FAST HEURISTIC ALGORITHMS FOR RECTILINEAR STEINER TREES

Dana Richards

Computer Science Report No. TR-86-28
November 25, 1986

Fast Heuristic Algorithms for Rectilinear Steiner Trees

Dana Richards

Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, VA 22903

ABSTRACT

A fundamental problem in circuit design is how to connect n points in the plane, to make them electrically common using the least amount of wire. The tree formed, a Steiner tree, is usually constructed with respect to the rectilinear metric. The problem is known to be NP-complete; an extensive review of proposed heuristics is given. An early algorithm by Hanan is shown to have an $O(n \log n)$ time implementation using computational geometry techniques. The algorithm can be modified to do sequential searching in $O(n^2)$ total time. However it shown that the latter approach runs in $O(n^{3/2})$ expected time, for n points selected from an $m \times m$ grid. Empirical results are presented for problems up to 10000 points.

Keywords: Steiner trees, rectilinear metric, heuristic algorithms, computational geometry, average case analysis, VLSI design

Fast Heuristic Algorithms for Rectilinear Steiner Trees

Dana Richards

Department of Computer Science
University of Virginia

1. Introduction

A fundamental problem in circuit design is how to connect n points in the plane, to make them electrically common. The objective studied here is to use the least amount of wire in forming the connection. For a variety of technological and engineering reasons the segments of such a connection are typically horizontal and vertical. The effect is that the distances are measured with respect to the rectilinear metric. In particular the distance between two points u and v is *distance* $(u, v) = |x_u - x_v| + |y_u - y_v|$. This paper reviews various approaches to this problem and concentrates on the earliest known algorithm. An efficient implementation of that algorithm is presented and some results about the expected time complexity are established.

The simplest solution to the problem is to find the rectinear minimum spanning tree (RMST) of the n given points. If the problem is viewed as an instance of a complete graph with rectilinear distance edge weights then any "classical" minimum spanning tree (MST) algorithm can be used. For example, Prim's algorithm can be implemented in $O(n^2)$ time [TARJ83] which is optimal for a complete graph. In an RMST two points can be connected by an infinite number of shortest rectilinear wires. Even if connecting wires are restricted to have at most one bend there can still be two ways to place each wire. Pairs of wires can have segments overlap and the up to 2^{n-1} possible placements of the $n-1$ wires may contain varying amounts of overlap the circuit designer must remove.

Steiner trees provide a conceptually simpler starting point. A Steiner tree spans the n points but may contain additional points in the plane as vertices to provide additional internal branching. *Steiner tree* will refer to a tree of minimum total edge length or, depending on the context, an approximation to such a minimal tree. In particular this paper will discuss rectilinear Steiner trees (RST). For example an embedding of a RMST with overlap removed effectively produces an (approximate) RST, though not necessarily a minimal tree. There is no efficient procedure known for computing a minimal RST. The n points will be called *terminals* and the additional vertices are *Steiner points*.

The Steiner problem for the normal Euclidean metric (L_2) has a long history [GILB68] and many difficult results have been obtained. However these results have not suggested many algorithmic techniques and few reasonable exponential time algorithms even exist, e.g. [COCK86] and [HWAN86a]. The results are tied closely to the geometry and do not extend to the rectilinear metric. It is not surprising therefore that the RST problem is more closely linked to the Steiner problem on graphs, defined in the next section, than to the Euclidean case.

The principal result of this paper is to show that a classical RST heuristic can be implemented efficiently. This is of some importance since the algorithm is used in working systems [DUNL86]. A rather complete review of the literature is given and improvements to other algorithms are suggested as well.

2. Previous Results

The first investigation of the RST problem was by Hanan [HANA66]. He established the following important "dimension reduction" result. Let S be the set of n terminals in the plane. Extend horizontal and vertical lines through each of these points. Define the graph $G_S(V, E)$ by letting V be the set of intersections of these lines and there is an edge between two vertices if they are directly connected by a line, horizontally or vertically. An example appears in Figure 1. Hanan showed that a RST is contained in G_S, i.e. the segments of the tree are composed of edges of G_S.

The Steiner problem for graphs (GST) asks for the minimum weight subgraph that spans S, where $S \subseteq V$ for the graph $G(V, E)$. (The edge weights are assumed to be positive so the subgraph is a tree.) If $|S| = 2$ then this becomes a shortest-path calculation and if $S = V$ then the answer is a minimum spanning tree. Due to Hanan's theorem it follows that any algorithm for the GST problem can be used to solve the RST problem. Note that $|V| = N$ is possibly n^2 so that the problem size is larger, where $|S| = n$ as before. Let $|E| = M$.

The literature is reviewed below beginning with what is known about RST's. After that algorithms previously used for solving both the GST and the RST problem are presented. Many of these algorithms are based on two classical MST algorithms [GRAH85] which are mentioned for completeness. Prim's algorithm builds the MST one node at a time starting with an arbitrary node. At each stage the unused vertex nearest to the current tree is appended to the tree. Kruskal's algorithm starts with a forest of n singleton trees and at each step links the two nearest trees.

Hwang [HWAN76] showed for any set S that

$$\frac{\text{RMST}(S)}{\text{RST}(S)} \leqslant \frac{3}{2}$$

where $\text{RMST}(S)$ and $\text{RST}(S)$ are defined to be the length of the corresponding optimal trees. Let $\text{RMST}(n)$ be the random variable for the length of the RMST for n points drawn uniformly from the unit square. Gilbert [GILB65] established that $E[\text{RMST}(n)] = O(\sqrt{n})$, from which a similar bound follows for RST's. Chung and Graham [CHUN81] give a sharper result;

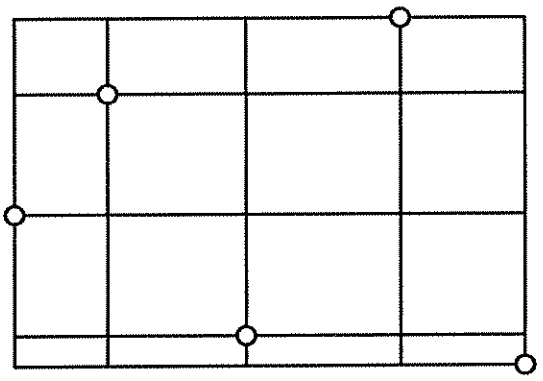

Figure 1

$$\sqrt{n} + O(1) \leqslant \mathrm{LST}(n) \leqslant \sqrt{n} + 1 + o(1)$$

where $\mathrm{LST}(n)$ is the length of the longest RST on any n points drawn from the unit square. Komlos and Shing [KOML23] showed that

$$\mathrm{RST}(n) \geqslant \frac{\sqrt{n}}{5}$$

with probability $1 - o(1)$ as n increases, where $\mathrm{RST}(n)$ is the random variable for the length of the RST for n points drawn uniformly from the unit square. While a RMST can serve as a good approximation to the RST, Chung and Hwang [CHUN79] showed that the semiperimeter of the least bounding rectangle, an often-used approximation, can be worse by a factor about $\frac{1}{2}\sqrt{n}$. (Such an approximation is optimal if $n = 3$ [HANA66].)

