded treewidth

In this section we view the Hanan grid G as a planar graph and use this viewpoint to obtain the required upper bound on the treewidth of a subgraph of G. In particular, given a shortest

path RST $\widehat{S}$, we show the existence of an optimal Steiner tree S_{opt} such that the treewidth of $\widehat{S} \cup S_{\mathsf{opt}}$ is sublinear in the number of terminal points T. First, we show that there is an optimal Steiner tree in G that has a bounded number of bends.

▶ **Lemma 17.** *Let T be a set of n points in $\mathbb{R}^2$ and G be the Hanan grid of T. Then there is an optimal rectilinear Steiner tree of T such that the number of bend vertices in the rectilinear Steiner tree is at most $3n$.*

With respect to a shortest path RST $\widehat{S}$, of G, we prove the next Lemma. In particular we show that the treewidth of $S = \widehat{S} \cup S_{\mathsf{opt}}$ is at most $41\sqrt{n}$. Here, S_{opt} is a carefully chosen optimal Steiner tree for T. In order to get the desired upper bound on the treewidth of S, we show that it does not contain $\mathcal{O}(\sqrt{n}) \times \mathcal{O}(\sqrt{n})$ grid as a minor. Towards this we prove that if there is a large grid then we can find a "clean part of the grid" (subgrid not containing vertices of T and bend vertices of either $\widehat{S}$ or S_{opt}) and reroute some of the paths in either $\widehat{S}$ or S_{opt}, that contradicts either the way $\widehat{S}$ is constructed or the optimality of S_{opt}.

▶ **Lemma 18.** *Given a set T of n points and a shortest path RST $\widehat{S}$ of T, there is an optimal rectilinear Steiner tree S_{opt} of T with the property that the treewidth of $\widehat{S} \cup S_{\mathsf{opt}}$ is bounded by $41\sqrt{n}$.*

Proof. Among the optimal Steiner trees of T with the minimum number of bend vertices, we select a tree S_{opt} which has maximum edge intersection with $E(\widehat{S})$. From Lemma 17, it follows that the number of bend vertices in S_{opt} is at most $3n$.

Let $S = \widehat{S} \cup S_{\mathsf{opt}}$. Let $\widehat{B}$ and B_{opt} be the set of bend vertices in $\widehat{S}$ and S_{opt} respectively. Let $U = T \cup \widehat{B} \cup B_{\mathsf{opt}}$ and $N = V(G) \setminus U$. Since $|T| = n$, $|\widehat{B}| \leq n$, and $|B_{\mathsf{opt}}| \leq 3n$, $|U| \leq 5n$. Let Π_S be a planar embedding of S, obtained by deleting all the edges and vertices not in S from the planar embedding Π of G. We show that the treewidth of S is at most $41\sqrt{n}$. We can assume that $n \geq 4$, as otherwise we can greedily find out the best rectilinear Steiner tree from the constant sized Hanan grid. For the sake of contradiction, assume that $\mathsf{tw}(S) > 41\sqrt{n}$. Then, by Proposition 10, there is a $9\sqrt{n} \times 9\sqrt{n}$ grid H appearing as a minor of S. Let $\mathcal{P}(H) = \{C_v | v \in V(H)\}$ be a minor model of H. Since H and G are connected graphs, $\mathcal{P}(H)$ is a partition of the vertex set $V(G)$. We identify a 3×3 subgrid H' of H by the following process. For any $v \in V(H)$, we mark the vertex v if $C_v \cap U \neq \emptyset$ (i.e, C_v contains a terminal or a bend vertex from $\widehat{S}$ or S_{opt}). Since $|U| \leq 5n$, the number of marked vertices in H is at most $5n$. Since H is a $9\sqrt{n} \times 9\sqrt{n}$ grid, there are at least $6n$ vertex disjoint 3×3 subgrids in H. This implies that there is a 3×3 subgrid H' in H such that each vertex of H' is unmarked. The fact that for $u \in V(H')$, $C_u \cap U = \emptyset$ implies the following observation.

▶ **Observation 19.** Let $u \in V(H')$ and $w \in C_u$.

