On Steiner Minimal Tree in Grid Graphs and Its Application to VLSI Routing

Michael Kaufmann[1], Shaodi Gao[2], K. Thulasiraman[2]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübigen, Sand 13, 72076 Tübingen, Germany
[2] Department of Electronic and Computer Engineering, Concordia University, 1455 De Maisonneuve Blvd. West, H3G 1M8, Montreal, Canada

Abstract. In this paper we present an algorithm for Steiner minimal trees in grid graphs with all terminals located on the boundary of the graph. The algorithm runs in $O(k^2 * \min\{k^2 \log k, n\})$ time, where k and n are the numbers of terminals and vertices of the graph, respectively. It can handle non-convex boundaries and is the fastest known for this case. We also describe a new approach to the homotopic routing problem in VLSI layout design, which applies our Steiner tree algorithm to construct minimum-length wires for multi-terminal nets.

1 Introduction

Given a set K of vertices, called *terminals*, in a graph $G(V, E)$, the Steiner tree problem is to find a minimum-length tree that spans all vertices in K. This minimum-length tree is called a *Steiner minimal tree*, while vertices with degree ≥ 3 in the tree are called *Steiner vertices*. This problem has been extensively studied for many years because of its wide variety of applications, such as communication networks and VLSI layout design. While the general problem is known to be NP-complete [4], it can be solved in polynomial time when G is planar and all terminals lie on one face of G.

In this paper we first consider a special case of the Steiner tree problem in which (1) $G(V, E)$ is a grid graph without holes, i.e., every finite face has exactly four incident vertices, and (2) all terminals are located on the boundary of the infinite face of G. For simplicity, we call it *the boundary of* G. This boundary is a rectilinear polygon, which is allowed to be non-convex (cf. Fig. 1). We present a Steiner tree algorithm that runs in $O(k^2 * \min\{k^2 \log k, n\})$ time, where $k = |K|$ and $n = |V|$. Section 2 is a general description of the algorithm. Section 3 gives the implementation of the algorithm and analyses its time complexity.

The Steiner tree problem in grid graphs is very important to formulate VLSI routing problems, where the routing area of a single net is often non-convex. The previously known algorithms which can handle non-convex boudaries were given by Provan [8] and by Erickson et al. [3]. Their time complexities are $O(n^2k^2)$ and $O(nk^3 + (n \log n)k^2)$, respectively. Richards and Salowe [9] developed an $O(k\nu^4)$-time algorithm, where ν is the number of the boundary sides of the graph. However, their algorithm can only handle grid graphs with convex boundaries.

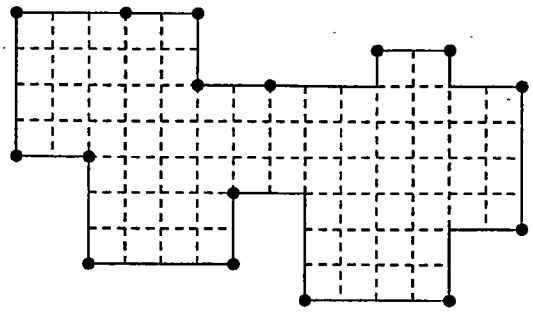


Fig. 1. An example of the Steiner tree problem in a grid graph with terminals (solid dots) lying on the boundary.

In Section 4 we extend our result to construct a collection of Steiner minimal trees in a grid graph, which is allowed to have holes. While terminals of the Steiner trees may lie on the boundary of the graph as well as on those of the holes, the topology of each tree is given. This problem is called *homotopic routing* in VLSI layout design. The goal is to find vertex-disjoint Steiner minimal trees for the given collection of terminal sets. Homotopic routing is first introduced by Leiserson/Maley [6], but they only dealt with problems where each terminal set has cardinality of 2. We present an efficient algorithm for terminal sets of cardinality ≥ 2 by using the Steiner tree algorithm described in Section 2 and Section 3.

2 The Dynamic Programming Approach

Without loss of generality, we assume that the boundary of G is a simple polygon P. The case in which the boundary is not a simple polygon can be solved by partitioning G and then solving several simple-polygon instances. Since all terminals are located on P, we can define an *interval* $[u, v)$, for $u, v \in K$, to be is the set of terminals, including u but not v, visited by traversing P counterclockwise from u to v. Intervals $(u, v]$ and $[u, v]$ are defined similarly.