The complexity of nearly every Steiner problem is the same; they are all NP-hard (see [GARE79]). Karp showed the GST problem was hard and it remains so even if all edges have unit weight or the graph is planar (as ours is). Both the Euclidean and the rectilinear Steiner tree problems are hard, as well as the discretized (rounding up) Euclidean version. All but the Euclidean problem are known to be in NP. Clearly the bulk of applied research will be on heuristics.

Several exact, and exponential time, algorithms have been proposed for the GST problem. These can be divided into two types. The first type uses mathematical programming techniques, such as Lagrangian relaxation. Beasley [BEAS84] has demonstrated good performance and presents a good survey discussion. The second type uses combinatorial techniques such as dynamic programming. When n is small relative to N, as in this paper, the best algorithm is in an overlooked paper by Levin [LEVI71]. It runs in $O(3^n N + 2^n N^2)$ time but can be improved to $O(3^n N + 2^n (N \log N + M))$ time.

Approximation algorithms for the GST problem are based on Prim's or Kruskal's approaches. Takahashi and Matsuyama [TAKA80] use a Prim-based algorithm that connects nearest unused point from S to the current tree; the connection is a path when a direct connection does not exist. It runs in $O(n N^2)$ time. It can be revised [RAYW86] to include a postprocessing step to find a MST of the chosen subgraph and then prune away any useless vertices. Kou, Markowsky, and Berman [KOU81] use an unspecified MST subroutine. If their algorithm uses Prim's routine it behaves like a restricted version of the above revised algorithm. If it uses Kruskal's algorithm Wu, Widmayer, and Wong [WU86] show that it has an efficient $O(M \log N)$ implementation. All these algorithms produce solutions that are within $2(1 - 1/n)$ of optimal. A more complex Kruskal-based scheme is given by Rayward-Smith [RAYW83].

For the RST problem itself only one exact algorithm has been proposed [YANG72a]. It is a branch-and-bound algorithm and appears to be applicable only for $n < 10$. However they also proposed a suboptimal branch-and-bound algorithm which, while still exponential time, appears to be applicable for $n < 30$. When compared with known exact results the answers were remarkably close to optimal [YANG72b]. It simply uses Prim's algorithm but instead of choosing one of the two possible one-bend orientations of the new wire it explores both.

The earliest polynomial time approximation algorithm is due to Hanan and is discussed at length in the next section. Fu [FU67] gave an unanalyzed manual technique to iteratively improve a spanning tree by creating and breaking cycles. Hanan [HANA72] disproves Fu's claim of optimality. An $O(n^4)$ time solution that uses several ad hoc stages was proposed [SMIT79]. It began by selecting a linear-sized subset of the n^2 vertices of G_S as candidates for Steiner points.

Several heuristics begin with a RMST. Since the underlying complete graph has $\Omega(n^2)$ edges to improve on the $O(n^2)$ time bound requires preprocessing to reduce

the size of the underlying graph. The rectilinear Voronoi diagram (defined analogously to the Euclidean case) has a Delauney triangulation which contains enough edges to find the RMST. Since the triangulation corresponds to a planar graph the RMST can be found in $O(n)$ additional time [TARJ83]. Hwang [HWAN79a] shows, with standard divide-and-conquer techniques, how to find the Delauney triangulation in $O(n \log n)$ time . It should be noted that the details for the rectilinear Voronoi diagram are more complicated than for the Euclidean case and the overall scheme may daunt the typical staff of a circuit designer.

Hwang gave a heuristic for the RST problem, using the above RMST solution, that was based on earlier work by Lee, Bose, and Hwang [LEE76]. That work built a RST in Prim fashion. It used a "3-point connection scheme" instead of simply connecting the nearest unused vertex. This involved a constant time search around the intended 2-point connection for three points which could be connected in a Steiner fashion, perhaps introducing a new Steiner point. It ran in $O(n^2)$ time with most of the time spent on deciding which point to connect next. Hwang [HWAN79b] proposed an $O(n \log n)$ implementation that began with a RMST and from that inferred an ordering of the vertices which the above algorithm could use to decide which node to connect next and where to try to connect it.

Smith, Lee, and Liebman [SMIT80] proposed an $O(n \log n)$ time approach based on iteratively improving the RMST found over the Delauney triangulation. The technique is complex and ad hoc.

Bern and de Carvalho [BERN85] investigated Kruskal-based approaches attributed to Thompson. Variations attributed to Ng and the themselves were proposed that were supposed to be faster than, but usually inferior to, the original. Thompson began with n singleton trees and at each stage a new wire connects two trees and the shortest such wire is chosen. This wire may not necessarily connect terminals but it can be "slid" so that it at least has one endpoint at a terminal, Steiner point, or corner of a previous wire. There are only $O(n)$ such positions. They assume the points are grid points of an $m \times m$ grid. They give an unusual analysis; even though m is theoretically unrelated to n (except $m > \sqrt{n}$) they assume a data structure with $O(m^2)$ is permisssable. Thompson's algorithm is implemented in $O(m\, n^2 \log n)$ time and their variation takes $O(m\, n^2)$ time.

Thompson's algorithm can, by a small alteration, be made to run in $O(n^2 \log n)$ time. (The alteration requires that after each new wire is placed each of the $O(n)$ relevant points needs to add at most one entry to the priority queue and that entry can be determined in constant time.) Since the same alteration applied to their variation also yields the same time bounds there is no reason to prefer the variation. However the variant algorithm has proven amenable to analysis. Bern [BERN86b] has shown that the expected improvement of the approximate RST relative to the optimal RMST is bounded away from 0 in the limit. The proven bound, 0.00098, is much less than empirical studies indicate it should be. It is also shown that a linear number of Steiner points are expected. The results are for n points drawn at random from the unit square distributed according to a Poisson process with intensity n; a uniform distribution remains to be analyzed.