- **(i)** $\mathsf{degree}_{\widehat{S}}(w), \mathsf{degree}_{S_{\mathsf{opt}}}(w) \in \{0, 2\}$. If for any $S_i \in \{\widehat{S}, S_{\mathsf{opt}}\}$, $\mathsf{degree}_{S_i}(w) = 2$, then the two edges in S_i incident with w are of same kind (either horizontal or vertical).
- **(ii)** If one horizontal (vertical) edge incident with w is present in S, then the other horizontal (vertical) edge incident with w is also present in S. Hence $\mathsf{degree}_S(w) \in \{2, 4\}$.

Note that H' is a connected graph and is a minor of a connected graph S. Let $\Pi_{H'}$ be a planar embedding derived from Π_S. By Lemma 9, we know that there is a simple cycle C' in S with the following properties.

- **(i)** $V(C') \subseteq \bigcup_{u \in V(H')} C_u$.
- **(ii)** For each vertex $w \in G$ that is contained in the internal region of C' in Π, there is a vertex $u \in H'$ with $w \in C_u$. In particular, all the vertices of $V(S) \setminus \bigcup_{v \in V(H')} C_v$ (which includes U) are not in the internal region of C'.

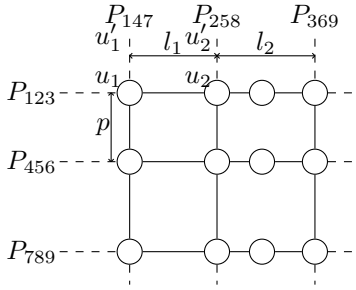

Figure 2 The subgrid G'.

(iii) For the internal vertex $v \in V(H')$, all the vertices in C_v are in the internal region of C'.

(iv) Finally, there is a vertex $w \in C_v$, in the internal region of C', such that $\mathsf{degree}_S(w) \geq 3$. By Observation 19, $\mathsf{degree}_S(w) = 4$.

That is, there is a cycle C' in the Hanan grid G such that $V(C') \subseteq V(S) \setminus U$, every point in the internal region of C' does not correspond to any vertex in U and there is a vertex w of degree 4 in S, which is in the internal region of C'. The following claim follows from Observation 19.

▶ **Claim 20.** *Let $u, v \in V(G)$ such that the points u and v are on the same horizontal line or on the same vertical line. If the line segment L connecting u and v does not intersect with the outer region of C', and there is an edge $v_1v_2 \in E(S)$ on the line L, then all the edges on the line segment L belong to $E(S)$.*

Let C' be a minimum-weight cycle satisfying properties $(i), (ii)$ and (iv) and w' be a vertex of degree 4 in the internal region of C'. Let $E_{w'}$ be the set of edges of S not in the outer region of C' and each edge either belongs to the horizontal line or vertical line containing w'.

▶ **Claim 21.** *Graph $G' = C' \cup E_{w'}$ is a subgrid of G. Moreover, $V(G') \subseteq V(G) \setminus U$, $E(G') \subseteq E(S)$, and all the subdivision vertices in G' have degree exactly 2 in S.*

The next claim provides us with the insight on how subpaths of $\widehat{S}$ and S_{opt} behave in G'.

▶ **Claim 22.** *Let F_h and F_v be the sets of horizontal and vertical edges in $G' = C' \cup E_{w'}$ respectively. Then exactly one of the following conditions is true.*

1. *$F_h \subseteq E(\widehat{S})$ and $F_v \subseteq E(S_{\mathsf{opt}})$.*
2. *$F_v \subseteq E(\widehat{S})$ and $F_h \subseteq E(S_{\mathsf{opt}})$.*

Note that G' is a 3×3 subgrid of G. Let G' be formed by horizontal paths $P_{123}, P_{456}, P_{789}$ and vertical paths $P_{147}, P_{258}, P_{369}$. Let $u_1, \ldots, u_9$ be the 9 vertices in G' such that the path P_{ijk}, where $i, j, k \in [9]$, contains the vertices u_i, u_j and u_k. Due to Claim 22, without loss of generality, we may assume that the horizontal paths belong to S_{opt} and the vertical paths belong to $\widehat{S}$. For a path P_{ijk}, we use P_{ij} and P_{jk} to denote the sub-paths of P_{ijk} connecting u_i and u_j, and u_j and u_k respectively. Let the length of the sub-path P_{12} be l_1 and the length of the sub-path P_{23} be l_2. By the definition of G', the length of P_{45} is also l_1, and the length of P_{56} is l_2. Let the length of P_{14} be p. The length of both P_{25} and P_{36} is p. (See Figure 2).

