Rectilinear Group Steiner Trees and Applications in VLSI Design*

Martin Zachariasen †   André Rohe ‡

November 7, 2000

Abstract

Given a set of disjoint groups of points in the plane, the rectilinear group Steiner tree problem is the problem of finding a shortest interconnection (under the rectilinear metric) which includes at least one point from each group. This is an important generalization of the well-known rectilinear Steiner tree problem which has direct applications in VLSI design, i.e., it is the fundamental problem that has to be solved in the detailed routing phase, since the logical units typically allow the nets to connect to several electrically equivalent ports. We present a first (tailored) exact algorithm for solving the rectilinear group Steiner tree problem (and related variants of the problem). The algorithm essentially constructs a subgraph of the corresponding Hanan grid on which existing algorithms for solving the Steiner tree problem in graphs are applied. The reductions of the Hanan grid are performed by applying point deletions and by generating full Steiner trees on the remaining points. Experimental results for real-world VLSI instances with up to 100 groups are presented.

1 Introduction

Interconnection problems in VLSI design are often stated as rectilinear Steiner tree problems in the plane: Given a set of points Z, construct a shortest tree using only horizontal and vertical line segments which interconnects Z. However, in real-world VLSI instances, there is usually not a one-to-one correspondence between points in the plane and the (logical) electrical terminals that should be connected. Instead, each terminal may consist of one or more line segments (or even rectangles) from which any point may be chosen as connection point.

In order to understand the nature of the problem, we will briefly describe the electrical background. In Figure 1 inverters and their structure on a chip

*Working Paper 00906, Institute for Discrete Mathematics, University of Bonn, Germany.
†Department of Computer Science, University of Copenhagen, Denmark; e-mail: martinz@diku.dk.
‡Institute for Discrete Mathematics, University of Bonn, Germany; e-mail: rohe@or.uni-bonn.de.

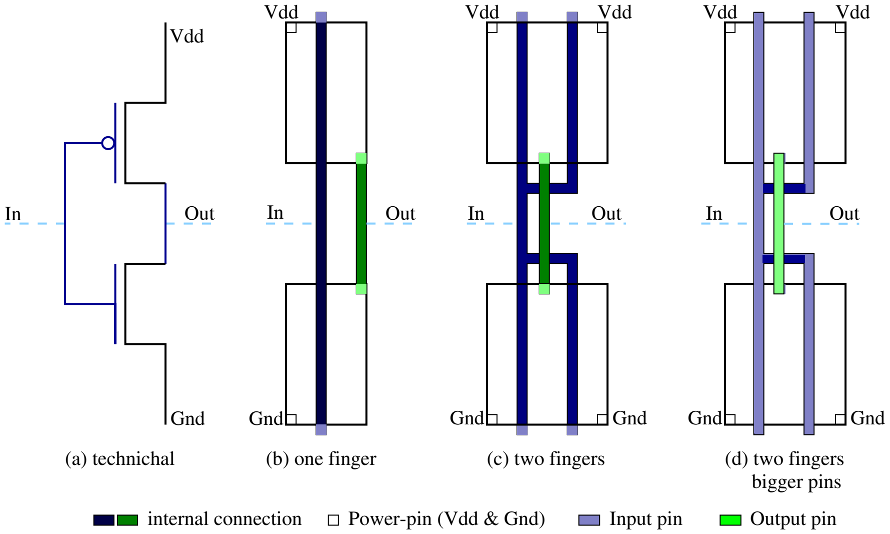


Figure 1: An inverter: transistor-level design and three different implementations.

are shown. Figure 1(a) shows the transistor-level design of an inverter; (b), (c) and (d) show three different implementations. They implement the same logic, but have different electrical behaviour. An inverter has input- and the output-*pins* which are regions in the plane where it is possible to connect a wire to the inverter. The pin-areas are shown as light-gray boxes (e.g., in (c) we have four input and two output-pins). In all examples each I/O pin consists of several areas. These areas are connected, but the connections cannot be used as a bridge for routing since they have a very high resistance; a signal through this connection would therefore take too long to propagate. Such a pin is called soft. Often output-pins do not have this property: They are connected just like a wire, so here it is possible to connect to any point on the pin and leave at any other point (hard pins).

On state-of-the-art IBM chips the pins are often structured as in Figure 1(d). Here possible pin-areas are sets of parallel lines (there are also three-finger inverters, etc.). Our algorithms will take advantage of this structure.

Of course, the problem we consider here is not the general VLSI routing problem which can be stated as follows: Given a 3-dimensional grid-graph, we would like to connect up to 10^6 sets of vertices (also called terminal nets), under the objective of minimizing total net length. The grid is up to $40,000 \times 40,000 \times 8$ nodes big. There are many constraints; the most important one is that no two nets are allowed to use the same edge or node. Algorithms to solve this general problem exist, but do not work for instances of this size [9, 12, 24]. Therefore, the nets are routed sequentially and the rectilinear Steiner tree problem with obstacles [5, 29] is considered (the routed nets form the obstacles).

Our motivation for studying Steiner trees connecting multiple pin-areas (but without obstacles) is that solving this problem for all nets provides a lower bound on the required wire-length of a chip. Note that the problems are reduced to two dimensions since the third dimension plays only a minor role. For more information about VLSI routing in general, the reader is referred to [18, 21, 22].

In this paper we show that the multiple pin-area VLSI routing problem reduces to the following *rectilinear group Steiner tree problem (RGSTP)*: Given a set of groups $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_k\}$, where each group Z_i, $i = 1, \ldots, k$, is a finite set of points in the plane, construct a shortest rectilinear tree which spans at least one point from each group. Such a tree is called a *rectilinear group Steiner minimum tree (RGSMT)*.

This problem is clearly NP-hard since it is a generalization of the classical rectilinear Steiner tree problem (in which all groups have size one) [6]. Even if Steiner points are not allowed, i.e., only direct connections between the given points can be used, the problem remains NP-hard [15]; note that the restriction of this problem to groups of size one is the polynomially solvable minimum spanning tree problem. Ihler [14] considered another restriction of RGSTP in which all the terminals are required to lie on two parallel lines and showed that this problem is also NP-hard.

The classical rectilinear Steiner tree problem admits a polynomial-time approximation scheme [1], which means that we can compute a solution within a factor of $1 + \epsilon$ from the optimum in polynomial time for any given $\epsilon > 0$. The group problem appears to be more difficult to approximate, since not even a constant-factor approximation algorithm is currently known for the problem. The general graph problem with arbitrary distances has been studied more intensively. We summarize the findings for the general problem below, but it should be pointed out that it may be possible to obtain better algorithms for the rectilinear problem, as well as for other geometric versions.

The group Steiner tree problem in graphs cannot be approximated better than a factor $\ln k$ in polynomial time, assuming P $\neq$ NP [4, 15]. Several (polynomial-time) approximation algorithms have been proposed for the problem [3, 7, 11, 15, 17, 23]. The best guarantee found which is independent of n is $O(k^\epsilon)$ for any fixed $\epsilon > 0$ [11]. A poly-logarithmic bound of $O(\log^2 n \log k \log\log n)$ was recently given by Charikar et al. [3]. This sophisticated algorithm uses a linear programming relaxation of the problem which is mapped into an integer solution by deterministic rounding techniques. If the number of vertices in each group is bounded by a constant $\rho \geq 1$, an approximation bound of 2ρ can be obtained [23]. This bound was also obtained via a linear programming relaxation. Since the groups in real-world VLSI instances in fact are small, this bound is perhaps of particular interest.

Practical heuristics were first considered by Reich and Widmayer [19]. These algorithms were experimentally evaluated in [15] and appeared to compute good solutions. Other practical heuristics showing good performance have been proposed [2, 11].

In this paper we give a first (tailored) exact algorithm for solving RGSTP. We show that the problem reduces to the ordinary Steiner tree problem in graphs; the graph is essentially the Hanan grid over all given points. In order to solve this problem efficiently we first show how the original set of points can be reduced without affecting the value of the optimal solution. By using a full Steiner tree generation approach we reduce the graph considerably, before standard branch-and-cut algorithms for the Steiner tree problem in graphs are applied. Computational results for a large set of real-world VLSI instances are presented.

The paper is organized as follows: In Section 2 we give all necessary definitions. Point reduction theorems are presented in Section 3. Our full Steiner tree generation approach is described in Section 4. Computational results — including a short description of existing branch-and-cut algorithms for the Steiner tree problem in graphs — are presented in Section 5. Finally, we give some concluding remarks in Section 6.

2 Definitions and Basic Observations

In this section we first give some basic definitions and observations related to the (ordinary) rectilinear Steiner tree problem in the plane; we use the same notation as in [20, 27]. Subsequently, we give some definitions for the group problem.

2.1 Rectilinear Steiner Trees

The rectilinear (or L_1) distance between two points $u = (x(u), y(u))$ and $v = (x(v), y(v))$ in $\mathbb{R}^2$ is $|x(u) - x(v)| + |y(u) - y(v)|$ and is denoted by $|uv|$. A rectilinear Steiner minimum tree (RSMT) interconnecting a set of terminals Z consists of vertical and horizontal *segments*. Segments intersect only at their endpoints, which are either terminals (belonging to Z), *corner points* (having degree two), *T-points* (having degree three) or *cross-points* (having degree four). A *Steiner point* is a non-terminal having degree three or more, that is, either a T-point or a cross-point.

The *Hanan grid* for Z is obtained by constructing vertical and horizontal lines through all terminals in Z. Since an RSMT always exists in this grid [10], it can be found by solving the Steiner tree problem in the corresponding Hanan grid graph. However, it is well-known that these graph problems are difficult to solve by using, e.g., branch-and-cut methods [16].

2.2 Rectilinear Full Steiner Trees

An RSMT is a union of *full Steiner trees (FSTs)*, in which every leaf is a terminal (having degree one) and all other nodes (having degree three or more) are Steiner points. Hwang [13] proved that there always exists an RSMT for which every FST has one of the two topologies shown in Figure 2: An FST spanning k terminals consists of a corner (also denoted the backbone) given by a *root* and a *tip*. The root is incident to the *long* leg and the tip incident to the *short* leg of the corner. FST type (i) has $k-2$ alternating segments incident to the long leg and no segment incident to the short leg, while type (ii) has $k-3$ alternating segments incident to the long leg and one segment incident to the short leg.