Two basic paradigms in Computational Geometry are divide-and-conquer and and the line-sweep. Komlos and Shing [KOML23] proposed a divide-and-conquer algorithm based on two-dimensional partitioning. They start with n points, assumed to be uniformly distributed in the unit square, and a parameter t. Next iteratively partition, using medians, the square into small rectangles until each rectangle contains approximately t points. Find the optimal RST for each rectangle and combine these trees to give the final tree, after some clean-up steps. If each optimal subproblem is solved in $f(t)$ steps then the algorithm runs in $O(f(t)n + n \log n)$ time, which is $O(n \log n)$ for $t = O(\log \log n)$ if Levin's algorithm is used. They show that their

approximation is within a factor of $1 + O(1/\sqrt{t})$ of optimal, with probability $1 - o(1)$ as n increases. Another slightly faster algorithm was presented that depends heavily on the unifomity of the point distribution. Hence, even though the same probabilistic bound above holds, this algorithm has a worst-case performance that is at least a factor of t of optimal. No implementation was attempted.

Hanan, in unpublished work [HANA65], suggested an algorithm that today would be called a line-sweep algorithm. It is essentially Prim-based. The rest of this paper discusses improvements to that algorithm. Servit [SERV81] failed to see an efficient implementation of Hanan's algorithm and proposed a crude approximation to it which runs in $O(n \log n)$ time. Our implementation of Hanan's algorithm runs in $O(n \log n)$ time, removing the need for an alternative.

There are some instances of the RST for which there are polynomial time algorithms. If the terminals lie on the boundary of a $p \times q$ grid, corresponding to G_S, then an $O(p^2q + q^2p)$ time dynamic programming algorithm exists [AHO77], which has recently been improved to $O(p + q)$ time [AGAR86]. Bern [BERN86a] has generalized this by considering the GST problem on a planar graph with all the points on f faces of an embedding. He gives an $O(N^{2f+2})$ time algorithm which yields an $O(n^5)$ time algorithm for the RST problem where each terminal is maximal, i.e. each has at least one empty open quadrant. Wald and Colburn [WALD82] give a linear time algorithm when the graph is outerplanar.

3. An Implementation of Hanan's Algorithm

Hanan's algorithm begins by sorting the n points in ascending order of the y-coordinates. By processing the points in this order it simulates the effect of sweeping a horizontal bar over the plane, bottom to top, and processing each point as it is covered by the bar. While it is not important for correctness, if points with equal y-coordinates were sorted by their x-coordinate it would improve the efficiency somewhat.

The algorithm begins with the singleton tree composed of a point with least y-coordinate. As each new point is processed it is connected to the current tree by the wire of shortest length. To remove bias due to the bottom-to-top directedness, the point arrangement is rotated 90° three times, and each time a new appoximate RST is constructed. The best of the four trees is chosen.

An implementation was not suggested by Hanan. The chief difficulty lies in the fact a new wire may connect to the middle of a previous wire. However there are only $O(n)$ wires and the shortest connection a given wire can be computed in constant time. So each new point can inspect the set if $O(n)$ terminals, Steiner points, corners, and wires of the current tree and determine the shortest wire in linear time. This leads to an $O(n^2)$ time implementation; a fact that was observed before without explanation [HWAN78, HWAN79b, SERV81].

The key observation in the implementation is that lower parts of the tree become effectively hidden and can be forgotten. A point a is said to cover point b if b is in the 90° cone below a. Formally, a *covers* b if $y_b < y_a$, $x_a + y_a \geqslant x_b + y_b$, and $x_a - y_a \leqslant x_b - y_b$.

Lemma 1: Let a and b be two points anywhere in the current tree when c is processed. If a covers b then $\mathit{distance}(c,a) \leqslant \mathit{distance}(c,b)$.

Proof: A simple proof can be constructed by case analysis. For example, if $x_c \leqslant x_b \leqslant x_a$ then $\mathit{distance}(c,a) = x_a - x_c + y_c - y_a$ and $\mathit{distance}(c,b) = x_b - x_c + y_c - y_b$ and the relation holds. A geometric proof of the result could be given. The observation needed is that the "circle" of radius $\mathit{distance}(c,a)$ about c (drawn as a diamond in the plane) does not properly contain any points of the cone below a. □

Lemma 2: Let a, b, and c be as in Lemma 1 such that b does not cover a. If $x_c \geqslant x_a \geqslant x_b$ or $x_b \geqslant x_a \geqslant x_c$ then *distance* $(c,a) \leqslant$ *distance* (c,b).

Proof: Straightforward algebra suffices. □

Lemmas 1 and 2 are illustrated in Figures 2(a) and 2(b) respectively. The implication of Lemma 1 is that each horizontal segment and the tip of each uncovered vertical segment generates a cone and the relevent portion of the tree is the set of points not covered by other points. This is shown in Figure 3.

When the covered portion of the tree is removed what remains is a set of horizontal segments and single points; these single points will be regarded as degenerate

