ELSEVIER

Discrete Applied Mathematics 90 (1999) 161–171

DISCRETE APPLIED MATHEMATICS

Computing optimal rectilinear Steiner trees: A survey and experimental evaluation

Joseph L. Ganley *

Cadence Design Systems, Inc., 2615 John Milton Drive, Oak Hill, VA 20171, USA

Received 18 May 1995; received in revised form 13 June 1997; accepted 16 February 1998

Abstract

The rectilinear Steiner tree problem is to find a minimum-length rectilinear interconnection of a set of points in the plane. A reduction from the rectilinear Steiner tree problem to the graph Steiner tree problem allows the use of exact algorithms for the graph Steiner tree problem to solve the rectilinear problem. Furthermore, a number of more direct, geometric algorithms have been devised for computing optimal rectilinear Steiner trees. This paper surveys algorithms for computing optimal rectilinear Steiner trees and presents experimental results comparing nine of them: graph Steiner tree algorithms due to Beasley, Bern, Dreyfus and Wagner, Hakimi, and Shore, Foulds, and Gibbons and geometric algorithms due to Ganley and Cohoon, Salowe and Warme, and Thomborson, Alpern, and Carter.

1. Introduction

The *rectilinear Steiner tree* (*RST*) problem is stated as follows: given a set T of n points called *terminals* in the plane, find a set S of additional points called *Steiner points* such that the length of a rectilinear minimum spanning tree of $T \cup S$ is minimized. Garey and Johnson [11] prove that the RST problem is NP-complete, indicating that a polynomial-time algorithm to compute an optimal RST is unlikely to exist.

However, a number of exponential-time algorithms have been devised for computing optimal RSTs of small instances. Such algorithms are particularly pertinent because in VLSI routing applications, a typical instance often contains few terminals [7].

In such research, performance in practice is clearly critical. However, few researchers have compared their algorithms empirically with other algorithms for computing optimal RSTs. Here we seek to fill this gap in the literature. We present experimental results for many of the best algorithms for computing optimal RSTs [1, 3, 6, 8–10, 12, 21, 22, 24], including both graph-based and geometric algorithms.

* Fax +1 703 476 9422; e-mail: ganley@cadence.com.

0166-218X/99/$ – see front matter
PII: S0166-218X(98)00089-4

2. Graph algorithms

An early result on the RST problem is *Hanan's theorem* [13], which provides a reduction from the RST problem to the *graph Steiner tree* (GST) problem. Hanan proved that for any instance, an optimal RST exists in which every Steiner point lies at the intersection of two orthogonal lines that contain terminals. Hanan's theorem implies that a graph G called the *Hanan grid graph* is guaranteed to contain an optimal RST. G is constructed as follows: draw a horizontal and vertical line through each terminal. The vertices in G correspond to the intersections of the lines. There is an edge between two vertices if they are adjacent along a line, and the weight of an edge is the rectilinear distance between its endpoints.

Henceforth, let n denote the number of terminals and let m denote the number of nonterminal vertices in the Hanan grid graph. Note that $m = O(n^2)$ in the worst case.

2.1. Graph reductions

Often many vertices in G can be deleted, along with their adjacent edges, while retaining the guarantee that G contains an optimal RST. Many reduction techniques for general graphs have been devised (see [15]) and have been shown to be quite effective in some types of graphs. However, unfortunately, most of these reductions are ineffective when applied to the Hanan grid graph [26].

Other reductions have been devised that are specific to the Hanan grid graph. One such reduction is the *convex-hull reduction* of Yang and Wing [28]. Any nonterminal vertex that is adjacent to exactly two orthogonal edges e_1 and e_2 can be deleted if the other two edges forming a rectangle with e_1 and e_2 are present. The vertices remaining after this reduction has been performed are precisely those that lie within the rectilinear convex hull of the terminals [19]. In pathological cases the convex-hull reduction may have no effect, but for small, randomly generated sets of terminals it is typically quite effective.