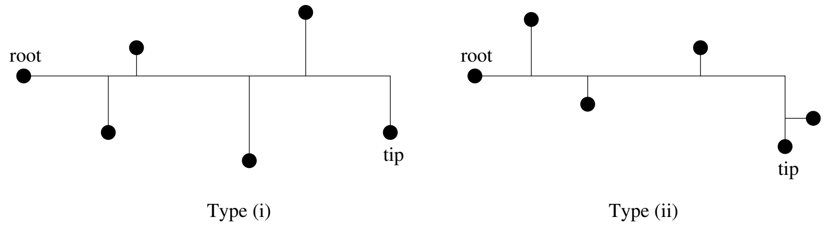


Figure 2: Hwang-topology full Steiner trees (FSTs).

Experience has shown that a sufficiently small superset of the FSTs for an RSMT can be generated very efficiently in practice [27]. This has formed the basis for the currently most successful exact algorithm for solving the rectilinear Steiner tree problem [25].

2.3 Rectilinear Group Steiner Trees

For the group problem, each terminal point is replaced by a set (or group) of points. The task is to construct a shortest rectilinear tree that interconnects at least one point from each group. Let the set of groups be $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_k\}$ and let $Z = Z_1 \cup Z_2 \cup \ldots \cup Z_k = \{z_1, z_2, \ldots, z_n\}$ denote the complete set of points (terminals). We assume that $Z_i \cap Z_j = \emptyset$ for all $i \neq j$. In order to identify terminals with groups we define $gr(z_i) = l$ if $z_i \in Z_l$.

A terminal set $V \subseteq Z$ is *valid* if it contains exactly one member from each group, i.e., if $|Z_l \cap V| = 1$ for all $l = 1, \ldots, k$. Let $\mathcal{V} = \{V \subseteq Z \ : \ |Z_l \cap V| = 1,\ l = 1, \ldots, k\}$ be the set of all possible valid terminal sets.

One possible approach for solving the group problem is to construct an RSMT for each valid set $V \in \mathcal{V}$, denoted by $RSMT(V)$, and select the shortest such tree. A consequence of this observation is that a rectilinear group Steiner minimum tree (RGSMT) exists in the Hanan grid for Z; in Section 3 (Theorem 1) we give a stronger version of this result by exploiting the typical structure of VLSI instances. Another consequence is that there exists an RGSMT which is a union of FSTs with respect to V having Hwang-topology. Each of these FSTs spans terminals from Z, but the union of all FSTs does not have to interconnect all terminals from Z. In Section 4 we show how a fairly small set of FSTs containing at least one RGSMT can be generated.

Consider $RSMT(V)$ for a valid set V. For a given group Z_l let $z_i \in Z_l \cap V$ be the terminal chosen from the group. It may be necessary for $RSMT(V)$ to *intersect*, in geometric terms, some other terminal $z_j \in Z_l \setminus \{z_i\}$. Avoiding such intersections, that is, solving the problem in which exactly one terminal from each group is intersected, requires a definition of how to make detours around terminals and therefore has limited practical merit.

3 Point Reductions

In the VLSI routing problem each pin i, $i = 1, \ldots, k$, consists of a (finite) set of axis-aligned closed rectangles $R_{i1}, R_{i2}, \ldots, R_{i\lambda_i}$, where λ_i denotes the number of rectangles for pin i. The rectangles may be line segments or single points, and we assume that every pair of rectangles is *disjoint* (Figure 3a). In real VLSI-instances the rectangle corners are given by integer coordinates; furthermore, the tree that is sought is restricted to be on the integer grid, i.e., all intersections and Steiner points should have integer coordinates. As will be shown below, this will automatically be the case if we perform the proposed reductions.

The shortest tree T^* that we seek should, for every pin, intersect at least one of its rectangles. Assign any of these intersections to be the *connection point* of the pin (or group) in T^*. In the following we will give some conditions that can be used to eliminate possible connection points from consideration.

Assume that an RGSMT intersects a rectangle R_{ij}. Since the rectangles have disjoint interiors, the tree has to intersect the boundary of R_{ij}. Therefore, it suffices to consider the boundary of each rectangle as candidate set for the connection point of the group — or alternatively, the four line segments that make up the boundary of each rectangle (Figure 3b).

For each pin i we therefore have a set of vertical or horizontal segments $s_{i1}, s_{i2}, \ldots, s_{i\gamma_i}$, where γ_i is the number of segments for pin i. In Theorem 1 we show that it suffices to consider a finite set of points for each pin (or group). This set of points is a subset of the points that would be obtained by constructing the Hanan grid of the endpoints of all segments and finding the intersections of this grid with the segments (Figure 3c). In Section 3.2 we give further conditions for removing points from considerations (Figure 3d).

3.1 From Segments to Points

In this section we show that only a finite set of points on the given segments needs to be considered. The theorem is an extension of the Hanan grid theorem for ordinary rectilinear Steiner trees.

Definition 1 *Consider an interior point p in a vertical segment s. Depending on the output of the following algorithm, the point p is said to be* vertically free.

Let h_l (resp. h_r) be the halfline starting in p and pointing to the left (resp. right) of s (Figure 4). Each of these halflines intersects the other groups in a set of points; if the halflines overlap an horizontal segment only the endpoints of the segment need to be considered. Denote by L_l and L_r the list of points intersected by h_l (resp. h_r) in increasing distance from p, and not including p itself. Note that one or both of these lists may be empty.

Now we start traversing L_l. There are three cases to consider, depending on the current point $q \in L_l$:

- *$gr(q) = gr(p)$:* stop
- *q is an interior point in a vertical segment and $gr(q) \neq gr(p)$:* continue

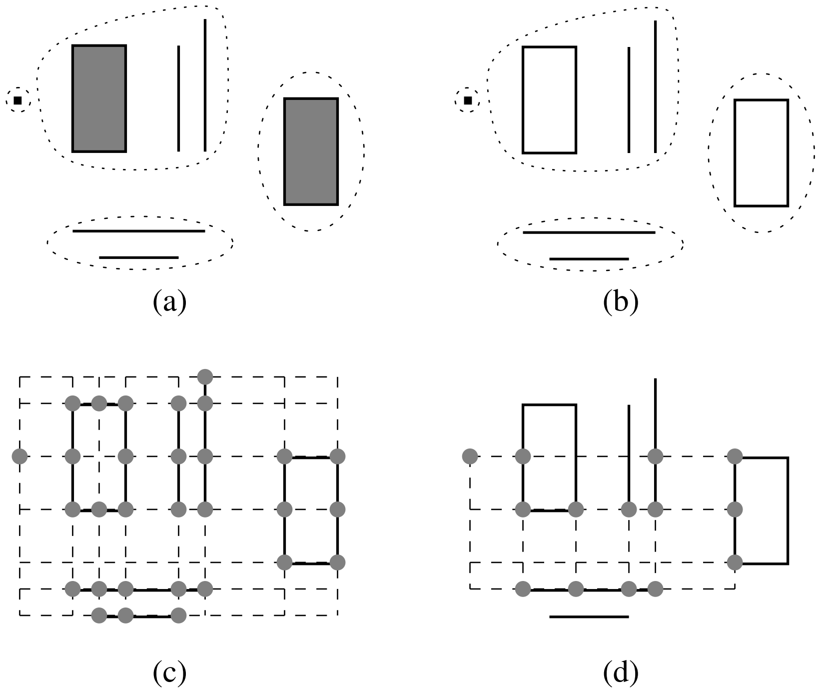


Figure 3: From rectangles to segments to points. (a) Original pin rectangles; (b) rectangle boundaries; (c) Hanan grid reduction; (d) improved reduction.

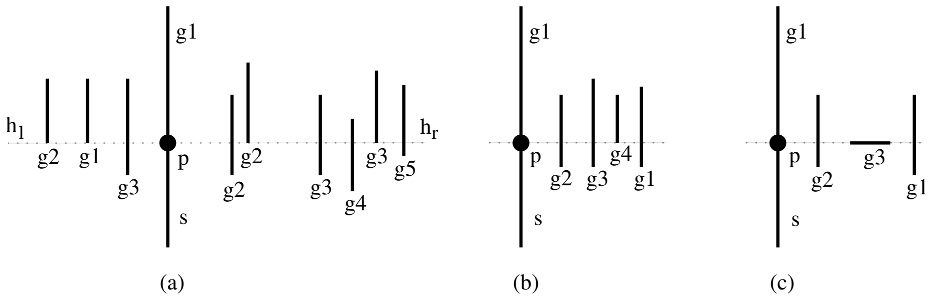


Figure 4: Vertically free point definition; for each segment we use gi to indicate that the segment belongs to group i. (a) point p is vertically free; (b) point p is *not* vertically free; (c) point p is *not* vertically free.

- *q is an endpoint in a segment and $gr(q) \neq gr(p)$: if $gr(q)$ has not been intersected before in this traversal then output "p is* not *vertically free", otherwise* continue

If this traversal ends without concluding that p is not vertically free and the same happens when traversing L_r, then the point p is vertically free.

Informally, an interior point p of a vertical segment is vertically free if, when looking left or right, the first point that is met in every group is an interior point of a vertical segment. Furthermore, if a point that belongs to the same group as p is encountered, no points beyond that need to be considered. An *horizontally free point* is defined analogously.

Construct the Hanan grid of the endpoints of all segments. Since there exists an RGSMT in this grid [28], we only need to consider the finite set of intersection points of the Hanan grid as possible connection points, more precisely the intersection points that overlap with the given segments. Thus we have reduced the original VLSI problem to RGSTP, where each group consists of a finite set of terminals. The union of all terminals is as usual be denoted Z.

For the obtained RGSTP, let $\mathcal{C} \subseteq \mathcal{V}$ be the subset of valid sets such that for all $C \in \mathcal{C}$, $RSMT(C)$ is an RGSMT. Let $Z_f \subset Z$ be the set of vertically free points. In the following lemma we prove that there exists a $C \in \mathcal{C}$ such that $C \cap Z_f = \emptyset$.