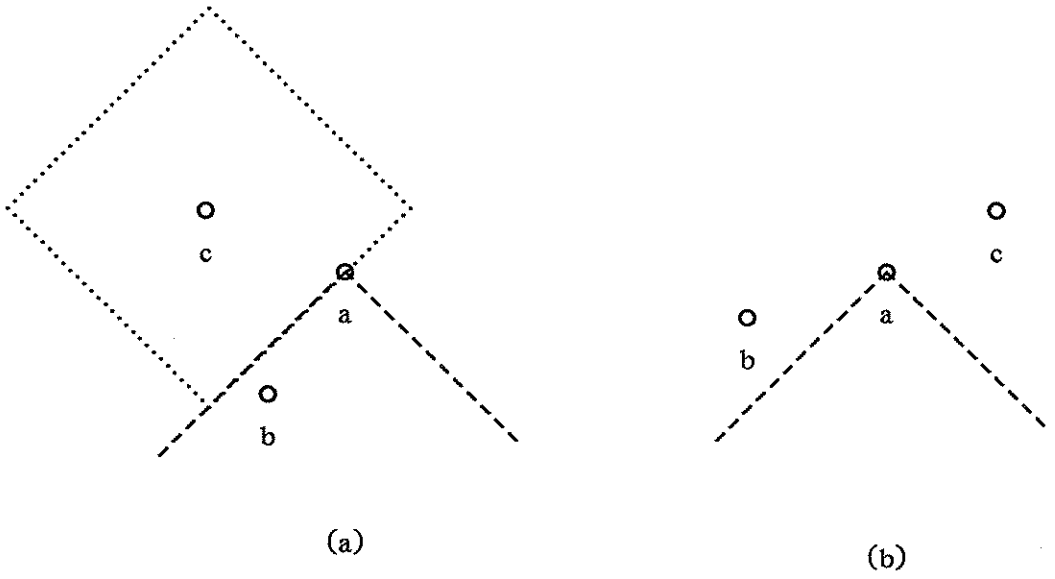

Figure 2

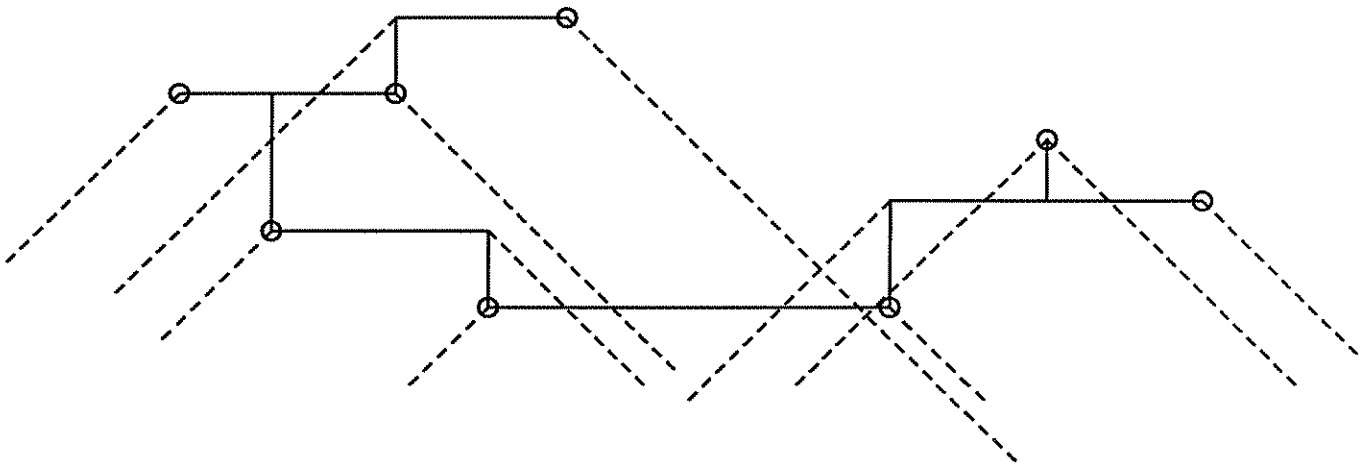

Figure 3

horizontal segments in the sequel. In Figure 4 this is shown with vertical lines through some endpoints to illustrate how the set of left endpoints partitions the set of x-coordinates.

Our algorithm is shown in Figure 5, for one of the four identical passes. The important data structure, *SEGMENTS*, is manipulated by the routines *pred_succ* and *addsegment*. It maintains for the current tree the set of horizontal segments that are not covered and the segments are ordered by their left endpoints. When a new point p is processed *pred_succ(p)* returns the two segments, a and b, adjacent in the order such that p is to the right of the left endpoint of b (or above b) and p is to the left of the left endpoint of a. Lemma 2 implies the only points that are candidates for being the nearest point to p are the points of a or the left endpoint of b.

While p is regarded as a point initially, in the loop it is thought of as a degenerate horizontal segment. The statement "extend p to the left end of a" then makes sense; see Figure 6. The actual code contains additional tests to ignore 0 length connections. Clearly, at the end of the loop, p should now be added to *SEGMENTS* since it is an uncovered segment of the new tree. However p may cover all or portions of other segments still in *SEGMENTS*. The routine *addsegment(p)* is responsible for the update. It is convenient to note that all the removed segments form a contiguous block in the order and at most two segments are shortened, one at either end of that block. This is illustrated in Figure 7. Using the assumption that all points have integer coordinates then what are open endpoints in *SEGMENTS*, such as the right endpoint of c in Figure 7, can be pruned back to the next grid point. This removes several equality tests and simplifies the code.

When implementated in the C language the program involves half a page of code plus the code for *initialize*, *pred_succ*, and *addsegment*. *Initialize* sorts the data and processes the first point. *SEGMENTS* is created with this point and two additional points at $-\infty$ and ∞; this ensures that *pred_succ* is well-defined. *SEGMENTS* can be implemented in two obvious ways. The first is to use a linear linked list with the elements maintained in order. It makes sense to use a "roving" bidirectional pointer rather than beginning at the head of the list each time. This is because *addsegment* operates in the same vicinity that *pred_succ* left off.

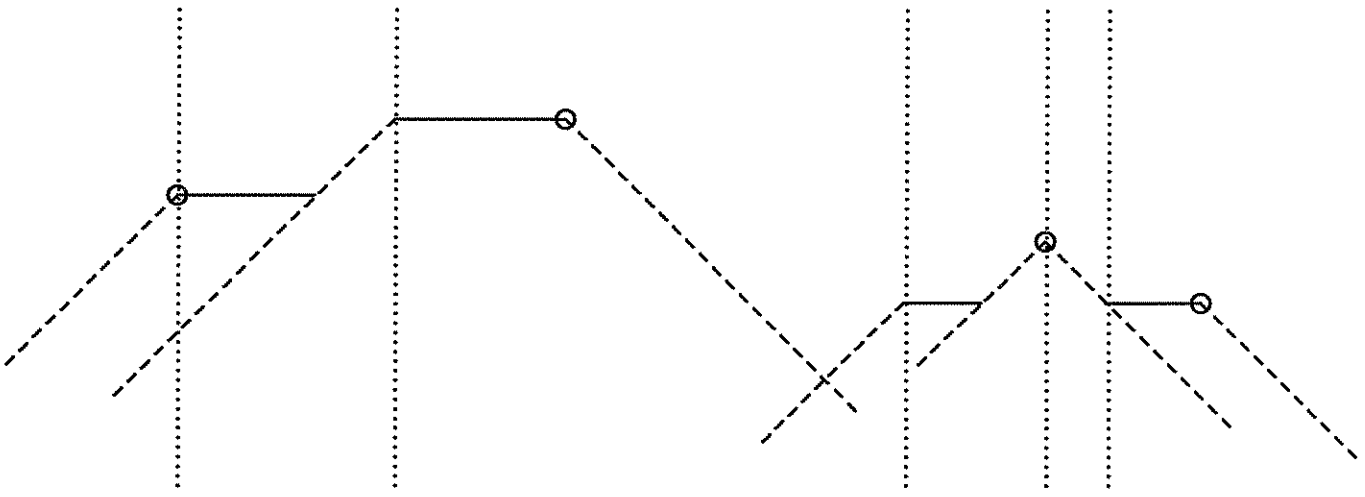

Figure 4

```
intialize;
while more points do
    p ← get_next_point
    (a,b) ← pred_succ (p)
    if p closer to segment a then
        if p is to the right of a then
            extend p to the left end of a;
            connect to a
        else
            connect straight down to a
        endif
    else
        extend p to the right end of b;
        connect to b
    endif
    addsegment (p)
endwhile
```