The convex-hull reduction often leaves many terminals of degree 1. Such terminals can be deleted (along with their adjacent edge) and their neighbor made a terminal, and the appropriate edge added back into the final solution. We call this the *terminal reduction*. The most striking effect of the terminal reduction is not the nonterminals it removes, but rather the fact that often two or more terminals collapse into a single new terminal.

Fig. 1 illustrates the Hanan grid graph, the graph remaining after the convex-hull reduction, and the graph remaining after the terminal reduction.

2.2. Algorithms

Hakimi [12] presents one of the first algorithms for the GST problem, called the *spanning tree enumeration* algorithm. Hakimi's algorithm considers every subset S of $n-2$ or fewer nonterminals, computing an MST of $T \cup S$ for each and taking

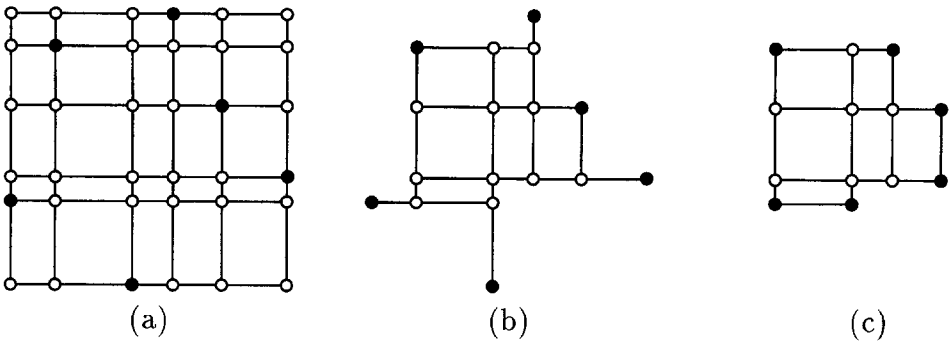

Fig. 1. (a) The Hanan grid graph, (b) the convex-hull reduction, and (c) the terminal reduction.

the minimum to be the optimal Steiner tree. The time complexity of the algorithm is $O(n^2 2^m)$.

Dreyfus and Wagner [6] present a dynamic programming algorithm for the GST problem. The algorithm is based on an elegant decomposition theorem that states that an optimal Steiner tree of a set S of terminals can be decomposed into three subsets A, B, and $\{v\}$ such that for some nonterminal vertex u, the union of the optimal Steiner trees of $A \cup \{u\}$ and $B \cup \{u\}$ and a shortest path from v to u is an optimal Steiner tree for S. The algorithm considers all subsets of the set of terminals in order of increasing size, and for each subset, considers every decomposition according to the theorem described above. At each step, the smaller optimal Steiner trees needed in the decomposition have already been computed and stored from previous iterations. The resulting algorithm has time complexity $O(m3^n)$. The algorithm of Dreyfus and Wagner is probably the most popularly used to date for computing optimal RSTs in practice (its use is prescribed in, e.g., [4, 5, 24]).

Bern [2] presents a number of modified versions of the Dreyfus–Wagner algorithm for particular types of planar graphs. One of them is an algorithm for a planar graph in which many of the terminals lie on the border of the infinite face. Let T denote the set of input terminals, and let $T_b \subseteq T$ be the set of terminals that lie on the border of the infinite face. An *interval* is a set of terminals that are consecutive around the border of the infinite face. In applying the Dreyfus–Wagner algorithm, it suffices to consider only sets S such that $S \cap T_b$ is an interval. Furthermore, in the decomposition step it suffices to consider only subsets A and B such that $A \cap T_b$ and $B \cap T_b$ are also intervals. If $k = |T_b|$, then the resulting algorithm has time complexity $O(n^2 k^3 3^{n-k})$. Bern conjectures that this algorithm would perform well for the RST problem since the convex-hull reduction typically leaves many terminals on the border of the infinite face.