Suppose $l_1 + l_2 > 2p$. Then we consider the graph S^* formed by deleting in S_{opt} the path P_{123} and adding the two paths P_{14} and P_{36}. Since all the subdivision vertices of G' are of degree 2 in S, we have that S^* is a Steiner tree of weight strictly less than the weight of S_{opt}. This contradicts the choice of S_{opt}. Hence, this is not possible.

Suppose $l_1 + l_2 \leq 2p$. Without loss of generality, let $l_1 \leq l_2$. Thus, $l_1 \leq p$. Consider the two paths P_{147} and P_{258}. They are vertical paths of $\widehat{S}$, such that all the vertices in these paths belong to $V(G) \setminus U$ (that is, non-terminals and non-bend vertices) and have degree 2 in $\widehat{S}$ (by Observation 19). This implies that all the edges in any path $R \in \{P_{147}, P_{258}\}$ are added in a single step while constructing $\widehat{S}$. Since both the paths are parallel, by Observation 13, if a path R_1 in $\widehat{S}$ has both P_{147} and P_{258} as sub-paths, then R cannot be a shortest path between its endpoints. Thus, by construction of $\widehat{S}$, both P_{147} and P_{258} could not have been added to $\widehat{S}$ in a single step of the construction. Also by construction, one of them is added to $\widehat{S}$ before the other. Without loss of generality, let P_{147} be added before P_{258}. Again by construction, a path, containing P_{258} as a sub-path, was added in the i^{th} step, to connect a terminal t to the already constructed $\widehat{S}_{i-1}$. Let R^* be the path added to S_{i-1}. By definition of G', this terminal t must lie outside the region formed by the subgrid G'. Since P_{258} was part of a shortest path between $\widehat{S}_{i-1}$ and t, by Observation 13, t must lie on a row strictly higher than or strictly lower than the rows in G'. Suppose that this terminal lies above P_{123}. By Observation 19, both the vertical edges incident on u_1 belong to $\widehat{S}$. Since u_1 and u_2 are of degree 2 in $\widehat{S}$, both the vertical edges incident with u_1 as well as u_2 are added, when the path containing P_{147} or P_{258} is added to construct $\widehat{S}$. This implies that both the vertical edges incident with u_1 are present in S_{i-1}. Let u_1u_1' be the vertical edge present in S_{i-1} and not in $E(P_{147})$. Let u_2u_2' be the vertical edge not present in P_{258}. Now, consider the path R_2, between u_2' and t, obtained by concatenating the horizontal path between u_1' and u_2', and the sub-path of R^* connecting u_2' and t. The length of the path R_2 is strictly less than that of R^* and $u_1' \in S_{i-1}$. This is a contradiction. The case when t lies below any row in G' is identical to the other case. Thus, there is no such subgrid G' of G, such that G' is a subgraph of S. This implies that there is no $9\sqrt{n} \times 9\sqrt{n}$ grid as a minor in S. Due to Proposition 10, the treewidth of S must be at most $\frac{9}{2} \cdot 9\sqrt{n} = 41\sqrt{n}$. ◀

3.3 Dynamic Programming Algorithm for Rectilinear Steiner Tree

In this section we utilize all the results proved in the previous sections and design our algorithm for Rectilinear Steiner Tree. By Lemma 18, we known that given a shortest path RST $\widehat{S}$, there exists an optimum Steiner tree S_{opt} such that the treewidth of $S = \widehat{S} \cup S_{\mathsf{opt}}$ is bounded by $41\sqrt{n}$. The idea of the algorithm is to *implicitly* do a dynamic programming over a tree decomposition of S, *even though we do not know what S is*, to compute an optimum Steiner tree for T.

To give a proper intuition to our algorithm, we first recall the important step of the dynamic programming algorithm for Steiner tree over the tree decomposition of the graph S (see [3, Theorem 7.8] for more details). Let $(\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ be a nice tree decomposition of S where $\mathbb{T}$ is a rooted tree. For a node t, let V_t be the union of all the bags present in the subtree of $\mathbb{T}$ rooted at t. For a node t, we define a graph $S_t = (V_t, E_t = \{e \in E(S) \ : \ e \text{ is introduced in the subtree rooted at } t\})$. The important step in the algorithm for Steiner Tree is to compute the following information: for each bag X_t, $X \subseteq X_t$ and a partition $\mathcal{P} = (P_1, \ldots, P_q)$ of X, the value $c[t, X, \mathcal{P}]$ is the minimum weight of a subgraph F of S_t with the following properties:

1. F has exactly q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$ for all $i \in [q]$. That is, $\mathcal{P}$ corresponds to connected components of F.
2. $X_t \cap V(F) = X$. That is, the vertices of $X_t \setminus X$ are untouched by F.
3. $T \cap V_t \subseteq V(F)$. That is, all the terminal vertices in S_t belong to F.