Figure 5

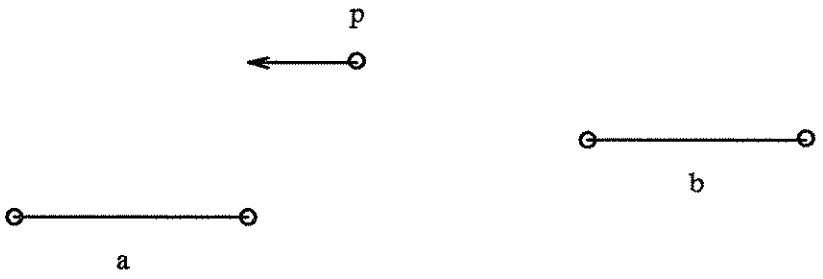

Figure 6

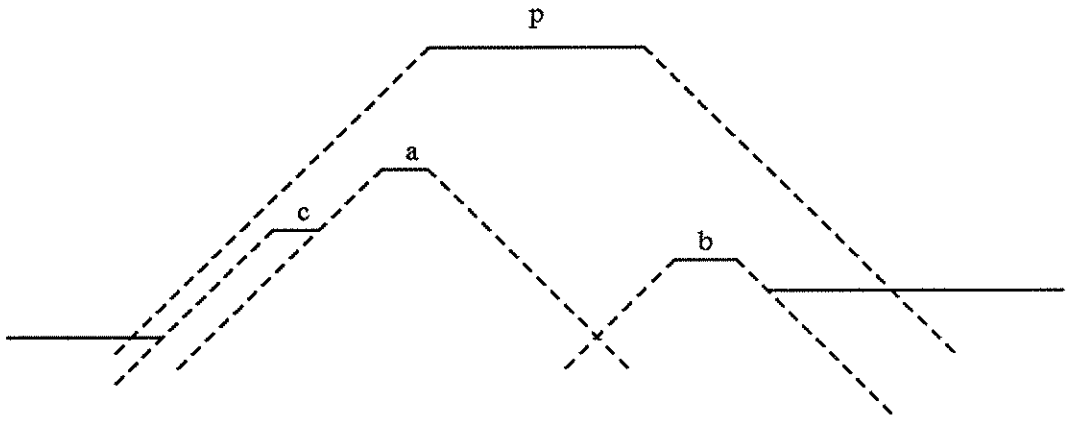

Figure 7

The second alternative for *SEGMENTS* is some balanced binary search tree. Splay trees [TARJ83] were selected since they have, in an amortized analysis, excellent performance and allow the simple implementations described below. In a splay tree it is a basic operation, a "splay", to split the keys into two sets separated by a value so that the first set is the left subtree of the root and the second set is the right subtree. This gives a natural one splay approach for *pred_succ*. By a slight modification it can split with respect to the left or the right edge of the cone below p. Hence using two such splays gives a simple implementation of *addsegment*.

4. Analysis of the Algorithm

It is clear that the initialization requires $O(n \log n)$ time for the sorting step. If each new point can be processed in $O(\log n)$ then the entire algorithm runs in $O(n \log n)$ time. Using splay trees each splay is done in $O(\log n)$ amortized time, hence both *pred_succ* and *addsegment* have the same bound. Since there is only constant additional work for each new point the overall bound follows.

If instead the linear list version is used then there can be $O(n)$ worst-case time per point leading to $O(n^2)$ overall performance. There are inputs which achieve these bounds. However in practice the number of elements in *SEGMENTS* after processing t points, N_t, would not be as large n.

To investigate N_t it is assumed that the n points are chosen at random from the vertices of an $m \times m$ grid. Consider S_t the set of the t lowest points. Let D_t be the number of points in S_t that are not covered by other points in S_t. Clearly $N_t \leqslant D_t$ since the algorithm will "extend" some points, potentially covering even more points.

Let $D_t(n,m) = E[D_t]$ where the expectation assumes all selections of n points are equally likely. To see that $D_t(n,m) < D_{t+1}(n,m)$ note that the t highest points could have been selected instead, and as a new lower point is added it can only add to D_t. Hence for upper bounds only $D(n,m) = D_n(n,m)$ needs to be considered. Let the rows of the grid be numbered $1, 2, \cdots$ from the top down. Let p_{ij} be the probability that grid point (i,j) is chosen and is not covered by a chosen grid point. Let $p = n/m^2$ be the probability a grid point is chosen and $q = 1-p$. Let A_{ij} be the set of grid points that could cover (i,j) and a_{ij} is the number of points in A_{ij}. It follows that

$$p_{ij} = p\, q^{a_{ij}}.$$

Note $a_{1j} = 0$ and $a_{ij} \geqslant i^2/2$ for $i > 1$.

Theorem: $D(n,m) \leqslant B(n,m)$ where

$$B(n,m) = \frac{n}{m}(1 + \sqrt{\frac{\pi}{2}}(-\ln(1 - \frac{n}{m^2}))^{-1/2})$$

Proof:

$$D(n,m) = \sum_{i=1}^{m}\sum_{j=1}^{m} p_{ij}$$

$$\leqslant m\,p + \sum_{i=2}^{m}\sum_{j=1}^{m} p\,q^{i^2/2}$$

$$\leqslant m\,p + m\,p \sum_{i=1}^{\infty} q^{i^2/2}$$

$$\leqslant m\,p\,(1 + \int_0^{\infty} q^{i^2/2} di)$$

$$= m\,p\,(1 + \int_0^{\infty} e^{i^2 \ln q\,/2} di)$$

$$= m\,p\left(1 + \sqrt{\frac{\pi}{-2\ln q}}\right)$$

where the last step uses the identity ([PURD85], p. 97)

$$\int_0^{\infty} e^{-t} t^{-1/2} dt = \Gamma(\frac{1}{2}) = \sqrt{\pi}$$

and the substitution $t = -i^2 \ln q\,/2$ $(dt = -\ln q\; i\; di)$. □

Corollary: $D(n,m) = O(\sqrt{n})$.

Proof: The stronger result that $0.88\sqrt{n} \leqslant B(n,m) \leqslant 2.26\sqrt{n}$, for $m \geqslant \sqrt{n}$ is established. The first term of $B(n,m)$, i.e. n/m, decreases, as m increases, from $\sqrt{n}$ to 0. The second term increases from 0 to $\sqrt{\pi n/2}$. To see this notice the second term is

$$\frac{\sqrt{\pi/2}\,n}{\sqrt{-\ln(1 - \frac{n}{m^2})^{m^2}}}$$

and as m increases $(1 - n/m^2)^{m^2}$ goes to e^{-n}. To get the upper bound just add the coefficients. (It is conjectured this coefficient could be lowered to 1.80.)