Lemma 1 *There exists an optimal solution to RGSTP which uses* no *vertically free points as connection points.*

Proof. Assume this is not true, i.e., all optimal solutions use at least one vertically free point as connection point. Let T^* be an RGSMT with the following properties: The y-coordinate of the bottommost vertically free connection point is as large as possible (recall that we only consider RGSMTs on the Hanan grid of the endpoints of the given segments). If there is more than one RGSMT fulfilling this criterion, we choose one that has the *fewest* number of bottommost vertically free connection points. We will show that we can remove at least one of these bottommost vertically free connection points, contradicting the fact that there actually exists such an optimal solution, that is, showing that there must exist an optimal solution which uses no vertically free connection points.

Pick one of the bottommost vertically free connection points (denoted by p). The point p cannot be incident to an up-going segment in T^*, since this would allow us to move p up to the next horizontal Hanan grid line without destroying the validity of the solution (recall that p is an interior point on its vertical segment). By moving p up we would construct an optimal solution having fewer bottommost vertically free connection points, contradicting our assumption.

Furthermore, p cannot *only* be incident to a down-going segment, since this would imply that T^* was not optimal. Therefore, there exists an horizontal segment of T^* which intersects p. Let s_p be this segment extended maximally (Figure 5). The segment s_p cannot intersect a non-vertically free point q for which $gr(q) = gr(p)$ since this would imply that we could move the connection point of this group from p to q, thus again reducing the number of bottommost vertically free connection points (Figure 5a). Therefore, we may assume that all points P intersected by s_p and belonging to the same group as p are *vertically free*.

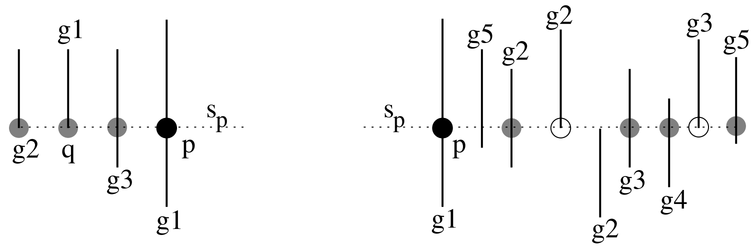


Figure 5: Proof illustration, Lemma 1. (a) s_p intersects a non-vertically free point of the same group as p; (b) all connection points become interior points in their respective vertical segments (white node: old connection point, gray node: new connection point).

Consider all other groups intersected by s_p. For each group we now know (since all points in P are vertically free), that there exists at least one *interior* point in a vertical segment that is intersected by s_p. For each group, make any of these interior points the new connection point of the group. Consequently, *all* connection points on s_p become interior points of vertical segments (Figure 5b).

Since all connection points on s_p are interior we can slide s_p and its connection points upwards (or downwards) without changing the length of the solution. By sliding s_p up to the next horizontal Hanan grid line, we have removed at least one of the original bottommost vertically free connection points, contradicting our assumption that T^* was an optimal solution having the fewest number of bottommost vertically free connection points; this finishes the proof. □

Theorem 1 *There exists an optimal solution to RGSTP which uses* no *vertically* or *horizontally free points as connection points.*

Proof. First we observe that a point can only be vertically or horizontally free — and not both (because all segments are either vertical or horizontal). Furthermore, Lemma 1 obviously also holds for horizontally free points. Thus, consider the set of optimal solutions which use no *horizontally* free points. Among these solutions we now again choose one for which the bottommost *vertically* free connection point has the largest possible y-coordinate and which — secondly — has the fewest number of bottommost vertically free connection points. We may again use the arguments of Lemma 1 to remove at least one bottommost vertically free connection point, thus contradicting our assumption.

However, there is one minor complication. We need to make sure that we do not create any *horizontally* free connection points in this process. When arguing that all intersections of s_p with points belonging to the same group as p are vertically free, we used the fact that if any non-vertically free point existed we could simply make this the new connection point for the group. However, this swap could create a new horizontally free connection point on some horizontal segment s'. But since s_p must intersect at least one of the endpoints of s', we can avoid this problem by using the intersected endpoint as the new connection point. $\square$

3.2 Additional Point Reductions

In this section we give some additional criteria for removing terminals while retaining at least one RGSMT. In contrast to the method given in the previous section, these tests are so-called sequential tests; this means that only one terminal can be deleted at a time, since deleting a terminal might make other terminals necessary. Therefore, the final result depends on the order in which the terminals are deleted.

We only give reduction criteria for specific orientations. Obviously they are also valid for arbitrary orientations.

Definition 2

$$box(x_1, y_1, x_2, y_2) := \{(x, y) \ : \ x_1 < x < x_2 \ \wedge \ y_1 < y < y_2\}$$

Lemma 2 *Let p, p_d and p_r be three distinct terminals of the same group with $x(p) = x(p_d)$, $y(p) > y(p_d)$, $x(p) < x(p_r)$ and $y(p) = y(p_r)$. If*

$$R = box(-\infty, y(p_d), (x(p) + x(p_r))/2, \infty)$$

does not include any terminal t with $gr(t) \neq gr(p)$, then there exists an optimal solution to RGSTP which does not use p as a connection point.

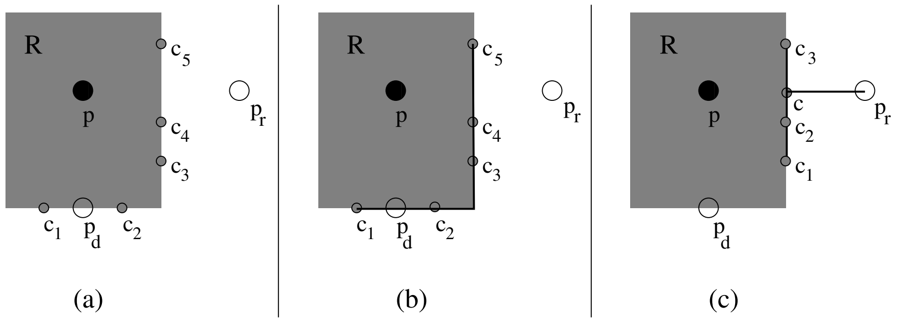


Figure 6: Lemma 2, proof illustration.

Proof. Assume this is not true, so let T^* be an RGSMT which uses p as a connection point but does not intersect p_d and p_r (if T^* intersects p_d or p_r, we just use this point as a connection point). Let $c_1, \ldots, c_k$ be the points on the border of R that have a connection to p just inside R, sorted by x- and then by y-coordinate (see Figure 6a).

Now we will transform T^* to an RGSMT which is not longer than T^* and intersects p_d or p_r. If such a solution exists, we have an optimal solution that does not use p as a connection point.

If $y(c_1) = y(p_d)$, we delete all connections inside R and add the edges (c_i, c_{i+1}) $\forall i \in 1, \ldots, k-1$ and (c_1, p_d) if $x(c_1) > x(p_d)$. The new tree intersects p_d and is not longer than T^* (Figure 6b).

If $y(c_1) > y(p_d)$, we delete all connections inside R and add the edges (c_i, c_{i+1}) $\forall i \in 1, \ldots, k-1$ and (c, p_r), with $x(c) = (x(p) + x(p_r))/2$ and $y(c) = y(p_r)$. If $y(c_k) < y(c)$, we also need the segment (c_k, c) to reconnect the tree; similarly, if $y(c_1) > y(c)$, we need to add the segment (c, c_1) to reconnect the tree. The new tree intersects p_r and is not longer than T^* (Figure 6c). $\square$

Note that an extension of this lemma is obtained when p_r does not exist and $\mathrm{box}(-\infty, y(p_d), \infty, \infty)$ does not include any terminal t with $gr(t) \neq gr(p)$.

Lemma 3 *Let p be a terminal, p_d and p_r as in Lemma 2 and p_u a terminal above p with $x(p) = x(p_u)$ and $gr(p) = gr(p_u)$. If*

$$R = box(-\infty, y(p_d), x(p_r), y(p_u))$$

does not include any terminal t with $gr(t) \neq gr(p)$, then there exists an optimal solution to RGSTP which does not use p as a connection point.

Proof. We first assume that p is a leaf in an RGSMT that is a union of Hwang-topology FSTs and distinguish between three cases:

(i) p has a connection to the right. Let u be the endpoint of this segment. If this segment intersects p_r we are done (make p_r the new connection point), so assume that $u \in R$. Now, since u is part of a Hwang topology FST it must be vertically connected (possibly through some other nodes) to some terminal z. If $z \in R$ then $gr(z) = gr(p)$ and we are done. Otherwise, $z \notin R$, which means that we can slide the segment pu either up or down until either p_u or p_d is hit, making either p_u or p_d the new connection point.

(ii) p has a connection to the left. The (partial) FSTs that might occur are shown in Figure 7. Arrows in the Figure show connections that leave R and go directly to a terminal. Subcase (a) can be dealt with by moving the horizontal segment up or down. Subcase (b), an even number of connections to the horizontal line, can be reduced to subcase (a) by a flip and shift combination (subcase (b')). Subcase (c), an odd number of connections to the horizontal line, is also changed this way (subcase (c')); here we intersect p_d or p_u and are done.

(iii) p has a connection up or down. Let u be the endpoint of this segment. If u has an horizontal connection both to the left and to the right, this means that p and u are part of an FST that can be transformed into a tree that intersects either p_u or p_d. This transformation is similar to the one given in case (ii). Otherwise, u has a single horizontal connection to the right, since a connection to the left has to go to some terminal in the same group as p. This means that u is a corner point; by flipping this corner point we reduce this case to case (i).

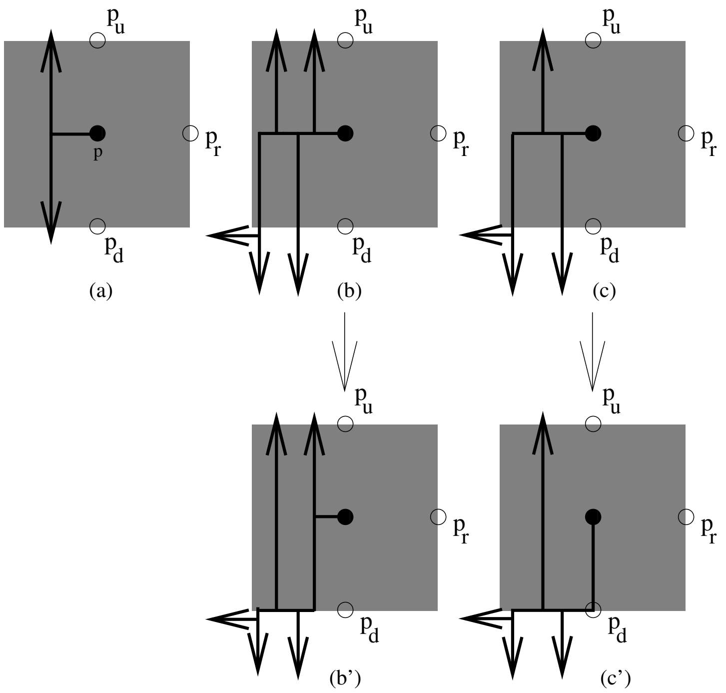


Figure 7: Lemma 3, connection to the left.