Shore, et al. [22] present a branch-and-bound algorithm for the GST problem (Yang and Wing [28] present a slightly less efficient version of the same algorithm). Branching is accomplished by fixing an edge to be included in or excluded from the optimal Steiner tree. Lower bounds are computed by considering minimum connectivity

properties among the various components in a partial solution. The algorithm has worst-case time complexity $O(2^{|E|})$, which can be as large as $O(2^{4(n+m)})$.

Beasley [1] presents a more involved branch-and-bound algorithm for the GST problem. Here, branching is accomplished by fixing nonterminals to be included in or excluded from the optimal Steiner tree. Lower bounds are computed by considering a formulation of the GST problem as a minimum spanning tree (which Beasley calls a shortest spanning tree) in an augmented version of the graph. Beasley formulates this version of the problem as a linear program and considers a Lagrangean relaxation of the linear program. A strength of his relaxation is that it is an unconstrained minimum spanning tree problem and thus can be solved efficiently without actually solving a linear program. Lower bounds are computed in this manner and then iteratively improved using subgradient optimization. Beasley's algorithm performs quite well in many types of graphs, but its performance when applied to the Hanan grid graph has not been previously examined.

Many other algorithms have been devised for the GST problem, most of which are also based on relaxations of integer programming formulations of the problem. Limited experimental evidence is available comparing these various algorithms, but it appears that Beasley's algorithm described above is among the best of them [15].

3. Geometric algorithms

A number of algorithms have also been devised that solve the RST problem directly in a geometric fashion, without the use of Hanan's graph reduction. This section surveys all such known algorithms.

3.1. Hwang's theorem

One important result concerning the RST problem is *Hwang's theorem* [14], which is exploited by many of the geometric RST algorithms described below. A *full set* of terminals is one for which, in every optimal RST of the terminals, every terminal is a leaf. An RST of a full set is called a *full tree*. Hwang proved that a rectilinear full tree can have only one of two simple topologies, which are illustrated in Fig. 2. Hwang's theorem has two important algorithmic implications. The first is that if a set of terminals is to be a full set, then it must be connectable according to one of the topologies specified by Hwang's theorem. The second is that if a set of terminals is a full set, then its optimal RST can be computed in linear time using Hwang's theorem.

3.2. Algorithms

Thomborson et al. [24] present an algorithm that lies at the interface between graph and geometric algorithms. They modify the Dreyfus–Wagner algorithm by exploiting

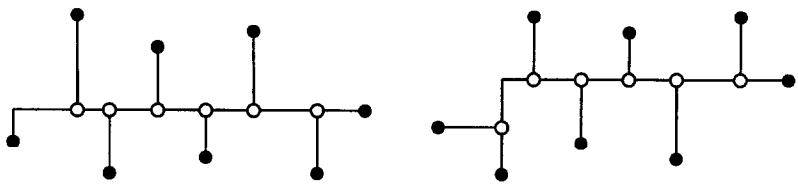

Fig. 2. Possible full tree topologies according to Hwang's theorem.

some geometric properties of the RST problem. These modifications do not change the asymptotic run time of $O(m3^n)$, but they do make a linear improvement in a subdominant term in the runtime and speed up the algorithm in practice, as will be shown below.