To get the lower bound note that the two terms cross when $m = \sqrt{n/c}$ where $c = 1 - e^{-\pi/2}$. This gives a lower bound coefficient of $1/\sqrt{c} < 0.88$. A careful analysis can raise this coefficient to 1. □

The corollary does not necessarily represent a tight bound, though it is conjectured that $D(n,m) = \Omega(\sqrt{n})$, but it does confirm a hypothesis that was formed after empirical testing. In Table 1 the observed values of N_n (averaged over 5 runs) is compared with the theoretical upper bound $B(n,m)$. (The final column of the table is discussed later.) Note that for fixed n, as m increases $B(n,m)$ becomes a better bound for the size of the data structure *SEGMENTS*.

It follows that when using the simple list version algorithm should run in $O(n^{3/2})$ expected time, since each subroutine call should execute in expected $O(\sqrt{n})$

(n,m)	$B(n,m)$	N_n observed	% improvement
100,40	14.83	7.25	4.02
100,100	13.50	8.05	4.99
100,500	12.73	8.37	5.93
10000,400	148.33	70.89	3.99
10000,10000	126.32	84.67	3.95

Table 1

time. Even though careful attention was paid to the splay tree code it is still the case that for moderate values of n the linear list version actually runs faster than the splay tree version. For example, for $n = 10000$ the list version takes 56.7 seconds and the tree version takes 59.0 seconds.

5. Empirical Comparisons

The time necessary to solve problems with $n < 100$ was less than a second and even for $n = 10000$ less than a minute was required (for both versions). These times are somewhat better than the times reported for other $O(n \log n)$ algorithms. No data is even available on test cases in the literature for $n > 400$. (For comparison, the $O(n^2)$ time RMST algorithm that was used took 5220 seconds for $n = 10000$.) However, as explained below, Hanan's algorithm does not produce superior trees.

Our experiments were only done on random data; n points selected from an $m \times m$ grid. These kinds of experiments, with larger and larger n, tend to overlook the fact that in practice n is small. For example Hu and Shing [HU85] state "in circuit routing, the average number of pins is between 2.5 and 3.5." Hanan's algorithm is known to produce good results for small n, e.g. it generates an optimal tree for $n < 4$ [HANA66]. It can be argued that for small n the overhead of constructing a Voronoi diagram is excessive.

Servit [SERV81] tested Hanan's algorithm on two printed circuit board problems. His "typical digital" board had 109 two-point nets, 55 three-point nets, and 40 larger nets. The average number of terminals per net was 3.94 and the maximum was 112. For all but the six largest nets the optimal RST's were determined. The average relative error, i.e. (APPROX - OPTIMAL) / OPTIMAL, for the entire board was 0.05%. This excellent result is due to the large number of smaller nets.

However efficient algorithms are needed for many problems, such as the layout of the power lines, ground lines, and various buses. These often are the critical cases. It should also be remarked that some decisions in the design process are made depending on the length of the tree alone. The tree itself is not needed, instead a good estimate of its length is sufficient.

Many runs were made with random data. Since for large n it is intractable to get the optimal result the standard convention of computing the relative improvement of the obtained approximate solution to the RMST was used, i.e. informally (RMST - APPROX) / RMST. Some of the results appear in the final column of Table 1. The improvement was between 3% and 7% in nearly all cases with an overall average improvement of around 4%. The improvement tends to increase for smaller n. In Table 2 the results are compared with those performances found in the literature. The largest reported problem successfully attempted in each case is given with the corresponding running time reported (admittedly on different computers). Bern [BERN86b] guessed that, while the best improvement is 33%, the average

is about 12%.

To understand why Hanan's algorithm performs as it does one needs only to look at the tree it produces, e.g. Figure 8. The initial stages tend to establish a horizontal "ground" from which disjoint subtrees grow up. If these subtrees grow

Investigators	Problem Size	Time (secs)	% Improvement
Yang & Wing	35	185	11
Smith & Liebman	40	34	7
Smith, Lee & Liebman	40	1	8
Lee, Bose & Hwang	35	5	9
Hwang	-	-	9
Bern & de Carvalho	40	1	9
Hanan (new)	10000	56	4