The dynamic-programming approach was proposed by Dreyfus and Wagner, which they used to solve the Steiner tree problem in general graphs. The key observation is that a Steiner minimal tree for K can be split at any of its vertices v into two Steiner minimal trees for $I \cup v$ and $(K \backslash I) \cup \{v\}$. In the case that G is a planar graph and all terminals lie on P, both I and $K \backslash I$ are intervals of K.

For each interval $[a, b)$ of K and each vertex v of G, let $C(v, [a, b))$ represent the length of a Steiner minimal tree, called a *C-tree*, for terminal set $[a, b) \cup \{v\}$. Thus the length of a Steiner minimal tree for K is $C(v, K \backslash \{v\})$, for any choice $v \in K$. For $[a, b)$ of cardinality at least two, let $B(v, [a, b))$ represent the length of a Steiner minimal tree for terminal set $[a, b) \cup \{v\}$ subject to the constraint that v has degree ≥ 2 in the tree. This tree is called a *B-tree*.

The computation of $B(v,[a,b))$ and $C(v,[a,b))$ proceeds in order of the cardinality of the interval $[a,b)$. $B(v,[a,b))$ can be computed as the sum of the lengths of two Steiner minimal trees for the subsets of $[a,b) \cup \{v\}$. That is

$$B(v,[a,b)) = \min_{a \neq x \in [a,b)} \{C(v,[a,x)) + C(v,[x,b))\} \quad (1)$$

The values of $C(v,[a,b))$ can be computed using the values of $B(u,[a,b))$ for all $u \in V$.

$$C(v,[a,b)) = \min_{u \in V} \{B(u,[a,b)) + d(u,v)\} \quad (2)$$

where $d(u,v)$ is the shortest-path distance between u and v in G. The initial conditions are $C(v,) = 0$ and $B(v,) = 0$ for all v. At the end of the computation, the length of a Steiner minimal tree will be $C(v, K \backslash \{v\})$ for any $v \in K$. The tree itself can be recovered by retaining a record of the corresponding B- and C-trees.

The number of B- and C-values to be computed is of the same order as the number of possible choices of vertices $v \in V$ and intervals $I \subseteq K$, which is $O(nk^2)$. A simple-minded approach requires $O(k)$ time for computing a B-value and $O(n)$ time for a C-value, which leads to a total running time of $((n+k)nk^2)$. In the following section we specialize the algorithm to the case of grid graphs without holes and describe a more efficient way to compute the B- and C-values.

3 Computation of B- and C-values

3.1 Computation of B-values

For a given problem instance, there are normally more than one Steiner trees of minimum length. Let Λ represent the set of Steiner minimal trees for the terminal set K in a grid graph G. We break ties in Λ by choosing trees with edges as far to the "left" as possible. More exactly, we prefer trees in $\Lambda_1 = \{\lambda | \lambda \in \Lambda, \sum X(l) \times len(\lambda \cap l)$ is minimized$\}$, where $len(\lambda \cap l)$ is the length of the intersection of tree λ with vertical line l, and $X(l)$ is the x-coordinate of l. The weighted sum $\sum X(l) \times len(\lambda \cap l)$ is called the *leftness* of tree λ. Similarly, we define the *topness* of tree λ to be the weighted sum $\sum Y(l) \times len(\lambda \cap l)$, where l is a horizontal line, and $Y(l)$ is the y-coordinate of l. We further break ties in Λ_1 by choosing trees of the maximal topness: $\Lambda_2 = \{\lambda | \lambda \in \Lambda_1, \sum Y(l) \times len(\lambda \cap l)$is maximized$\}$. The following lemma is originally proven by Hanan [5], but our claim is stronger.

Lemma 1. *Let v be a vertex of degree ≥ 2 in Steiner tree $\lambda \in \Lambda_2$. Then each straight-lined path in λ which contains v can reach the boundary of G and intersect at least one terminal.*

Proof. Let e be the grid edge incident to v and p the longest straight-lined path containing e. In the following we show if neither endpoint of p is a terminal, then λ is not in Λ_2.