For our purposes the second step is redundant but we still carry it out to give a proper analogy with the known algorithm for Steiner Tree over graph of bounded treewidth we

are referring to. Note that the number of blocks in the partition $\mathcal{P}$ is q. Throughout this section, q will denote the number of blocks of the partition in question.

It is known that computing the values $c[t, X, \mathcal{P}]$ for each tuple $(t, X, \mathcal{P})$, where $t \in V(\mathbb{T}), X \subseteq X_t$ and $\mathcal{P}$ is a partition of X, is enough to compute the value of an optimum Steiner tree of T in S and this can be computed in time $\mathsf{tw}^{\mathcal{O}(\mathsf{tw})}|V(S)|^{\mathcal{O}(1)}$ (See chapter 7 in the book [3]). In our case, we do not know the graph $S = \widehat{S} \cup S_{\mathsf{opt}}$ and a tree decomposition of S, but we know that the treewidth of S is at most $41\sqrt{n}$. This implies that the number of choices for bags in a tree decomposition of S is bounded by $n^{\mathcal{O}(\sqrt{n})}$. Consider the properties 1 and 2 mentioned above. They are *local* properties of the witness subgraph F with respect to the bag X_t. But the property 3 says that all the terminals in the subgraph S_t should be present in F. In fact we can bound the *potential* sets $T \cap V(S_t)$ using the rectilinear Steiner tree $\widehat{S}$. Observe that any bag X_t is a separator of size $\mathcal{O}(\sqrt{n})$ for S and thus for $\widehat{S}$, which implies that *for every connected component C of $\widehat{S} - X_t$, either $T \cap V(C)$ is fully in V_t or no vertex in $T \cap V(C)$ belongs to V_t.* Since each vertex in G has degree at most 4 and X_t is a bag in a tree decomposition, the number of connected components in $\widehat{S} - X_t$ is at most $4|X_t| \leq 164(\sqrt{n}+1)$. As observed before, for any connected component C of $\widehat{S} - X_t$, either $T \cap V(C)$ is fully in V_t or no vertex in $T \cap V(C)$ belongs to V_t. This implies that the potential sets $T' \subseteq T$ such that $T' = (V_t \setminus X_t) \cap T$ is bounded by $2^{164(\sqrt{n}+1)}$ and we can enumerate them in sub-exponential time. Using this observation we could keep track of property 3 as well, even though we do not know the graph S and its tree decomposition. Now, we move towards the formal explanation of our algorithm. Towards that we define a notion of *type* which is the analogue of a tuple $(t, X, \mathcal{P})$ in the dynamic programming we explained above.

▶ **Definition 23.** A type is a tuple $(Y, Y' \subseteq Y, \mathcal{P}, T')$ such that the following holds.

(i) Y is a subset of $V(G)$ of size at most $41\sqrt{n} + 2$.

(ii) $\mathcal{P}$ is a partition of Y'.

(iii) There exists a set of components $C_1, \ldots, C_q$ in $\widehat{S} - Y$ such that $T' = T \cap (Y' \cup \bigcup_{i=1}^{q} V(C_i))$.

Informally, in a type $(Y, Y', \mathcal{P}, T')$, Y represents a potential bag Y of a node (say t) in a tree decomposition of S. The set Y' and partition $\mathcal{P}$ have the same meaning as that of $(t, Y', \mathcal{P})$ in the normal dynamic programming algorithm for Steiner Tree. The set T' is the set of terminals in the graph S_t. The next lemma gives an upper bound on the number of types.