Table 2

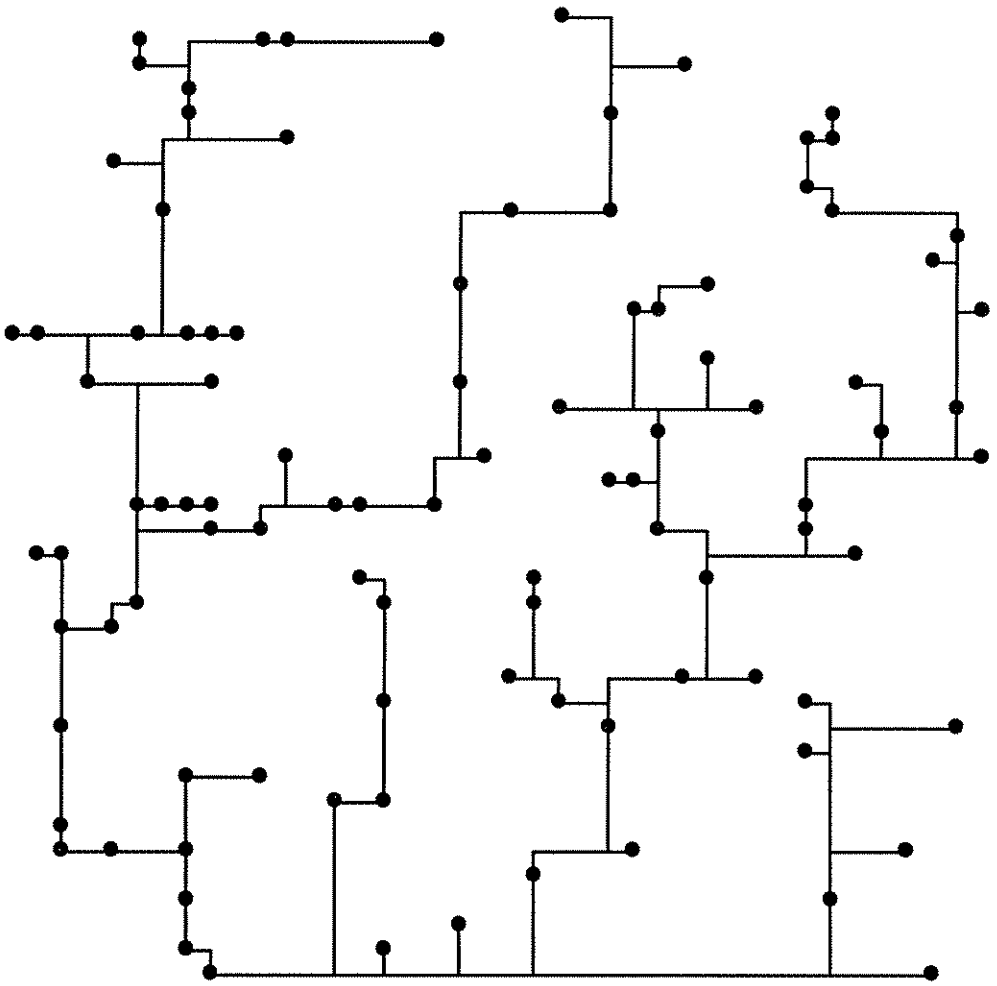

Figure 8

near to each other but are far apart at their roots the potential savings of a later connection will be overlooked. Unlike algorithms that start with a RMST, Hanan's algorithm can produce a tree worse than the RMST. In fact examples can be constructed (e.g. with the points in a large "X" formation) where the ratio of the length of the approximation to the length of the RMST approaches 3/2. Occasionally a tree that was marginally longer than the RMST was produced during the experiments. However in no case was the best of trees inferior to the RMST.

A postprocessing step could trace adjacent contours of two of the subtrees growing up from the ground. These tracings could find shorter connections that are overlooked by the basic algorithm. A search of all adjacent contours should be possible in $O(n)$ time, given the tree. However the details of this have not been investigated; several approaches seem worth trying.

6. Conclusions

An efficient new implementation of an old heuristic for the RST problem has been presented. While its $O(n \log n)$ time complexity has been equalled before, it is conceptually simpler and therefore easier to maintain. An even simpler $O(n^{3/2})$ expected time version is actually faster on moderately large problems.

There are several open problems. There is no good theorem for the expected length of a RST relative to the RMST. For Hanan's algorithm there is no known tight upper bound on its worst-case performance relative to the RMST. The postprocessing step, mentioned in the previous section, needs to be explored.

It has been observed empirically [BERN85] that if a RMST is embedded, by choosing one of the two possible one-corner wires for non-trivial edges, and then the resultant "doubling" of edges is subtracted out, we get a 5% improvement over the original RMST. This suggests the following heuristic which is simpler than the similar approach of Smith, Lee, and Liebman [SMIT80]. Begin with a RMST and represent each (unembedded) edge by straight lines in the plane. This straight line MST is planar. For each pair of cyclically adjacent edges around a node compute the maximum overlap, i.e. doubling, which an embedding of these two edges could have. Enter these $O(n)$ quantities into a priority queue. Remove the maximum overlapping pair of edges from the queue and introduce a Steiner point as shown in Figure 9. The four or fewer entries in the queue using the edge (a,b) are updated using (a,d) or (b,d); (b,c) is similarly updated. Iterate until no overlap exists.

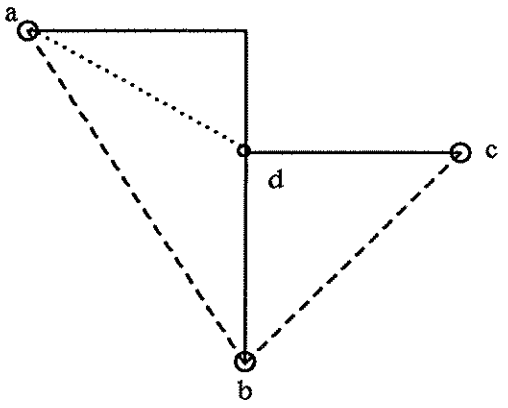

Figure 9

This can easily be achieved in $O(n \log n)$ time given a RMST. Unfortunately an overall $O(n \log n)$ bound is achievd only by using the Delauney triangulation.

A Steiner problem closely related to the rectilinear case involves terminals which are identified with equal length strings. The distance between two terminals is the number of corresponding characters that differ in their strings. The least spanning tree is called the Steiner tree in phylogeny since these can be regarded as gene sequences. [FOUL83, SHOR82]. The Steiner points introduced could be regarded as "missing links". This is known to be NP-complete [FOUL82] even for binary strings. It is an open question which, if any, of the heuristics discussed here are effective for this problem.

Circuit designers have considered other metrics that generalize the rectlinear geometry with its two orthogonal axes. For example four axes separated by 45° have been used, e.g. [WHIT86]. The hexagonal system, with three equally spaced axes, has been proposed; recent work by Hwang and Weng [HWAN86b] suggests new approaches for this case. How to solve the Steiner tree problem for these metrics remains an open problem.

7. References

[AGAR86] P. K. Agarwal and M. T. Shing, Algorithms for the Special Cases of Rectilinear Steiner Trees: I. Points on the Boundary of a Rectilinear Rectangle, Tech. Report TRCS86-17, Univ. of California at Santa Barbara, 1986.

[AHO77] A. V. Aho, M. R. Garey and F. K. Hwang, Rectilinear Steiner Trees: Efficient Special-Case Algorithms, *Networks*, 7, 1977, pp. 37-58.

[BEAS84] J. E. Beasley, An Algorithm for the Steiner Problem in Graphs, *Networks*, **14**, 1984, pp. 147-159.

[BERN85] M. W. Bern and M. de Carvalho, A Greedy Heuristic for the Rectilinear Steiner Tree Problem, Tech. Report, Computer Science Division, Univ. of California at Berkeley, 1985.

[BERN86a] M. W. Bern, A More General Special Case of the Steiner Tree Problem, Tech. Report, Computer Science Division, Univ. of California at Berkeley, 1986.

[BERN86b] M. W. Bern, Two Probabilistic Results on Rectilinear Steiner Trees, *Proc. 18th Annual ACM Symp on Theory of Computing*, 1986, pp. 433-441. , [Corrected version available.].

[CHUN79] F. R. K. Chung and F. K. Hwang, The Largest Minimal Rectilinear Steiner Trees for a Set of n Points Enclosed in a Rectangle with Given Perimeter, *Networks*, **9**, 1979, pp. 19-36.

[CHUN81] F. R. K. Chung and R. L. Graham, On Steiner Trees for Bounded Point Sets, *Geometriae Dedicata*, **11**, 1981, pp. 353-361.