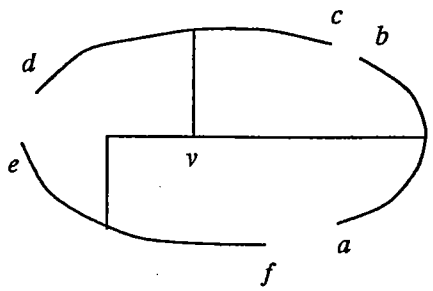


Fig. 2.

Assume that p is a vertical line (the case of horizontal lines can be treated similarly). We can transform tree λ into another tree λ' by moving p to the left or to the right while keeping the connections between p and the horizontal edges that are incident to p in λ. If p moves to the left (right), then the length of every horizontal edge incident to p from the left (right) side of p increases by the same amount, while the length of every edge on the other side decreases by the same amount. This is because p does not contain any terminal. If there are more incident edges on the right side than on the left side, then we move p to the right and get a tree λ' whose length is smaller than that of λ. Otherwise, we move p to the left to get a tree λ' whose length is not larger than that of λ but whose leftness is smaller than that of λ. In both cases λ does not have the properties required by Λ_2. □

We call a B-tree for interval $[a, b)$ and vertex v *restricted* if (1) there are two straight-lined paths p_y and p_z in the subtree from v to two terminals $y, z \in [a, b)$ and (2) the corner formed by p_y and p_z which faces interval $[y, z] \subset [a, b)$ is 90° or 180°. From Lemma 1, we can obtain the following lemma.

Lemma 2. *Let λ be a Steiner minimal tree in Λ_2. Then λ and any of its subtrees can be split into two subtrees at some vertex $v \in V \cap \lambda$ such that if a subtree is a B-tree, it is restricted.*

According to the above lemma, we only need to consider all the restricted B-trees in order to construct a Steiner minimal tree $\lambda \in \Lambda_2$. In Fig. 2 the B-tree for interval $[c, f]$ and vertex v is not restricted. But we can split the Steiner tree into a restricted B-tree for interval $[a, d]$ and v and a C-tree for $[e, f]$ and v. The following lemma suggests a more efficient way to construct all restricted B-trees for a vertex $v \in V$.

Lemma 3. *Let λ be a restricted B-tree for interval $[a, b)$ and vertex $v \in V$ with two straight-lined paths from v intersecting terminals y and z. If the length of the restricted B-tree for $[y, z]$ and v satisfies the equation $B(v, [y, z]) = C(v, [y, x)) + C(v, [x, z])$, for some terminal $x \in [a, b)$, then the length of the restricted B-tree for $[a, b)$ can be obtained by $B(v, [a, b)) = C(v, [a, x)) + C(v, [x, b))$.*

Proof. In the B-tree for $[a, b)$, the two straight-lined paths from v completely separate terminals in $[y, z]$ from those in $[a, b)\backslash[y, z]$ (cf. Fig. 3). Only the terminals in $[y, z]$ determine the subtree that spans these terminals and v. Therefore this subtree should be identical to the restricted B-tree for $[y, z]$ and v. That means both restricted B-trees have the same separating terminal x. □

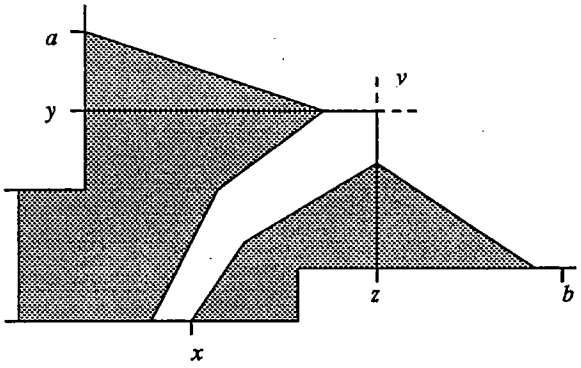


Fig. 3. The computation of the B-value of $\{v, [a, b)\}$.