Up to this point we showed that if p is a leaf we can always include either p_u, p_d or p_r in the tree. If p is not a leaf, the case study above shows that the only non-leaf case that occurs has exactly one connection to the left and one to the right. If one of the two corresponding FSTs can be made to intersect some other terminal in the same group as p while still intersecting p, we are done (replace connection point p by the other terminal). Assume that the right FST is transformed into another FST which intersects p_u (p_d) but no longer p. Then it is always possible to transform the left FST into a tree that intersects p_u (p_d) *also*, as shown in the above construction; we can slide the connection from p both up and down as required. □

4 FST Generation

As pointed out in Section 2.3, an RGSMT is a union of full Steiner trees (FSTs) spanning subsets of terminals from Z, the complete set of terminals in all groups. A well-known difficulty when solving the (ordinary) rectilinear Steiner tree problem is the large number of optimal solutions. Even if the edges of optimal solutions are restricted to lie on the Hanan grid — as they will do when Hwang-topology FSTs are used — there are typically many ways in which a path may travel without changing its length. It is therefore important to have a precise characterization of the particular optimal solutions that one is looking for. In Section 4.1 we give such a characterization using an FST-based description.

The generation of a set of FSTs $\mathcal{F}$ containing at least one RGSMT according to this description is given in Section 4.4. This generation algorithm is similar to the one given by Zachariasen [27] for the rectilinear Steiner tree problem. In Sections 4.2 and 4.3 we present the two most important properties of RGSMTs that are used to limit the number of surviving FSTs.

4.1 An FST-based Characterization of RGSMTs

An RGSMT is an RSMT for some valid set V^* (see Section 2.3). Such an RGSMT may intersect terminals not from V^*, and we will now assume that such terminals actually are part of the RGSMT in the sense that segments from the RGSMT not just pass over these terminals but are required to split at such terminals. It is advantageous only to consider RGSMTs for which the number of terminals spanned is *maximized*. Secondly, we may assume that these RGSMTs are unions of FSTs interconnecting the terminals spanned. Finally, we may assume the the number of FSTs is maximized, e.g., it should not be possible to split an FST into two FSTs having the same total length.

We illustrate this characterization in Figure 8. The RGSMT in Figure 8a does not intersect a maximum number of terminals, while the RGSMT in Figure 8b does. However, the latter RGSMT does not have a maximum number of FSTs. An RGSMT fulfilling our characterization is shown in Figure 8c.

The advantage of this characterization is that the FSTs we seek cannot be transformed into trees intersecting terminals not already spanned (without increasing their length). Consider an FST F in such an RGSMT. Since F cannot be made to intersect any terminal not already spanned, its *inner rectangle* (for a rough definition see Figure 9), including its border, must be completely free of terminals, except for the terminals that define it [27]. Note that this property is solely based on our specific characterization of RGSMTs. In Section 4.2 we give other empty-region properties which must hold for FSTs in *any* RGSMT.

4.2 Empty Regions

Let uv be a segment in an RGSMT. The so-called *lune* $\mathcal{L}(u,v) = \{p \in \mathbb{R}^2 \, : \, |pu| < |uv| \wedge |pv| < |uv|\}$ is the set of points closer to both u and v than u and v are to each other. For the ordinary rectilinear Steiner tree problem it is easy to see

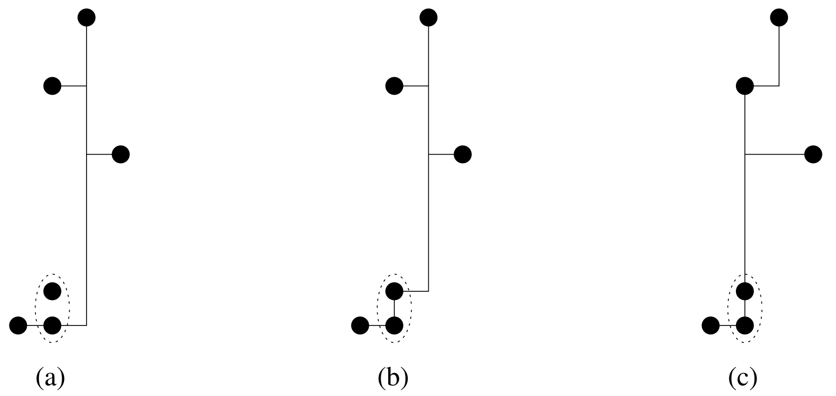


Figure 8: Transformation of an RGSMT into an alternative RGSMT fulfilling our characterization.

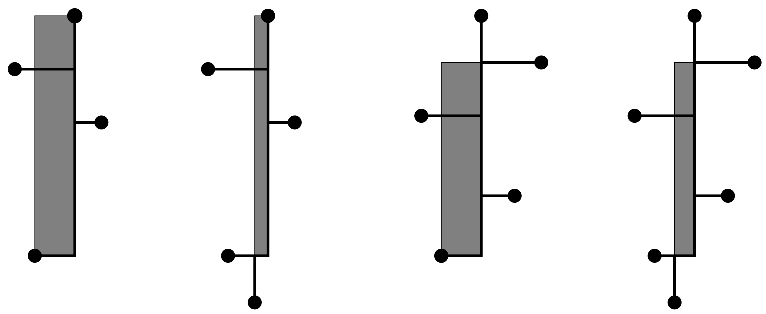
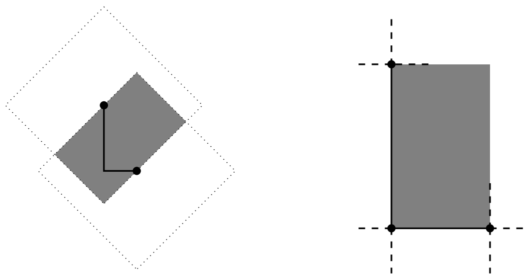


Figure 9: Empty inner rectangle; the size of the empty region depends on the FST-type and the parity of the number of terminals attached to the long leg.

that $\mathcal{L}(u, v)$ must be completely free of terminals [25, 27] since otherwise the tree containing uv can be shortened.

The generalization to the group problem is straightforward. Instead of requiring that the lune should be completely empty, we must have that not *all* terminals of some group Z_l are in $\mathcal{L}(u, v)$. Assume the opposite: Since at least one terminal $z_i \in Z_l$ is spanned by the tree we may remove the segment uv and add the segment uz_i or the segment vz_i, depending on whether z_i belongs to the component of v or u when uv is removed. The resulting tree is shorter and still feasible — a contradiction.

This means that all known empty regions for the ordinary rectilinear problem can be reused for the group problem using the appropriate interpretation of emptiness. In Figure 10 we present the two further types of empty regions used in this paper. For a proof of their validness, see e.g. [27].



(a) (b)

Figure 10: Empty regions. (a) Empty skew diamond; (b) Empty corner rectangle.

4.3 Group Bottleneck Steiner Distances

In this section we describe a method for providing upper bounds on the length of edges in an RGSMT. We essentially generalize the notion of *bottleneck Steiner distances* for ordinary Steiner tree problems to the group problem.

Let e be any edge on the path $P(z_i, z_j)$ connecting two terminals z_i and z_j in an RGSMT. We would like to compute an upper bound $BSD(z_i, z_j)$ on $|e|$. For the ordinary problem a good bound is obtained by computing the MST over all terminals and let $BSD(z_i, z_j)$ be the length of the longest edge on the path in the MST between the two terminals in question; note that by using this definition we get the natural relation $BSD(z_i, z_j) \leq |z_i z_j|$.

The problem when working with groups is that we have no preliminary knowledge about *which* terminals are going to be in an optimal solution (of course, except for groups with only one terminal). Therefore, we need to take this lack of knowledge into account when computing a similar bound for the group problem.

Let $BSD_V(z_i, z_j)$ be the bottleneck Steiner distance between z_i and z_j given that we span the valid set V (and $z_i, z_j \in V$). Now we define

$$BSD(z_i, z_j) = \max_{V \in \mathcal{V}, z_i, z_j \in V} BSD_V(z_i, z_j)$$

Obviously this gives a valid bound, since we take the maximum over all valid sets that contain z_i and z_j. Unfortunately, it is in general NP-hard to compute this bound (see the proof in the Appendix).

Therefore, we give two straightforward polynomial-time algorithms that compute an approximation (i.e., an upper bound) to $BSD(z_i, z_j)$ for all pairs of terminals z_i and z_j. Since $|z_i z_j|$ is a trivial valid upper bound we will only consider improvements over this bound.

In the following we let $|z_i Z_l|$ denote the *maximum* distance from z_i to some terminal in the group Z_l while $|Z_l Z_m|$ denotes the maximum distance between a terminal in Z_l and a terminal in Z_m.

MST-based algorithm

Construct an edge-weighted undirected graph G_{MST} with $n + k$ vertices, one for each terminal $z_i \in Z$ and one for each group $Z_l \in \mathcal{Z}$. Each terminal z_i is connected to exactly one group, namely one of the groups Z_l for which $|z_i Z_l|$ is minimized and $z_i \notin Z_l$ (one of the groups that is "closest" to z_i). All pairs of groups are connected by an edge. The length of an edge (z_i, Z_l) is $|z_i Z_l|$ and the length of an edge (Z_l, Z_m) is $|Z_l Z_m|$. Clearly, this graph can be constructed in $O(n^2)$ time.

Now we compute an MST in G_{MST}; the single edge adjacent to each of the terminals will always be part of an MST. An approximation to $BSD(z_i, z_j)$, denoted by $BSD_{MST}(z_i, z_j)$, is the length of the longest edge on the path $P_{MST}(z_i, z_j)$ in the MST between z_i and z_j.