[COCK86] E. J. Cockayne and E. E. Hewgill, Exact Computation on Steiner Minimal Trees in the Plane, *Information Processing Letters*, 22, 1986, pp. 151-156.

[DUNL86] A. E. Dunlop, personal communication., 1986.

[FOUL82] L. R. Foulds and R. L. Graham, The Steiner Problem in Phylogeny is NP-Complete, *Advances in Applied Math.*, **3**, 1982, pp. 43-49.

[FOUL83] L. R. Foulds and V. J. Rayward-Smith, Steiner Problems in Graphs: Algorithms and Applications, *Engineering Optimization*, 7, 1983, pp. 7-16.

[FU67] Y. Fu, Application of Linear Graph Theory to Printed Circuits, *Proc. Asilomar Conf. Systems and Circuits*, 1967, pp. 721-728.

[GARE79] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, 1979.

[GILB65] E. N. Gilbert, Random Minimal Trees, *Journal SIAM*, **13**, 1965, pp. 376-387.

[GILB68] E. N. Gilbert and H. O. Pollack, Steiner Minimal Trees, *SIAM Journal Applied Math.*, **16**, 1968, pp. 1-29.

[GRAH85] R. L. Graham and P. Hell, On the History of the Minimum Spanning Tree Problem, *Annals of the History of Computing*, **7**, 1985, pp. 43-57.

[HANA65] M. Hanan, Net Wiring for Large Scale Integrated Circuits, RC 1375, IBM Research Report, 1965.

[HANA66] M. Hanan, On Steiner's Problem with Rectilinear Distance, *Journal SIAM Applied Math.*, **14**, 1966, pp. 255-265.

[HANA72] M. Hanan, A Counterexample to a Theorem of Fu on Steiner's Problem, *IEEE Trans. on Circuit Theory*, **CT-19**, 1972, pp. 74.

[HU85] T. C. Hu and M. T. Shing, A Decomposition Algorithm for Circuit Routing, in *VLSI Circuit Layout: Theory and Design*, T. C. Hu and E. S. Kuh (ed.), IEEE Press, 1985, 144-152.

[HWAN76] F. K. Hwang, On Steiner Minimal Trees with Rectilinear Distance, *SIAM Journal Applied Math.*, **30**, 1976, pp. 104-114.

[HWAN78] F. K. Hwang, The Rectilinear Steiner Problem, *Design Automation & Fault Tolerant Computing*, **3**, 1978, pp. 303-310.

[HWAN79a] F. K. Hwang, An O(n log n) Algorithm for Rectilinear Minimal Spanning Trees, *Journal ACM*, **26**, 1979, pp. 177-182.

[HWAN79b] F. K. Hwang, An O(n log n) Algorithm for Suboptimal Rectilinear Steiner Trees, *IEEE Trans. on Circuits and Systems*, **CAS-26**, 1979, pp. 75-77.

[HWAN86a] F. K. Hwang, A Linear Time Algorithm for Full Steiner Trees, *Operations Research Letters*, **4**, 1986, pp. 235-237.

[HWAN86b] F. K. Hwang and J. F. Weng, Hexagonal Coordinate Systems and Steiner Minimal Trees, *Discrete Math.*, **62**, 1986, pp. 49-57.

[KOML23] J. Komlos and M. T. Shing, Probabilistic Partitioning Algorithms for the Rectilinear Steiner Problem, *Networks*, **15**, 1985, pp. 413-423.

[KOU81] L. Kou, G. Markowsky and L. Berman, A Fast Algorithm for Steiner Trees, *Acta Informatica*, **15**, 1981, pp. 141-145.

[LEE76] J. H. Lee, N. K. Bose and F. K. Hwang, Use of Steiner's Problem in Supoptimal Routing in Rectilinear Metric, *IEEE Trans. on Circuits and Systems*, **CAS-23**, 1976, pp. 470-476.

[LEVI71] A. J. Levin, Algorithm for the Shortest Connection of a Group of Graph Vertices, *Soviet Math. Doklady*, **12**, 1971, pp. 1477-1481.

[PURD85] P. W. Purdom and C. A. Brown, *The Analysis of Algorithms*, Holt, Rinehart and Winston, 1985.

[RAYW83] V. J. Rayward-Smith, The Computation of Nearly Minimal Steiner Trees in Graphs, *Intl. Journal Math. Educ. Sci. Technol.*, **14**, 1983, pp. 15-23.

[RAYW86] V. J. Rayward-Smith and A. Clare, On Finding Steiner Vertices, *Networks*, **16**, 1986, pp. 283-294.

[SERV81] M. Servit, Heuristic Algorithms for Rectilinear Steiner Trees, *Digital Processes*, **7**, 1981, pp. 21-32.

[SHOR82] M. L. Shore, L. R. Foulds and P. B. Gibbons, An Algorithm for the Steiner Problem in Graphs, *Networks*, **12**, 1982, pp. 323-333.

[SMIT79] J. M. Smith and J. S. Liebman, Steiner Trees, Steiner Circuits and the Interference Problem in Building Design, *Engineering Optimization*, **4**, 1979, pp. 15-36.

[SMIT80] J. M. Smith, D. T. Lee and J. S. Liebman, An O(N log N) Heuristic Algorithm for the Rectilinear Steiner Minimal Tree Problem, *Engineering Optimization*, **4**, 1980, pp. 179-192.

[TAKA80] H. Takahashi and A. Matsuyama, An Approximate Solution for the Steiner Problem in Graphs, *Math. Japonica*, **24**, 1980, pp. 573-577.

[TARJ83] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, 1983.

[WALD82] J. A. Wald and C. J. Colbourn, Steiner Trees in Outerplanar Graphs, *Proc. 13th S.E. Conf. on Combinatorics, Graph Theory, and Computing*, 1982, pp. 15-22.

[WHIT86] T. Whitney and C. Mead, An Integer Based Hierarchical Representation for VLSI, in *Advanced Research in VLSI*, C. E. Leiserson (ed.), MIT Press, 1986, 241-257.

[WU86] Y. F. Wu, 6. Widmayer and C. K. Wong, A Faster Approximation Algorithm for the Steiner Problem in Graphs, *Acta Informatica*, **23**, 1986, pp. 223-229.

[YANG72a] Y. Y. Yang and O. Wing, Optimal and Suboptimal Solution Algorithms for the Wiring Problem, *Proc. IEEE Intl. Symp. Circuit Theory*, 1972, pp. 154-158.

[YANG72b] Y. Y. Yang and O. Wing, Supoptimal Algorithm for a Wire Routing Problem, *IEEE Trans. on Circuit Theory*, **CT-19**, 1972, pp. 508-510.