For each vertex $v \in V$, there are up to six special intervals $[y_i, z_i]$ with terminals y_i and z_i having either the same x-coordinate or the same y-coordinate as v. For each of these interval $[y_i, z_i]$ and v, we compute the restricted B-tree, which produces a separating terminal $x_i \in [y_i, z_i]$. According to Lemma 3, a restricted B-tree for v and an interval $I \subset K$ can be constructed by choosing one of these separating terminals.

During the computation of the B-tree for v and a special interval $[y, z]$, we have to try all possible terminals in $[y, z]$ to get the B-value $B(v, [y, z]) = \min_{x \in (y,z)}\{C(v, [y, x)) + C(v, [x, z])\}$. It takes $O(k)$ time for finding the separating terminals regarding a vertex $v \in V$, and $O(kn)$ time for finding all separating terminals. After that the computation of $B(v, [a, b))$ for a vertex $v \in V$ and an interval $[a, b) \subset K$ requires $O(1)$ time, because we have to try at most six possible separating terminals. Therefore, the total running time for the computation of B-values is $(k^2 n)$.

3.2 Computation of C-values

For this part of the algorithm, we follow the idea proposed by Erickson et al. [3]. The computations of $C(v, I)$ for different choices of vertex $v \in V$, but for the same choice of interval I will be carried out simultaneously. We direct the grid graph by replacing each edge with a pair of directed arcs. We then augment

the graph with a fictitious source s and place an arc with cost $B(u, I)$ from s to each vertex $u \in V$. An algorithm for finding shortest paths from s can determine values $C(v, I) = B(v, I) + d(u, v)$ for all choices of v. The shortest-path algorithm requires $O((|V| + |E|) \times f$ time for graph $G(V, E)$, where f is the time for updating a path length. If heaps are used as data structures, then $f = \log n$. Since all values of $B(v, I)$ and $C(v, I)$ do not exceed n in our application, we simply use n buckets as the data structure with each bucket i, $1 \leq i \leq n$, containing vertices with value i. Then updating the value of a vertex or fetching a vertex of minimum value takes $O(1)$ time. Therefore it takes $O(n)$ time to compute all C-values for one interval and $O(k^2 n)$ time for all intervals.

3.3 The overall time complexity

The time complexity of our Steiner tree algorithm is $O(k^2 n)$ according to the above analysis. In the case $k^2 \ll n$, we can achieve even better result. Lemma 1 implies that by computing B- and C-values we only need to consider the vertices $v \in V$ which have the same x- or y-coordinates as terminals in K. There are $O(k^2)$ vertices satisfying this condition. That means in the time analysis n can be replaced by k^2. Then the computation of B-values takes $O(k^2)$ time. By computing C-values we have to use a heap as the data structure. The time for all C-values for one interval is $O(k^2 \log k)$. Therefore we can also solve the problem in $O(k^4 \log k)$ time in case $k^2 \ll n$.

Lemma 4. *If all terminals are located on the boundary of a grid graph without holes, then the problem of finding a Steiner minimal tree in the graph can be solved in time* $O(k^2 * \min\{k^2 \log k, n\})$.

4 The Steiner Tree Algorithm in Homotopic Routing

In this section we apply the above described Steiner tree algorithm to construct minimum-length interconnections for a collection of terminal sets in a grid graph. In this case, the grid graph may contain holes, i.e., finite faces enclosed by more than four grid edges. Terminals are located on the boundary of the graph as well as on those of the holes. The interconnection topology for each terminal set is given. This problem is called *homotopic routing*, which has found more and more applications in VLSI layout design [1, 7].

4.1 Definitions and previous results

Formally, the problem of homotopic routing is given by a *sketch* $S = (M, W)$ which consists of a set M of *modules*, and a set W of *nets*. Modules represent circuit components and each net define a set of terminals to be connected. The topology of each net is given by a simple loop that connects all terminals of the net. This loop may not cross or enclose any modules. Homotopic routing transforms the loop of each net into a Steiner minimal tree in the *routing graph*