The validity of the bound follows from the fact that for any valid set $V \in \mathcal{V}$ such that $z_i, z_j \in V$, we can obtain a path $P_V(z_i, z_j)$ which corresponds to $P_{MST}(z_i, z_j)$ but uses one of the terminals in V for each group visited by $P_{MST}(z_i, z_j)$. The longest edge on this path will never be greater than $BSD_{MST}(z_i, z_j)$.

The graph G_{MST} has $O(n)$ vertices and $O(k^2 + n)$ edges. An MST can therefore be computed in $O(n \log n + k^2)$ time. Furthermore, the longest edges can be identified by performing a breadth-first-search from each terminal node giving a total running time of $O(n^2)$.

Floyd-Warshall-based algorithm

The weakness of the previous approach is that the terminals are always connected to the "closest" group. An alternative is to use a variant of the Floyd-Warshall all-pairs shortest path algorithm.

Assume the terminals are indexed from 1 to n and the groups from $n + 1$ to $n + k$. We define a distance matrix $d_{\mu\nu}$ for every pair of terminals/groups as follows:

$$d_{\mu\nu} := \begin{cases} |z_\mu z_\nu| & 1 \le \mu \le n,\ 1 \le \nu \le n \\ |z_\mu Z_\nu| & 1 \le \mu \le n,\ n+1 \le \nu \le n+k \\ |z_\nu Z_\mu| & n+1 \le \mu \le n+k,\ 1 \le \nu \le n \\ |Z_\mu Z_\nu| & n+1 \le \mu \le n+k,\ n+1 \le \nu \le n+k \end{cases}$$

Now, for each group Z_l we iteratively try to update the distance matrix by testing whether it is advantageous to travel via group Z_l:

$$d_{\mu\nu} = \min(d_{\mu\nu}, \max(d_{\mu l}, d_{l\nu}))$$

The resulting distance $BSD_{FW}(z_i, z_j) = d_{ij}$ for a pair of terminals z_j and z_j is clearly a valid upper bound for $BSD(z_i, z_j)$. Also, it is easy to see that this bound dominates the MST-bound — that is, $BSD_{FW}(z_i, z_j) \le BSD_{MST}(z_i, z_j)$ — since the MST path will be among the paths that are tried by the Floyd-Warshall algorithm. However, the $O(kn^2)$ running time of the Floyd-Warshall algorithm is greater than the $O(n^2)$ running time of the MST-based algorithm.

Comparison of BSD_{MST} and BSD_{FW} bounds

Both approximations are very good in practice, but since the bound BSD_{FW} is provably better and has a reasonable running time we chose to use this bound in our FST-generation algorithm. Nevertheless, both bounds are non-optimal as shown in Figure 11. The reason is that both algorithms use the maximum distance between groups and this distance is not necessarily the BSD. Since the groups in VLSI instances typically have a small (geometric) diameter, the maximum distance will usually be a good approximation to the distance between terminals in different groups.

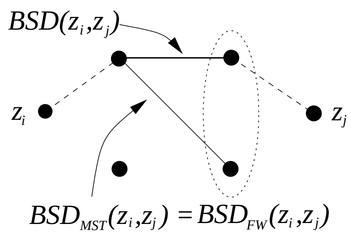


Figure 11: Five-group counter-example showing that BSD_{MST} and BSD_{FW} are not optimal.

4.4 FST Generation Algorithm

In this section we adapt the FST generation algorithm from [27] to the group problem. The first phase of the algorithm is the preprocessing in which data structures are set up that help pruning non-optimal FSTs. For every pair of terminals $z_i, z_j \in Z$ we compute $BSD_{FW}(z_i, z_j)$ and determine whether the rectangle given by z_i and z_j is group-empty.

The FST generation algorithm grows FSTs from a given root assuming that the long leg of the FST has a given direction (north, west, south and east). In the following we present the main idea of the algorithm for given root and direction.

Consider the root and east-bound direction shown in Figure 12a. First all terminals that can be attached to the long leg are identified; using group-empty regions and BSD properties many terminals can be ignored as candidates. Using this limited set of terminals, a backtrack search algorithm is used to generate all FSTs having the given root and direction. The fundamental property that is used is that the incident segments alternate along the long leg.

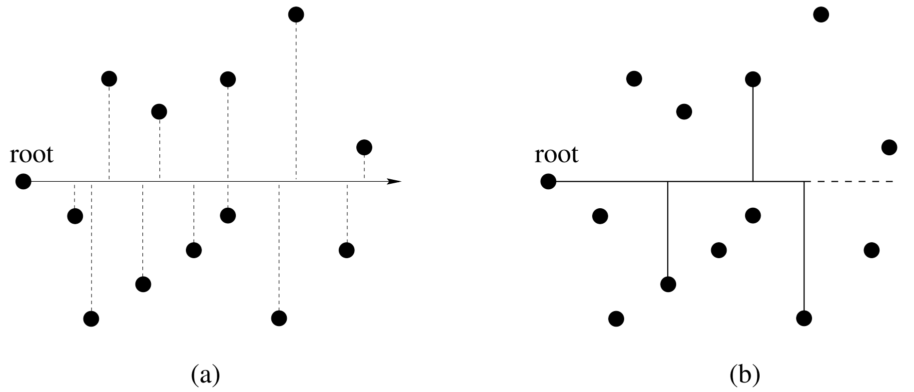


Figure 12: Growing FSTs. (a) Terminal candidates to be attached to the long leg; (b) Partial FST.

For a given partial FST (Figure 12b) a number of tests are applied. Firstly, the partial FST is tested as a potential type (i) FST candidate. Then all type (ii) FSTs having the last incident segment as short leg are tried. Finally, it is tested whether the partial FST can be part of a larger FST; if not, the search backtracks — otherwise the tree is extended by adding another incident segment to the long leg. All these tests are based on BSD-properties, group-empty regions and other simple optimality criteria for FSTs [27].

For the RSMT problem this algorithm is surprisingly fast despite the fact that no polynomial bound on its running time exists. For the group problem the tests are weaker. This results in a larger number of surviving FSTs as shown in Section 5. For some problem instances — and particular roots and directions — combinatorial explosion makes the approach impractical. In these few cases, we cancel FST generation and generate a so-called local Hanan grid instead. This is a grid graph that contains all possible FSTs for the given root and direction as subgraphs. The cut-off was chosen such that the number of local Hanan grids was as small as possible, since these local grids tended to make the solution of the graph problem harder.

5 Computational Results

The new exact algorithm was experimentally evaluated on two types of instances:

State-of-the-art IBM chip instances. We took two current IBM designs (the POWER3 and POWER4 architectures) and extracted nets with more than 10 groups; altogether 126 instances with 11 to 99 groups were extracted.

Random instances. For the random instances we specified the total number of points, the average size of a group and the geometric extension of the groups.

A picture of one chip and one random instance can be seen in Figure 13. RGSMTs for these two problem instances are shown in Figure 14.

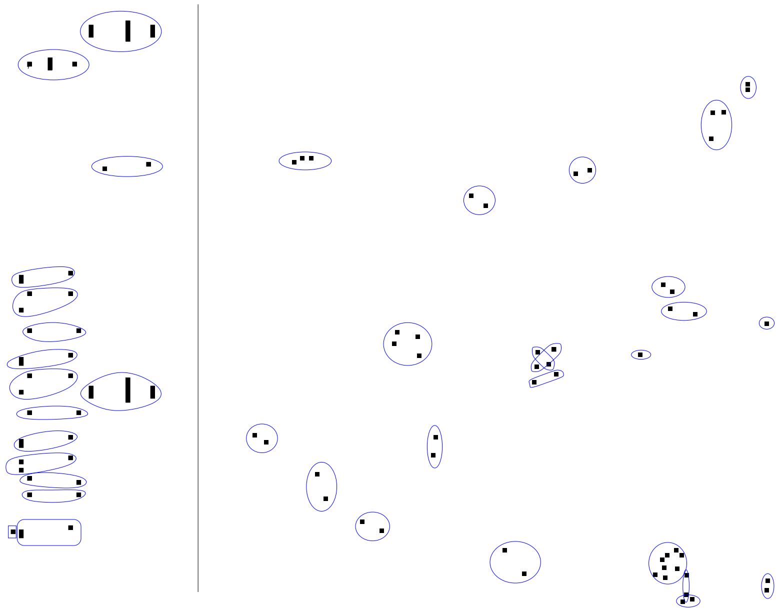


Figure 13: A chip (left; 16 groups) and a random instance (right; 22 groups). Each group is surrounded by a closed curve.

All experiments were performed on a Pentium II 400Mhz PC; we used the operating system Linux 2.2 and compiled the C-code with gcc (version 2.95) and optimization flag "-O3".

The FST generation output can be used in two different ways. Either each FST can be considered to be an hyperedge in an hypergraph with the terminal points as vertices. In order to solve the group problem one super-terminal vertex is added for each group. These super-terminals are connected to the

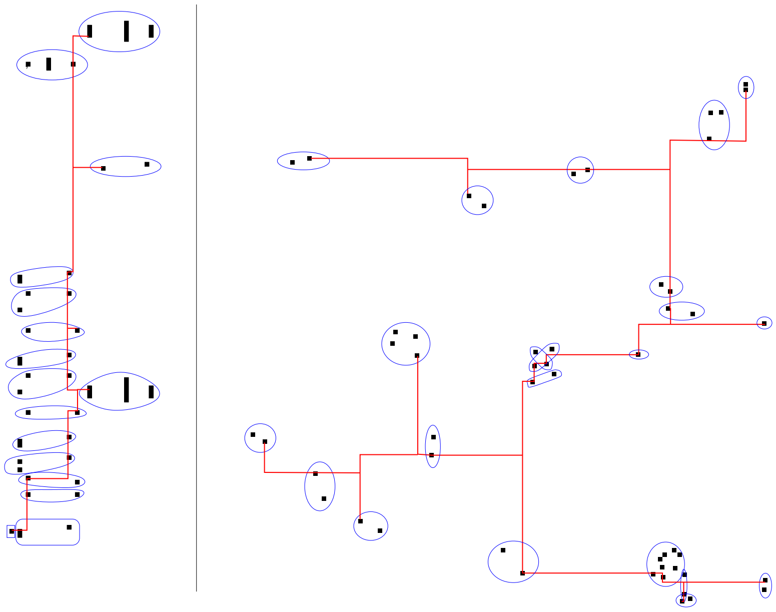


Figure 14: RGSMTs for the two instances in Figure 13.

respective group members with (hyper)edges of high weight. Solving the Steiner tree problem in this hypergraph (where the super-terminals are required to be spanned), solves the group Steiner tree problem.