▶ **Lemma 24.** *There is a $2^{\mathcal{O}(\sqrt{n}\log n)}n^{\mathcal{O}(1)}$ time algorithm $\mathcal{B}$ enumerating all the types.*

Our algorithm is a dynamic programming algorithm over the types of S. As motivated earlier, this algorithm essentially describes the ideas of the dynamic programming algorithm for Steiner Tree over a tree decomposition of an input graph. Let $N = 3(|V(G)| + |E(G)|)$. Our algorithm computes values $\mathcal{A}[i, D]$, where $i \in [N]$ and D is a type. We want the table $\mathcal{A}[,]$ to contain all the information that is necessary for the correctness of the dynamic programming algorithm for Steiner Tree over a tree decomposition of S. To motivate the definition of $\mathcal{A}[,]$ we assume a *hypothetical tree decomposition* $\mathcal{T} = (\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ of S. For ease of understanding, let it be a nice tree decomposition and let the tree be rooted at a node $r \in \mathbb{T}$. The level of a vertex $t \in \mathbb{T}$ is the height of the subtree of $\mathbb{T}$ rooted at t. The *height* of a node t is the number of vertices in the longest downward path to a leaf from that node. Note that the level of any node of $\mathbb{T}$ is at most N. Suppose $t \in \mathbb{T}$ is a node at level i, and corresponds to the bag X_t. Let V_t be the union of bags present in the subtree rooted at t. Let the graph S_t be defined as $(V_t, \{e \mid e \text{ is introduced in a the subtree rooted at } t\})$. Let $T' = V_t \cap T$. Then, for any $X \subseteq X_t$, and a partition $\mathcal{P}$ of X having q blocks, $\mathcal{A}[i, (X_t, X, \mathcal{P}, T')] = c[t, X, \mathcal{P}]$. As mentioned before, $c[t, X, \mathcal{P}]$ is the minimum weight of the subgraph F of S_t such that the

following hold: (i) F has q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$, (ii) $X_t \cap V(F) = X$, and (iii) $T \cap V_t \subseteq V(F)$. For other pairs (i, D), we do not guarantee that the value of $\mathcal{A}[i, D]$ is meaningful. However, it is enough to maintain reasonable values for only the above subset of pairs (i, D). Of course we do not know S and thus we do not know the tree decomposition $\mathcal{T}$, so we store values in the table $\mathcal{A}[,]$ in such a way that given *any* nice tree decomposition of S, we have information pertaining to it. Thus, given a pair (i, D) where $D = (Y, Y' \subseteq Y, \mathcal{P}, T')$, we view Y as a bag of some hypothetical nice tree decomposition, $\mathcal{T}$, of S and assume that the level of the bag corresponding to Y in $\mathcal{T}$ is i. At a level i of this hypothetical nice tree decomposition, any bag is one of at most five kinds. We guess the relationship between a bag at level i and its children, which must be at level $i-1$. For example, if our hypothetical node t corresponds to an introduce vertex bag X_t, then we pretend that we know X_t, the child node t', the bag $X_{t'}$, and the vertex v that is being introduced at node t. Thereafter, for a subset $X \subseteq X_t$, and a partition $\mathcal{P}$ of X, we try to compute $\mathcal{A}[i, (X_t, X, \mathcal{P}, T')]$ using that values of $\mathcal{A}$ calculated at step $i-1$ of the algorithm. The calculation ensures that $\mathcal{A}[i, (X_t, X, \mathcal{P}, T')] = c[t, X, \mathcal{P}]$. In what follows we give a formal definition of $\mathcal{A}[,]$. We write a recurrence relation for $\mathcal{A}[i, D]$, where $i \in [N]$ and D is a type. We fix a terminal t^* in T

$$\mathcal{A}[1, D] = \begin{cases} 0 & \text{if } D = (\{t^*\}, \{t^*\}, \{\{t^*\}\}, \{t^*\}) \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

To define $\mathcal{A}[i, D]$ for $i \in [N] \setminus \{1\}$ and a type $D = (Y, Y', \mathcal{P}, T')$, we first define many intermediate values and take the minimum over all such values.

We first try to *view* Y as an introduce node in some nice tree decomposition of S and having level i. This viewpoint results in the following recurrence. For all $v \in Y$,

$$I_v[i, D] = \begin{cases} \infty & \text{if } v \notin Y' \text{ and } v \in T \\ \mathcal{A}[i-1, (Y \setminus \{v\}, Y', \mathcal{P}, T')] & \text{if } v \notin Y' \text{ and } v \notin T \\ \mathcal{A}[i-1, (Y \setminus \{v\}, Y' \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, T' \setminus \{v\})] & \text{if } v \in Y' \end{cases} \tag{2}$$

Intuitively, if Y is a bag corresponding to a node t in a tree decomposition of S and T' is the set of terminals in S_t, then Equation 2 corresponds to the computation of $c[t, Y', \mathcal{P}]$ in the dynamic programming algorithm of Steiner Tree. See [3, Theorem 7.8] for more detailed explanation.