Ganley and Cohoon [8–10] present two dynamic programming algorithms for computing optimal RSTs. The first is called *full-set dynamic programming* (*FDP*). The basic idea is that an optimal RST of any set S of terminals is either a full tree, and is thus computable in linear time using Hwang's theorem, or else it is composed of two smaller RSTs joined at a terminal. Thus, the algorithm enumerates all subsets S of the input set T of terminals in order of increasing cardinality. For each S, the algorithm computes an optimal full tree using Hwang's theorem, and it enumerates all subsets A and B of S such that $A \cup B = S$ and $|A \cap B| = 1$. The shortest RST seen is an optimal RST of S. Since the subsets are enumerated in order of increasing cardinality, at each step the optimal RSTs of the smaller subsets A and B have already been computed and stored. The FDP algorithm has time complexity $O(n3^n)$. Their second algorithm, called *screened full-set dynamic programming* (*SFDP*), is a modification of the FDP algorithm using the concept of *full-set screening*. The basic idea behind full-set screening is that many subsets of the set of terminals cannot be full sets. Thus, the algorithm applies a number of polynomial-time tests to each subset of the set of terminals to potentially eliminate it from candidacy as a possible full set. Once a set of candidate full sets is thus identified, the FDP algorithm is modified to require, in the decomposition step, that one of the two subsets into which a set is decomposed be a candidate full set. Adding some full-set handling mechanisms to the FDP algorithm, along with an upper bound of $O(n1.62^n)$ on the number of candidate full sets on n terminals, Ganley and Cohoon present the SFDP algorithm with time complexity $O(n^2 2.62^n)$.

Salowe and Warme [21] use a slightly different approach to decomposing an optimal RST into full sets. They use the same tests as the SFDP algorithm [9, 10] to produce the set of candidate full sets (indeed, the SFDP implementation uses full-set screening code provided by Salowe and Warme), but use branch-and-bound instead of dynamic programming to find the optimal decomposition into candidate full sets. The branch-and-bound algorithm uses a number of novel techniques to efficiently prune and examine the search space, and as will be shown in Section 5, the algorithm performs very well in practice. Unfortunately, the best known bound on its worst-case time

complexity is $O(2^{n1.62^n})$, derived from the bound on the number of candidate full sets proven by Ganley and Cohoon [9, 10].

A number of other geometric algorithms for computing optimal RSTs have been devised [18, 23, 27], but they are less efficient than at least some of the algorithms described above, and thus we do not consider them further.

4. Dynamic programming: Implementation notes

A large portion of the running time of the dynamic programming algorithms ([2, 6, 8–10, 24]) is incurred by the manipulation of subsets. In these algorithms, two types of subset manipulation are performed:

(1) Enumerate all m-element subsets of an n-element set, and

(2) Enumerate all proper subsets of a m-element set.

A substantial savings in running time is accomplished by performing these operations efficiently. This section describes techniques for doing so.

In our implementations, sets of terminals are represented as bits in an integer. Each bit is on if the corresponding element is a member of the set. Operation (1) is performed in the outermost loop of the dynamic programming algorithms. For each value of m, the algorithm enumerates every m-element subset of the set of terminals. The enumeration of all m-element subsets of an n-element set is accomplished by the following C code due to Inada [16]:

```
#define FirstSet(m) ((1 << (m)) - 1)

int NextSet(int p) {
   int n, l, r;

   r = prev & -prev;
   n = prev + r;
   l = n & -n;
   return (n + (((l - r) / r) >> 1));
}
...
d = FirstSet(m);
do {
   /* handle subset represented by d */
   d = NextSet(d);
} while ((d & (1 << (n - 1))) == 0);
```

This enumeration requires a constant number of arithmetic operations for each subset.

Operation (2) is performed in the inner loop of the dynamic programming algorithms, for each m-element subset enumerated by operation (1). This operation is also accomplished in constant time per subset by a technique attributed to Deneen and Shute

by Thomborson et al. [24]. The following C code implements this operation:

```
#define NextSubset(e, d) ((d) & ((e) - (d)))
#define FirstElement(d) (NextSubset(0, d))
...
for (e = FirstElement(d); e < (d - e); e = NextSubset(e, d)) {
   /* handle subsets represented by e and (d - e) */
}
```

The use of these techniques speeds up the dynamic programming algorithms considerably. More naïve approaches require $O(m)$ time per subset, whereas these techniques require constant time per subset if an arithmetic or bitwise operation is considered to require constant time. Of course, this is an unreasonable assumption in general, but in practice, the exponential space complexity of these algorithms restricts them to instances of fewer terminals than can be represented by a standard 32-bit integer. (Note also that the asymptotic time complexity of the algorithms is not affected by the choice of assumptions that these operations require $O(m)$ versus $O(1)$ time.)