Alternatively, the set of generated FSTs may overlayed on the Hanan grid and the subgraph of the Hanan grid covered extracted from the grid. This creates an (ordinary) graph for which super-terminal vertices again are added in order to solve the group Steiner tree problem as a Steiner tree problem in graphs.

The Steiner tree in (hyper)graph problems were solved using existing available codes:

GeoSteiner4 Steiner tree in hypergraph solver by Warme and Zachariasen [26]. This code uses an undirected integer programming (IP) formulation of the Steiner tree in hypergraph problem; the IP is solved using LP relaxation and branch-and-cut [29]. An overview of different formulations of the Steiner tree problem is given in [8].

Jack3 Steiner tree in graph solver by Koch and Martin [16]. This code uses a directed IP formulation of the Steiner tree problem that is also solved by LP relaxation and branch-and-cut.

In the following we first present detailed results for the POWER3 instances; in total 64 problem instances with 11 to 99 groups. For each problem size we picked one arbitrary instance from the chip. We return to the POWER4 instances and the random instances in the end of this section.

Table 1 shows the effect of Theorem 1 and Lemma 2 and 3. A considerable fraction of the terminals can be removed, in particular as a result of applying Theorem 1. No CPU-times are included since they are less than 1 second for all instances and therefore negligible compared to the running time of FST generation and the Steiner tree solver.

After applying point reductions we solved the remaining problem using three different methods. The first was to use JACK3 directly on the Hanan grid of the remaining terminal points. Secondly, we solved the reduced Hanan grid problem after FST generation using JACK3. Finally, we used GEOSTEINER4 to solve the hypergraph problem obtained via FST generation.

Table 2 shows the behaviour of FST generation for nets from the POWER3 architecture. It can be seen that the running time of the algorithm is quite small (especially if it is compared to the running time of the Steiner tree in graph solver afterwards). The number of generated FSTs is quite high compared to normal rectilinear Steiner tree problems in the plane. As described in Section 4.4, we might generate exponentially many FSTs (in theory and practice), which can be avoided by generating local Hanan grids. The average Hanan grid reduction is about 60%, so just 40% of the nodes and edges in the Hanan grid remain in the instance we feed to the Steiner tree in graph solver.

Table 3 shows the running time of the graph solvers using the three different types of input. In the last column the speed-up of the runtime comparing the Hanan grid and the reduced grid graph (for all the problems that were solved) can be found. If an instance was not solved within 12 hours, we stopped the run. The table shows that several instances could be solved only with the reduced Hanan grid input. For the smaller instances which could be solved using all methods, we see an average speed reduction of 76.18%. On most instances JACK3 using reduced Hanan grid input was the fastest, but some instances could only be solved using GEOSTEINER4.

The results we got for the nets from the POWER4 chip were comparable (see Table 4). The instances had between 11 and 76 groups; all but 2 could be solved. In most cases the instances were easier to solve than POWER3 instances of comparable size.

For the random instances (Table 5) we get quite different results. Since most of the reductions work especially well for structured instances, we see a completely different behaviour here: The difference between the complete Hanan grid and reduced Hanan grid is very small. We made experiments for 8 different problem types; for each type we made 10 runs with different seeds. The first column shows the number of terminals and the geometric size of the clusters, while the second shows the average number of groups for the 10 instances. The maximum allotted running time was again 12 hours.

6 Conclusion

We presented the first (tailored) exact algorithm for the rectilinear group Steiner tree problem. By applying efficient terminal reductions and Hanan grid reduc-

Groups	Terminals	Theorem 1	Lemma 2 and 3
11	45	35 (22.22 %)	34 (2.86 %)
12	108	37 (65.74 %)	37 (0.00 %)
13	141	73 (48.23 %)	66 (9.59 %)
14	92	66 (28.26 %)	64 (3.03 %)
15	71	57 (19.72 %)	57 (0.00 %)
16	60	49 (18.33 %)	46 (6.12 %)
17	150	85 (43.33 %)	83 (2.35 %)
19	166	75 (54.82 %)	71 (5.33 %)
20	57	55 (3.51 %)	55 (0.00 %)
21	99	78 (21.21 %)	77 (1.28 %)
22	125	74 (40.80 %)	70 (5.41 %)
23	51	51 (0.00 %)	51 (0.00 %)
24	113	86 (23.89 %)	85 (1.16 %)
25	119	85 (28.57 %)	81 (4.71 %)
26	126	95 (24.60 %)	94 (1.05 %)
27	133	108 (18.80 %)	107 (0.93 %)
28	105	83 (20.95 %)	79 (4.82 %)
29	72	67 (6.94 %)	65 (2.99 %)
30	137	112 (18.25 %)	110 (1.79 %)
31	132	85 (35.61 %)	83 (2.35 %)
33	123	106 (13.82 %)	106 (0.00 %)
34	230	179 (22.17 %)	172 (3.91 %)
36	127	104 (18.11 %)	103 (0.96 %)
37	230	174 (24.35 %)	168 (3.45 %)
38	206	161 (21.84 %)	158 (1.86 %)
39	474	365 (23.00 %)	364 (0.27 %)
41	84	83 (1.19 %)	83 (0.00 %)
42	192	143 (25.52 %)	138 (3.50 %)
43	85	85 (0.00 %)	85 (0.00 %)
45	471	251 (46.71 %)	246 (1.99 %)
48	187	146 (21.93 %)	143 (2.05 %)
49	426	260 (38.97 %)	254 (2.31 %)
50	295	236 (20.00 %)	232 (1.69 %)
52	166	147 (11.45 %)	147 (0.00 %)
53	205	165 (19.51 %)	163 (1.21 %)
55	244	193 (20.90 %)	190 (1.55 %)
56	187	156 (16.58 %)	152 (2.56 %)
60	659	299 (54.63 %)	296 (1.00 %)
62	208	176 (15.38 %)	174 (1.14 %)
64	292	245 (16.10 %)	244 (0.41 %)
66	279	225 (19.35 %)	221 (1.78 %)
67	283	219 (22.61 %)	218 (0.46 %)
69	275	201 (26.91 %)	196 (2.49 %)
70	371	300 (19.14 %)	299 (0.33 %)
71	364	267 (26.65 %)	266 (0.37 %)
73	251	198 (21.12 %)	196 (1.01 %)
74	193	185 (4.15 %)	184 (0.54 %)
75	265	217 (18.11 %)	215 (0.92 %)
76	343	267 (22.16 %)	266 (0.37 %)
78	532	380 (28.57 %)	369 (2.89 %)
79	175	175 (0.00 %)	175 (0.00 %)
80	281	231 (17.79 %)	228 (1.30 %)
83	412	341 (17.23 %)	339 (0.59 %)
84	431	296 (31.32 %)	290 (2.03 %)
85	197	197 (0.00 %)	197 (0.00 %)
86	269	237 (11.90 %)	235 (0.84 %)
88	311	254 (18.33 %)	254 (0.00 %)
91	422	274 (35.07 %)	272 (0.73 %)
92	542	412 (23.99 %)	411 (0.24 %)
94	491	359 (26.88 %)	355 (1.11 %)
95	466	340 (27.04 %)	335 (1.47 %)
96	495	404 (18.38 %)	397 (1.73 %)
98	518	428 (17.37 %)	423 (1.17 %)
99	790	351 (55.57 %)	348 (0.85 %)
Avg.		(23.21 %)	(1.70 %)

Table 1: Point reductions of Theorem 1 and Lemma 2 and 3 (POWER3 instances). Groups: Number of groups. Terminals: Number of terminals in Hanan grid. Theorem 1: Number of terminals after removing free points. Lemma 2 and 3: Number of terminals after additional point reductions.

Groups	FSTs	Nodes	Edges	NodeRed (%)	EdgeRed (%)	CPU
11	448	491	950	73.93 %	76.11 %	0.03
12	181	168	317	50.00 %	53.00 %	0.01
13	1787	565	1121	44.96 %	45.32 %	0.24
14	281	174	340	26.44 %	27.35 %	0.03
15	311	183	347	24.59 %	25.94 %	0.03
16	668	448	862	54.46 %	56.61 %	0.05
17	445	345	689	48.70 %	48.62 %	0.05
19	504	699	1373	72.96 %	74.29 %	0.06
20	865	540	1042	54.63 %	56.43 %	0.11
21	1036	989	1946	76.04 %	77.18 %	0.11
22	763	967	1897	75.90 %	77.28 %	0.09
23	424	209	386	36.84 %	40.41 %	0.04
24	743	1200	2362	78.17 %	79.38 %	0.08
25	653	785	1542	68.66 %	69.65 %	0.19
26	1445	906	1787	55.63 %	56.35 %	6.24
27	2067	1353	2685	72.65 %	73.15 %	0.46
28	910	1536	3012	80.01 %	81.44 %	0.15
29	750	666	1277	63.21 %	65.86 %	0.61
30	2910	1590	3149	70.63 %	71.55 %	0.50
31	2421	1179	2307	72.60 %	74.34 %	0.32
33	2227	1078	2120	59.46 %	60.47 %	18.31
34	10526	3426	6837	63.69 %	63.81 %	30.41
36	1770	1404	2763	69.02 %	70.39 %	0.26
37	8615	2269	4525	55.44 %	55.58 %	11.58
38	2677	1478	2955	59.20 %	59.15 %	0.59
39	2523	983	2176	28.48 %	25.74 %	1.35
41	302	342	635	47.95 %	51.65 %	0.06
42	3625	1967	3894	64.16 %	64.74 %	0.87
43	269	215	382	19.53 %	21.99 %	0.07
45	15096	4995	10001	71.69 %	71.87 %	103.09
48	4412	3164	6252	70.76 %	72.20 %	19.02
49	5637	3109	6260	71.50 %	71.25 %	2.09
50	8548	2900	5822	61.41 %	61.34 %	51.14
52	3681	2104	4156	66.68 %	67.47 %	1.79
53	5585	4229	8383	81.67 %	82.45 %	1.70
55	20580	4122	8187	60.09 %	61.08 %	103.12
56	5657	2747	5423	68.95 %	70.68 %	5.59
60	3347	1155	2398	27.45 %	26.48 %	12.54
62	5045	818	1611	18.09 %	18.31 %	1.22
64	17890	3970	7931	54.11 %	54.48 %	132.82
66	24187	6336	12587	60.21 %	61.40 %	347.19
67	4596	3307	6569	70.15 %	70.73 %	5.36
69	3450	4485	8884	80.91 %	81.75 %	39.13
70	9219	6064	12127	75.79 %	75.83 %	35.13
71	7717	3889	7773	68.60 %	68.94 %	4.10
73	18149	7033	13945	73.13 %	74.09 %	49.71
74	7082	3367	6643	69.74 %	70.78 %	5.66
75	1953	1479	2925	50.71 %	52.31 %	1.01
76	12901	6924	13780	74.57 %	75.54 %	252.03
78	25677	9528	19072	75.38 %	75.59 %	878.26
79	6111	1399	2733	40.46 %	41.64 %	59.53
80	8737	6387	12666	76.66 %	77.65 %	54.15
83	33793	6986	13966	54.65 %	55.46 %	1249.68
84	17438	7968	15877	70.43 %	71.36 %	257.15
85	2297	598	1157	11.71 %	12.10 %	0.99
86	10572	4370	8659	68.88 %	69.89 %	89.44
88	2021	1271	2515	41.54 %	43.98 %	1.44
91	10636	4591	9131	70.75 %	71.59 %	35.87
92	20002	3680	7454	52.04 %	51.53 %	29.41
94	16879	7414	14809	73.35 %	74.10 %	179.15
95	12546	10931	21787	78.01 %	79.49 %	1026.84
96	24179	10379	20758	75.74 %	75.99 %	783.51
98	22617	7900	15847	71.33 %	71.32 %	63.21
99	26391	6739	13465	69.19 %	69.76 %	50.39
Avg.				60.53 %	61.47 %	