For all $u, v \in Y$ and $uv \in E(G)$,

$$I_{uv}[i, D] = \min \left\{ \min_{\mathcal{P}'} \{\mathcal{A}[i-1, (Y, Y', \mathcal{P}', T')] + \mathsf{recdist}(uv)\}, \mathcal{A}[i-1, D] \right\}, \tag{3}$$

where $\mathcal{P}'$ varies over partitions of Y' such that u and v are in different blocks of $\mathcal{P}'$ and by merging these two blocks we get the partition $\mathcal{P}$. Note that if $\{u, v\} \not\subseteq Y'$ or u and v are in same block of $\mathcal{P}$, then Equation 3 gives $I_{uv}[i, D] = \mathcal{A}[i-1, D]$. Equation 3 corresponds to the computation of values in the introduce edge node where the edge uv is introduced.

For all $w \in V(G)$,

$$F_w[i, D] = \min\{\min_{\mathcal{P}'}\{\mathcal{A}[i-1, (Y \cup \{w\}, Y' \cup \{w\}, \mathcal{P}', T')]\}, \mathcal{A}[i-1, (Y \cup \{w\}, Y', \mathcal{P}, T')]\}, \tag{4}$$

where $\mathcal{P}'$ in the inner minimum varies over all the partitions obtained by adding w to one of the existing blocks. Equation 4 corresponds to computation in a forget node where w is forgotten.

$$J[i, D] = \min_{\substack{\mathcal{P} = \mathcal{P}_1 \sqcup \mathcal{P}_2 \\ T' = T'_1 \cup T'_2}} \{\mathcal{A}[i-1, (Y, Y', \mathcal{P}_1, T'_1)] + \mathcal{A}[i-1, (Y, Y', \mathcal{P}_2, T'_2)]\} \tag{5}$$

Equation 5 corresponds to a computation in a join node. We define $\mathcal{A}[i, D]$ for $i \in [N] \setminus \{1\}$ and type $D = (Y, Y', \mathcal{P}, T')$ as,

$$\mathcal{A}[i, D] = \min \begin{cases} \min_{v \in Y} \ I_v[i, D] \\ \min\limits_{\substack{uv \in E(G) \\ u,v \in Y}} I_{uv}[i, D] \\ \min_{w \in V(G)} F_w[i, D] \\ J[i, D] \end{cases} \tag{6}$$

For each $i \in [N]$ and each type D, we associate with $A[i, D]$ a subgraph of S. We say that a subgraph F is of type $(Y, Y', \mathcal{P}, T')$, where $\mathcal{P} = \{P_1, \dots P_q\}$ if the following holds.

(a) The number of connected components in F is equal to $|\mathcal{P}| = q$. We can order the connected components $C_1, \dots, C_q$ of F such that $V(C_i) \cap Y = P_i$.

(b) $V(F) \cap T = T'$.

In the following lemma, we show the connection between $A[i, D]$ and a graph of type D.

▶ **Lemma 25.** *Let $i \in [N]$ and D be a type. Furthermore, let $A[i, D]$ be computed by the Equation 6, and have a finite value ℓ. Then there is a subgraph F, of type D, such that* $\mathsf{recdist}(F) \leq \ell$.

The next lemma relates an optimal rectilinear Steiner tree to the values computed for the table $\mathcal{A}[,]$. Using induction on the level of nodes in $\mathcal{T}$ we prove the following.

▶ **Lemma 26.** *Let $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ be a nice tree decomposition of S. For a node t, let X_t be the corresponding bag, $X \subseteq X_t$, $\mathcal{P}$ be a partition of X, V_t be the union of bags in the subtree rooted at t, and $T' = T \cap V_t$. Then $A[i, (X_t, X, \mathcal{P}, T')] \leq c[t, X, \mathcal{P}]$.*

Finally, we describe the subexponential algorithm for Rectilinear Steiner Tree.