5. Experimental results

We have implemented the algorithms of Beasley [1], Bern [2], Dreyfus and Wagner [6], Ganley and Cohoon [8–10], Hakimi [12], and Shore et al. [22]. We obtained implementations of the algorithms of Salowe and Warme [21] and Thomborson et al. [24] from their authors.

We test each of them on 100 randomly generated instances for each instance size. Our testing platform is a Sun Sparc-10™ workstation. We now present these results in two formats. Fig. 3 plots the average running times of each algorithm as a function of the number n of terminals.

Fig. 4 shows the size of an instance that each algorithm can solve in 15 minutes of CPU time. The running times of many of the algorithms exhibit high variance, so the width of the bars at each instance size is proportional to the number of instances that were completed within the 15 minute time limit.

As is expected from its time complexity, Hakimi's algorithm [12] is quite slow. It cannot solve problems of more than 10 terminals in 15 minutes of CPU time on a workstation.

The algorithm of Dreyfus and Wagner [6] performs a bit better. For the most part, it can solve 15-terminal problems in 15 minutes. The few 16-terminal problems that completed within the 15 minute time period are precisely those for which the terminal reduction eliminated one or more terminals.

Bern's algorithm [2] does not perform as well as one might hope. Bentley et al. [2] prove that the expected number of terminals on the border of the infinite face after the convex-hull reduction is $O(\log n)$, in which case Bern's improvement makes only a polynomial change in the time complexity of the Dreyfus and Wagner algorithm.

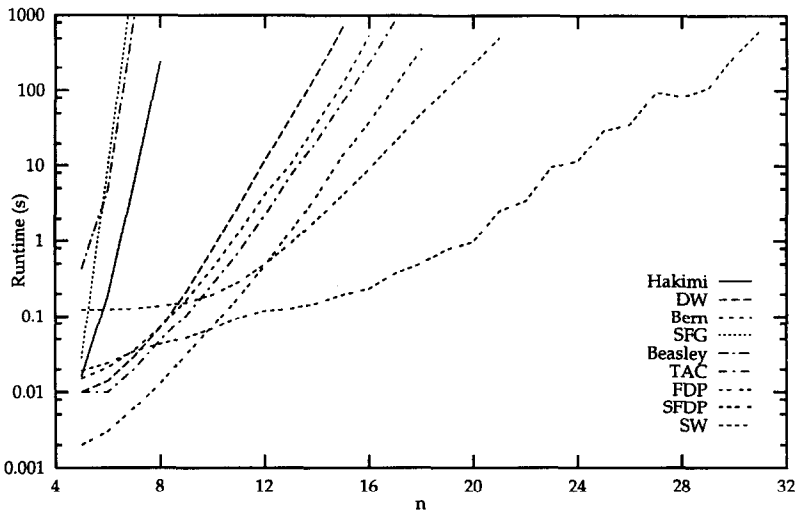

Fig. 3. Running times of the exact RST algorithms.