Table 2: FST Generation and Hanan grid reduction (POWER3 instances). Groups: Number of groups. FSTs: Number of FSTs generated. Nodes: Number of nodes in Hanan grid after FST generation. Edges: Number of edges in Hanan grid after FST generation. NodeRed: Node reduction in percent. EdgeRed: Edge reduction in percent. CPU: FST generation running time (in seconds).

Groups	GeoSteiner4	Jack3-Hanan	Jack3-FST	Red
11	5.02	14.74	1.19	91.93%
12	5.17	0.82	0.44	46.60%
13	247.90	59.27	26.41	55.44%
14	23.41	1.68	1.48	12.11%
15	14.56	1.62	1.03	36.58%
16	7.91	11.80	6.37	46.04%
17	98.89	15.82	2.06	86.98%
19	31.52	126.06	7.79	93.82%
20	21.21	41.98	14.87	64.58%
21	76.15	105.26	4.68	95.55%
22	29.07	349.48	20.58	94.11%
23	3.72	1.57	1.02	35.11%
24	70.32	597.29	9.25	98.45%
25	48.32	120.93	17.30	85.69%
26	176.33	153.60	61.45	59.99%
27	350.35	342.12	107.25	68.65%
28	33.25	565.62	25.32	95.52%
29	9.93	56.77	10.90	80.80%
30	314.52	1577.58	72.40	95.41%
31	103.76	1400.03	64.88	95.37%
33	184.03	168.76	54.75	67.56%
34	-	133430	16180.5	87.87%
36	177.89	1117.46	38.24	96.58%
37	35997.77	34612.17s	8881.83	74.34%
38	1723.53	7547.32	1309.89	82.64%
39	-	34648.4	22089.8	36.25%
41	11.42	19.61	1.74	91.13%
42	1235.04	35311.1	2102.77	94.05%
43	21.22	3.17	1.13	64.38%
45	-	-	13831.8	-
48	1758.99	-	884.86	-
49	10790.89	-	2562.2	-
50	-	-	21066.3	-
52	1938.16	-	-	-
53	668.01	-	179.49	-
55	-	-	-	-
56	2747.19	-	2194.92	-
60	-	-	-	-
62	1576.06	-	3355.68	-
64	-	-	-	-
66	-	-	-	-
67	6804.45	-	-	-
69	1844.85	-	332.15	-
70	8451.18	-	8577.65	-
71	8833.73	-	8114.51	-
73	-	-	-	-
74	3335.65	-	2393.81	-
75	1019.87	-	-	-
76	-	-	-	-
78	-	-	-	-
79	3462.83	-	818.19	-
80	-	-	6114.11	-
83	-	-	-	-
84	-	-	-	-
85	3187.05	-	301.4	-
86	-	-	-	-
88	1322.86	-	2817.62	-
91	-	-	-	-
92	-	-	26515.5	-
94	-	-	35188.7	-
95	-	-	-	-
96	-	-	-	-
98	-	-	-	-
99	-	-	-	-
Avg.				76.18%

Table 3: Running time (in seconds) of Steiner tree in graph solvers (POWER3 instances). Groups: Number of groups. GeoSteiner4: Running time of GeoSteiner4. Jack3-Hanan: Running time of Jack3 using complete Hanan grid. Jack3-FST: Running time of Jack3 using reduced Hanan grid. Red: Reduction of running time in percent.

Groups	GeoSteiner4	Jack3-Hanan	Jack3-FST	Red
11	7.30	3.09	1.05	66.00 %
13	4.06	10.12	0.47	95.35 %
14	7.53	3.75	1.94	48.29 %
15	20.36	19.76	2.68	86.44 %
16	222.02	161.20	18.65	88.43 %
17	26.13	363.34	3.89	98.93 %
18	6.42	180.08	4.05	97.75 %
19	3.00	21.73	0.73	96.64 %
21	838.50	31751.56	-	-
22	1033.84	107.24	22.35	79.16 %
23	144.09	33.87	9.88	70.83 %
24	173.01	26451.60	98.47	99.63 %
25	167.09	1300.08	89.35	93.13 %
26	2950.21	1034.85	122.22	88.19 %
27	60.24	106.55	21.35	79.96 %
28	1679.52	112.61	34.59	69.28 %
29	175.13	237.25	28.56	87.96 %
30	5226.59	4017.00	330.20	91.78 %
31	7535.11	1417.46	237.19	83.27 %
32	3457.47	298.03	112.66	62.20 %
33	77.74	79.80	18.71	76.55 %
34	103.15	142.33	43.22	69.63 %
35	1484.64	2533.55	501.34	80.21 %
36	1989.72	723.56	338.89	53.16 %
37	1828.32	3811.27	525.65	86.21 %
38	85.20	76.60	24.88	67.52 %
39	958.98	34860.52	144.48	99.59 %
40	16547.10	18097.72	1874.79	89.64 %
41	21212.69	-	731.45	-
42	-	12935.20	2306.72	82.17 %
43	4696.59	7938.64	117.80	98.52 %
44	2478.20	13066.60	49.37	99.62 %
45	1739.71	486.86	144.57	70.31 %
46	8307.46	5819.80	1586.06	72.75 %
47	1684.38	7591.80	46.75	99.38 %
48	86.89	34612.20	27.11	99.92 %
49	2874.04	23731.80	95.60	99.60 %
50	2666.39	-	166.58	-
51	15572.80	-	524.74	-
52	4286.21	-	207.11	-
53	2171.33	-	251.86	-
54	21009.99	-	917.15	-
55	6832.47	-	256.47	-
56	3918.02	-	641.56	-
58	7013.08	-	655.87	-
59	27290.58	-	565.57	-
60	4034.60	-	246.35	-
61	1801.74	-	124.84	-
62	40686.87	-	553.07	-
63	-	-	-	-
64	3941.42	-	203.53	-
66	3181.76	-	384.36	-
67	-	-	2536.79	-
68	7580.53	-	817.75	-
69	4490.96	-	136.05	-
70	1201.00	-	45.47	-
71	8783.01	-	193.55	-
72	-	-	-	-
73	-	-	24158.60	-
74	3116.85	-	672.13	-
75	-	-	597.89	-
76	5334.21	-	287.43	-
Avg.				83.66 %

Table 4: Running time (in seconds) of Steiner tree in graph solvers (POWER4 instances). (The instance with 21 groups could not be solved using Jack3 for unknown technical reasons.) Groups: Number of groups. GeoSteiner4: Running time of GeoSteiner4. Jack3-Hanan: Running time of Jack3 using complete Hanan grid. Jack3-FST: Running time of Jack3 using reduced Hanan grid. Red: Reduction of running time in percent.

Instance	Groups	FSTs	Nodes	Edges	NodeRed	EdgeRed	Jack3-FST
20/s	10.0	213.4	88.1	144.1	11.72%	14.55%	0.36
20/m	10.0	288.9	109.5	185.3	9.65%	11.53%	0.83
50/s	24.2	1001.7	296.3	512.5	8.30%	9.94%	15.33
50/m	24.2	1822.0	448.9	810.6	5.53%	6.32%	159.34
40/s	10.1	1555.1	385.6	716.4	2.96%	6.19%	30.87
40/m	10.1	1654.5	380.6	705.5	3.56%	8.58%	46.47 (9/10 solved)
100/s	24.4	8624.8	1530.6	2831.4	1.74%	3.83%	3930.85 (8/10 solved)
100/m	24.4	9684.8	2098.2	3910.7	1.31%	2.87%	28640.27 (1/10 solved)
Avg.					5.60%	7.97%	

Table 5: Results for random instances (averages over 10 instances for each problem type). Running times in seconds. Instance: Number of terminals (s=small cluster size, m=medium cluster size). Groups: Number of groups. FSTs: Number of FSTs generated. Nodes: Number of nodes in Hanan grid after FST generation. Edges: Number of edges in Hanan grid after FST generation. NodeRed: Node reduction in percent. EdgeRed: Edge reduction in percent. Jack3-FST: Running time of Jack3 using reduced Hanan grid.

tions based on the generation of full Steiner trees, VLSI instances with as many as 94 groups could be solved to optimality.

One motivation for studying exact algorithms for this problem is its fundamental applications in VLSI design. Given the ability to solve problem instances of significant size to optimality, practical heuristics can be evaluated with much better confidence. Preliminary experiments have shown that Prim-like heuristics for the problem can be more than 20% from optimum on typical VLSI instances. Also, the techniques presented here can be used to compute good lower bounds quickly; these lower bounds can provide a tight lower bound on the necessary routing length for the complete chip.