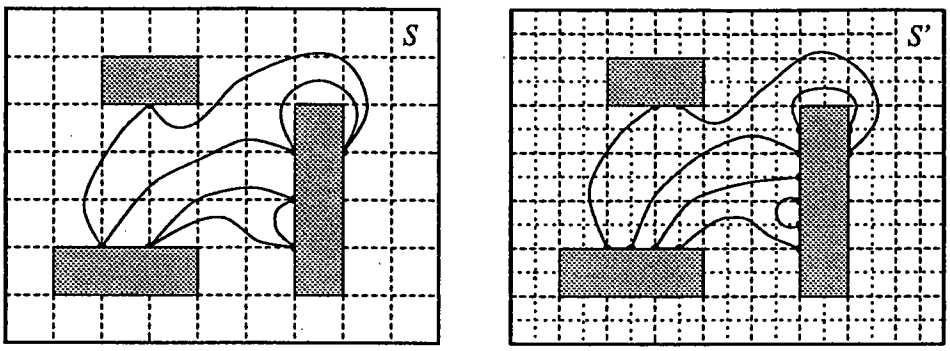


Fig. 4. Transforming a sketch of multi-terminal nets into a sketch of two-terminal nets.

$G = (V, E)$ which can be obtained from the underlying rectilinear grid by removing all grid points and grid edges covered by the modules (cf. Fig. 4). In order to maintain the topology of each net, any part of its loop may not be moved over modules during the transformation. The Steiner trees for two different nets must be vertex-disjoint.

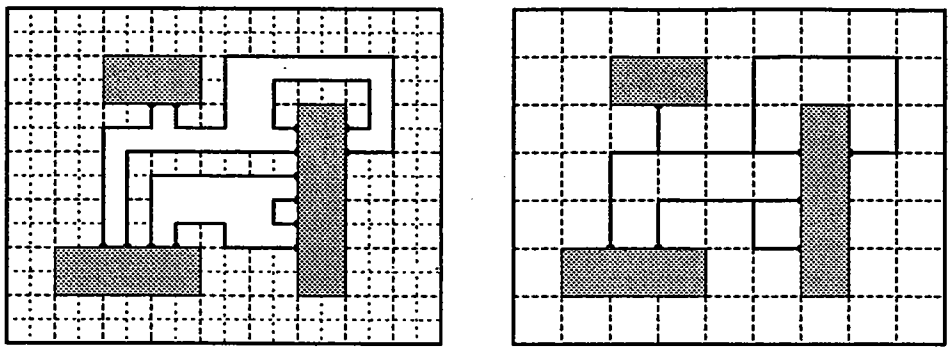
Leiserson/Maley [6] proposed an efficient algorithm to solve the homotopic routing problem for two-terminal nets. In the two-terminal case, the loop of each net degenerates into a simple curve. The algorithm transforms these curves into a set of vertex-disjoint paths in the routing graph. We summarize their results in the following lemma.

Lemma 5. *For the homotopic routing problem of two-terminal nets, the Leiserson/Maley algorithm can construct minimum-length solutions in* $O(n^2 \log n)$ *time.*

4.2 The routing algorithm for multi-terminal nets

To solve the multi-terminal net problem, we first split each k-terminal net into k two-terminal nets, which is called *subnets* of the k-terminal net. Then we apply the Leiserson/Maley algorithm to find a solution for the modified problem. In the solution, the k two-terminal subnets of a k-terminal net define a grid graph without holes. A Steiner minimal tree in this graph is a minimum-length interconnection for these k terminals in the routing graph. Let $S = (M, W)$ be the input sketch of the homotopic routing problem, and $G = (V, E)$ be the underlying routing graph with $|V| = n$. The algorithm consists of the following four steps.

Step 1: Transform a sketch S with multi-terminal nets into a sketch S' with two-terminal nets. We first refine the routing graph G by inserting an *additional line* between every two horizontal and vertical grid lines in G (cf. Fig. 4). Let the resulting graph be G'. Each edge in G' has half unit length. For each terminal,



(1) (2)

Fig. 5. (1) A detailed routing in G'. Grid edges in envelopes are omitted. (2) The corresponding routing in G.

we create another terminal on the neighboring additional grid point along the module boundary in G'. Then a k-terminal net will be split into k separated two-terminal nets in S', which can be connected by edges on module boundaries to form a ring. The resulting sketch S' only contains two-terminal nets.