▶ **Theorem 27.** Rectilinear Steiner Tree *can be solved in time $2^{\mathcal{O}(\sqrt{n} \log n)} n^{\mathcal{O}(1)}$.*

Description of the Algorithm. We take as input a set T of n terminal points, the Hanan grid G of T and the weight function $\mathsf{recdist}$. Then using Lemma 15 we compute a shortest path RST $\widehat{S}$. By Lemma 18, we know that there is an optimal Steiner tree S_{opt} with $\mathsf{tw}(\widehat{S} \cup S_{\mathsf{opt}}) \leq 41\sqrt{n}$. Based on the shortest path RST $\widehat{S}$, we apply Lemma 24, to enumerate all possible types D of G. We fix an integer $N = 3(|V(G)| + |E(G)|)$ and a terminal t^* in T. For each $i \in [N]$ and each type D, the algorithm computes values $\mathcal{A}[i, D]$, according to Equations 1 and 6. The values in $\mathcal{A}[,]$ are filled in the increasing order of i. Finally, the algorithm outputs $\min_{i \in [N]} \mathcal{A}[i, (\{t^*\}, \{t^*\}, \{\{t^*\}\}, T)]$. The size of the table $\mathcal{A}[,]$ is $N \cdot 2^{\mathcal{O}(\sqrt{n} \log n)}$ and each entry can be filled in time $2^{\mathcal{O}(\sqrt{n} \log n)} n^{\mathcal{O}(1)}$. Thus, the running time of the algorithm is $2^{\mathcal{O}(\sqrt{n} \log n)} n^{\mathcal{O}(1)}$. Using standard back-tracking tricks we can also output an optimal RST. ◀

References

1. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74, New York, 2007. ACM.
2. Marcus Brazil and Martin Zachariasen. *Optimal Interconnection Trees in the Plane: Theory, Algorithms and Applications*. Springer, 2015.
3. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer-Verlag, Berlin, 2015.

4 Linda L. Deneen, Gary M. Shute, and Clark D. Thomborson. A probably fast, provably optimal algorithm for rectilinear Steiner trees. *Random Structures Algorithms*, 5(4):535–557, 1994.

5 Stuart E. Dreyfus and Robert A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.

6 Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007. doi:10.1007/s00224-007-1324-4.

7 Joseph L. Ganley. Computing optimal rectilinear Steiner trees: a survey and experimental evaluation. *Discrete Appl. Math.*, 90(1-3):161–171, 1999.

8 Joseph L. Ganley and James P. Cohoon. Improved computation of optimal rectilinear Steiner minimal trees. *Internat. J. Comput. Geom. Appl.*, 7(5):457–472, 1997. doi:10.1142/S0218195997000272.

9 M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

10 Qian-Ping Gu and Hisao Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica*, 64(3):416–453, 2012. doi:10.1007/s00453-012-9627-5.

11 M. Hanan. On Steiner's problem with rectilinear distance. *SIAM J. Appl. Math.*, 14:255–265, 1966.

12 Frank K Hwang. On Steiner minimal trees with rectilinear distance. *SIAM J. Appl. Math.*, 30(1):104–114, 1976.

13 Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner tree problem*, volume 53 of *Annals of Discrete Mathematics*. North-Holland Publishing Co., Amsterdam, 1992.

14 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for Subset TSP on planar graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1812–1830. SIAM, 2014.

15 L. Nastansky, S. M. Selkow, and N. F. Stewart. Cost-minimal trees in directed acyclic graphs. *Z. Operations Res. Ser. A-B*, 18:A59–A67, 1974.

16 Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013. doi:10.1007/s00453-012-9630-x.

17 Marcin Pilipczuk, Michał Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Subexponential-time parameterized algorithm for Steiner tree on planar graphs. In *Proc. of the 30th International Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 353–364. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

18 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. In *Proc. 55th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 276–285. IEEE, 2014.

19 Hans Jürgen Prömel and Angelika Steger. *The Steiner Tree Problem*. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Braunschweig, 2002.

20 Weiping Shi and Chen Su. The Rectilinear Steiner Arborescence Problem Is NP-Complete. *SIAM J. Comput.*, 35(3):729–740, 2005. doi:10.1137/S0097539704371353.

21 Clark D Thomborson, Linda L Deneen, and Gary M Shute. Computing a rectilinear steiner minimal tree in $n^{O(\sqrt{n})}$ time. In *Parallel Algorithms and Architectures*, pages 176–183. Springer, 1987.