Acknowledgments

The authors would like to thank Thorsten Koch and Alexander Martin for making the Jack3 code available to us. We also thank David M. Warme for his contribution concerning GeoSteiner4. We thank Sven Peyer, David Pisinger, Jens Vygen and Jürgen Werber for reading and commenting on drafts of this paper. The research was made during a long-term visit of the first author at the Institute for Discrete Mathematics, University of Bonn; the research was partially supported by the European Community (DONET, contract number ERB TMRX-CT98-0202) and the Danish Natural Science Research Council (contract number 9901806).

References

[1] S. Arora. Polynomial Time Approximation Schemes for Euclidean TSP and other Geometric Problems. In *Proc. 37th Ann. Symp. on Foundations of Computer Science*, pages 2–13, 1996.

[2] C. D. Bateman, C. H. Helvig, G. Robins, and A. Zelikovsky. Provably-Good Routing Tree Construction with Multi-Port Terminals. In *ACM/SIGDA International Symposium on Physical Design*, 1997.

[3] M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via Trees: Deterministic Approximation Algorithms for Group Steiner Trees and k-median. In *Proc. Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 114–123, 1998.

[4] U. Feige. A Threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM*, 45(4):634–652, 1998.

[5] J. L. Ganley and J. P. Cohoon. Optimal Rectilinear Steiner Tree Routing in the Presence of Obstacles. Technical Report CS-94-03, Department of Computer Science, University of Virginia, January 1994.

[6] M. R. Garey and D. S. Johnson. The Rectilinear Steiner Tree Problem is NP-Complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[7] N. Garg, G. Konjevod, and R. Ravi. A Polylogarithmic Approximation Algorithm for the Group Steiner Tree Problem. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 253–259, 1998.

[8] M. X. Goemans and Y. S. Myung. A Catalog of Steiner Tree Formulations. *Networks*, 23:19–28, 1993.

[9] M. Grötschel, A. Martin, and R. Weismantel. The Steiner Tree Packing Problem in VLSI Design. *Mathematical Programming*, 78:265–281, 1997.

[10] M. Hanan. On Steiner's Problem with Rectilinear Distance. *SIAM Journal on Applied Mathematics*, 14(2):255–265, 1966.

[11] C. S. Helvig, G. Robins, and A. Zelikovsky. Improved Approximation Bounds for the Group Steiner Problem. In *Proc. Conference on Design Automation and Test in Europe*, pages 406–413, 1998.

[12] A. Hetzel. *Verdrahtung im VLSI-Design: Spezielle Teilprobleme und ein sequentielles Lösungsverfahren*. PhD thesis, Institute for Discrete Mathematics, University of Bonn, 1995.

[13] F. K. Hwang. On Steiner Minimal Trees with Rectilinear Distance. *SIAM Journal on Applied Mathematics*, 30:104–114, 1976.

[14] E. Ihler. The Rectilinear Class Steiner Tree Problem for Intervals on Two Parallel Lines. *Mathematical Programming*, 63(3):281–296, 1994.

[15] E. Ihler, G. Reich, and P. Widmayer. Class Steiner Trees and VLSI-Design. *Discrete Applied Mathematics*, 90:173–194, 1999.

[16] T. Koch and A. Martin. Solving Steiner Tree Problems in Graphs to Optimality. *Networks*, 33:207–232, 1998.

[17] G. Konjevod and R. Ravi. An Approximation Algorithm for the Covering Steiner Problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 338–344, 2000.

[18] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, Chichester, 1990.

[19] G. Reich and P. Widmayer. Beyond Steiner's Problem: A VLSI Oriented Generalization. In *Lecture Notes in Computer Science*, pages 196–210. Springer, 1990.

[20] J. S. Salowe and D. M. Warme. Thirty-Five-Point Rectilinear Steiner Minimal Trees in a Day. *Networks*, 25(2):69–87, 1995.

[21] M. Sarrafzadeh and C.K. Wong. *An Introduction to VLSI Physical Design*. McGraw-Hill, New York, 1996.

[22] N. A. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Boston, third edition, 1999.

[23] P. Slavík. *Approximation Algorithms for Set Cover and Related Problems*. PhD thesis, University of New York at Buffalo, 1998.

[24] E. Uchoa and M. Poggi de Aragao. Vertex-Disjoint Packing of Two Steiner Trees: Polyhedra and Branch-and-Cut. In *IPCO: 7th Integer Programming and Combinatorial Optimization Conference, Lecture Notes in Computer Science*, volume 1610, pages 439–452, 1999.

[25] D. M. Warme, P. Winter, and M. Zachariasen. Exact Algorithms for Plane Steiner Tree Problems: A Computational Study. In D.-Z. Du, J. M. Smith, and J. H. Rubinstein, editors, *Advances in Steiner Trees*, pages 81–116. Kluwer Academic Publishers, Boston, 1999.

[26] D. M. Warme and M. Zachariasen. Personal communication, 2000.

[27] M. Zachariasen. Rectilinear Full Steiner Tree Generation. *Networks*, 33:125–143, 1999.

[28] M. Zachariasen. A Catalog of Hanan Grid Problems. Technical Report 00892, Institute for Discrete Mathematics, University of Bonn, 2000.

[29] M. Zachariasen and P. Winter. Obstacle-Avoiding Euclidean Steiner Trees in the Plane: An Exact Algorithm. In *Lecture Notes in Computer Science*, volume 1619, pages 282–295. Springer, 1999.

Appendix

We give a proof that computing the group bottleneck Steiner distance for a single pair of terminals $z_i, z_j \in Z$ is NP-hard for general (integer) distance functions. This proof arose from fruitful discussions with Martin Skutella and Jens Vygen. We conjecture that computing group bottleneck Steiner distances under the L_1 metric is NP-hard, too.

Recall the definition from Section 4.3:

$$BSD(z_i, z_j) = \max_{V \in \mathcal{V}, z_i, z_j \in V} BSD_V(z_i, z_j)$$

where $BSD_V(z_i, z_j)$ is the bottleneck Steiner distance between z_i and z_j given that we span the valid set V (and $z_i, z_j \in V$).

Clearly, the corresponding decision problem is:

Given an integer $L \geq 0$, is $BSD(z_i, z_j) > L$?

This question can be restated as follows. Construct an undirected graph $G = (Z, E)$ having the set of terminals as its vertices. Two vertices u and v share an edge if and only if $|uv| \leq L$. The equivalent decision problem $CISP$ (connected induced subgraph problem) is now:

Does there exist a valid set $V' \in \mathcal{V}$ such that z_i and z_j are *not* connected in the graph induced by V' in G?

To see why the two problems are equivalent, first assume that z_i and z_j are connected in G for all valid sets $V \in \mathcal{V}$. This means that there is a path from z_i to z_j for which the maximum distance is shorter than or equal to L for any valid set, or $BSD(z_i, z_j) \leq L$.

If there exists a valid set $V' \in \mathcal{V}$ for which z_i and z_j are *not* connected, this implies that there is no path in G from z_i to z_j containing only edges having a distance smaller than or equal to L. Thus $BSD(z_i, z_j) > L$.

Now we prove NP-completeness by a reduction from 3SAT: given $\bigwedge_{i=1}^{m}(x_{a_i} \vee x_{b_i} \vee x_{c_i})$ with n variables, we can construct a $CISP$ instance for which z_i and z_j are *not* connected for some valid set $V' \in \mathcal{V}$ if and only if the 3SAT instance is satisfiable.

Define the graph $G = (V, E_1 \cup E_2 \cup E_3)$ as follows:

$$\begin{array}{lcl} V & := & \{z_i, z_j, x'_1, \neg x'_1, \ldots, x'_n, \neg x'_n, x_1, \ldots, x_{3m}\} \\ E_1 & := & \bigcup_{k=1}^{n}\{z_i, x'_k\} \cup \bigcup_{i=1}^{n}\{z_i, \neg x'_k\} \\ E_2 & := & \bigcup_{k=1}^{m}\{\neg x'_{a_k}, x_{3k-2}\} \cup \{\neg x'_{b_k}, x_{3k-1}\} \cup \{\neg x'_{c_k}, x_{3k}\} \\ E_3 & := & \bigcup_{k=1}^{3m}\{z_j, x_k\} \end{array}$$

That is, E_1 connects z_i with all variable-nodes, E_3 connects z_j with all clause-nodes and E_2 connects each clause-node with the negation of the variable in the variable nodes. The groups are defined by: each literal and its negation is a two-element group, each clause is a three-element group and z_i and z_j form two single element groups.

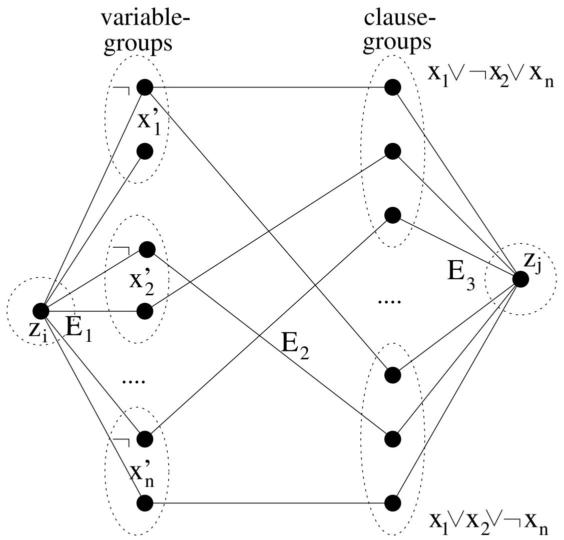


Figure 15: Illustration of NP-completeness proof.

Now the following holds: There exists a valid set $V' \in \mathcal{V}$ such that z_i and z_j are not connected in the induced subgraph of G if and only if the 3SAT problem is satisfiable.

Assume the 3SAT problem has a solution. We will pick a representative from each group such that z_i and z_j become disconnected. From the variable-groups (x'_k and $\neg x'_k$) we pick the one which is in the solution (so if x_l is true in the solution, we pick x'_l). Then we know that in every clause-group there is at least one node that has no connection to a picked variable-node (otherwise this literal would be not true contradicting the fact that we have a solution). We pick these nodes from the clause-groups and z_i and z_j become disconnected.

Assume now the graph is not connected for some valid set. Then each of the literals for this set of representatives is fulfilled and we have a solution for the 3SAT-problem. This finishes the NP-completeness proof of $CISP$.