Step 2: We apply the Leiserson/Maley algorithm to S' to find a detailed routing. In the solution, the two-terminal subnets of the same multi-terminal net form a rectilinear polygon (cf. Fig. 5.1). This polygon is called the *envelope* of the multi-terminal net. Envelopes of different multi-terminal nets are area-disjoint, i.e., boundary edges of the envelopes do not cross each other and no envelope encloses any other envelope. This is because the routing algorithm for two-terminal nets does not change the topology of S' and it constructs vertex-disjoint paths. In addition, envelopes in G' are simple polygons, i.e., the boundary lines of the same envelope do not touch each other except at the endpoints.

Step 3: Transform G' back to G by deleting the additional grid lines. Accordingly, we also shrink each envelope by moving every boundary line that is on an additional grid line to the next original grid line (cf. Fig. 5.2). After this operation, every envelope remains connected and area-disjoint from each other, because we shrink the envelopes during transforming G' back to G.

Step 4: We partition G into disjoint subgraphs according to the envelops determined by Step 3 and treat each subgraph separately. The envelope of a multi-terminal net w encloses a connected subgraph G_w of G, which will be used to construct a minimum-length interconnection for w. Since G_w does not contain any modules, it is a grid graph without holes. All the terminals of w lie on the boundary of G_w. Therefore we can find an interconnection for w by using

the Steiner tree algorithm described in Section 2. It takes $O(k_w^2 n_w)$ time for w, where k_w is the number of w's terminals and n_w is the size of G_w.

It is easy to see that Step 1 and Step 3 can be executed in time $O(n)$. In Step 2 the routing algorithm for two-terminal net requires $O(n^2 \log n)$ time according to [6]. In Step 4 the Steiner tree algorithm needs $k_i^2 n_i$ time for each net w_i of k_i terminals in subgraph G_i of size n_i. Therefore Step (D) takes $O(k^2 n)$ time for all the r nets with a total number k of terminals.

Lemma 6. *The problem of homotopic planar routing for multi-terminal nets can be solved in $O(n^2 \log n + k^2 n)$ time, where n is the size of the routing graph and k is the number of terminals. In the solution, the length of every net is minimized.*

5 Conclusion

We have presented an algorithm for finding Steiner minimal tree in grid graphs. This algorithm can also handle non-convex boundaries, and is faster than the previously known algorithms for this case. We also apply the algorithm to construct a collection of Steiner minimal trees for the homotopic routing problem. Our results show that any Steiner tree problem in grid graphs can be solved in polynomial time if the topology is given.

For the case that the boundary of a grid graph is convex, the algorithm by Richards and Salowe [9] can be faster if the number boundary sides is much smaller than the number of vertices of the graph. An obvious open question is how to extend their techniques to the case of non-convex boundaries.

References

1. Dai, W. W., Dayan, T., Staepelaere, D.: Topological routing in SURF: Generating a rubber-band sketch. *Proceedings of the 28th Design Automation Conference* (1991) 39–44
2. Dreyfus, S. E., Wagner, R. A.: The Steiner problem in graphs. *Networks*, **1** (1972) 196–207
3. Erickson, R. E., Monma, C. L., Veinott, A. F.: Send-and-split method for minimum-cost network flows. *Math. Oper. Res.*, **12** (1987) 634–664
4. Garey, M. R., Johnson, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman (1979)
5. Hanan, M.: On Steiner's problem with rectilinear distance. *SIAM J. Appl. Math.*, **14** (1966) 255–265
6. Leiserson, Ch., Maley, F. M.: Algorithms for routing and testing routability of planar VLSI layouts. *Proceedings of the 17th Symposium on Theory of Computing* (1985) 69–78
7. Maley, F. M.: Compaction with automatic jog introduction. *Proceedings of the 1985 Chapel Hill Conference on VLSI* (1985) 261–284
8. Provan, J. S.: Convexity and the Steiner tree problem. *Networks* **18** (1988) 55–72
9. Richards, D. S., Salowe, J. S.: A linear-time algorithm to construct a rectilinear Steiner tree for k-extremal points. *Algorithmica*, **7** (1992) 246-276

This article was processed using the LaTeX macro package with LLNCS style