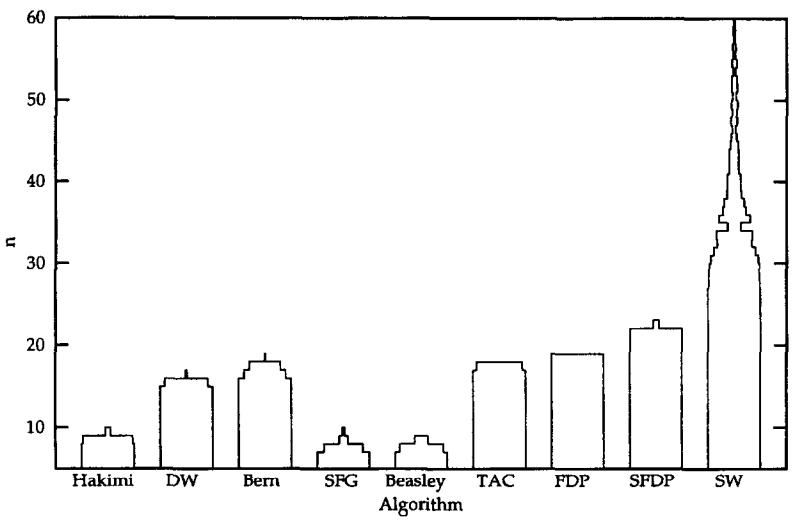

Fig. 4. Instances completed within 15 min of workstation CPU time.

In discussing the possible application of his algorithm to the RST problem, Bern conjectures that in spite of this asymptotic result, for small instances a sufficiently large number of terminals might lie on the border of the infinite face to make a substantial difference in the practical running time. However, even for small numbers of terminals, the number of terminals on the border of the infinite face seems to be logarithmic. Table 1 gives the average number of terminals on the infinite face after the convex-hull reduction for our test cases.

Table 1
Number k of terminals on the infinite face for test instances containing n terminals

n	3	4	5	6	7	8	9	10
k	3.0	4.0	4.5	5.2	6.0	6.6	7.0	7.6
n	11	12	13	14	15	16	17	18
k	7.8	8.2	8.9	8.8	8.9	9.8	9.8	9.9

The algorithm of Shore et al. [22] performs quite badly. The reason is that the lower bounds used in the algorithm, while fairly good for some types of graphs, are very loose in the Hanan grid graph. As a result, the algorithm examines much of the search space, which has size up to $O(2^{4(n+m)})$. The reader may note that this algorithm is slower than Hakimi's algorithm [12], which has time complexity $O(n^2 2^m)$.

Beasley's algorithm [1] performs much more poorly than one might expect, considering that it is among the best algorithms for the general GST problem. We attribute this drastic difference to two factors. The first is that, as mentioned in Section 2.1, the graph reductions used by Beasley perform quite poorly in the Hanan grid graph. The second is that the lower bounds produced by Beasley's lagrangean relaxation are far less tight in the Hanan grid graph than in the other types of graphs for which he presents results. Beasley's algorithm is quite intricate, and one might worry about errors in our implementation. However, we have tested our implementation on several of the test problems used by Beasley and it performs comparably (within a small constant factor of the times presented by Beasley, which may be accounted for by the fact that Beasley's testing platform is a Cray X–MP).

The algorithm of Thomborson et al. [25] is faster than Bern's algorithm [3] by roughly a factor of 2. This is not surprising, since both algorithms improve on the algorithm of Dreyfus and Wagner [6] by a factor of roughly $O(n)$.

The FDP algorithm of Ganley and Cohoon [8, 10] outperforms all those discussed so far. This is particularly interesting since the algorithm is so simple. It is also noteworthy that for small instances (those with fewer than 10 terminals), the FDP algorithm is the fastest of those tested.

Predictably, the SFDP algorithm of Ganley and Cohoon [9, 10] performs still better than the previous algorithms. Its time complexity, however, is still bounded from below by the $O(2^n)$ time incurred by enumerating every subset of the set of terminals. Ganley and Cohoon [9, 10] have proposed improvements that may reduce the number of subsets that must be considered, but these improvements have yet to be implemented.

For the instances tested here, the algorithm of Salowe and Warme [21] far outperforms all the others. For 10 or more terminals, it is the fastest of all the algorithms tested. Unfortunately, like the other branch-and-bound algorithms, its running time is quite erratic (for example, for the 30-terminal instances tested, the running times range

from a minimum of just under 1 second to a maximum of roughly 2.5 hours). Nonetheless, even its worst-case running times are typically faster than all the other algorithms, and it completes more than half of the instances with 33 or fewer terminals within 15 minutes.

6. Summary

One question that arises from these experiments is why the branch-and-bound algorithms, which work so well on other types of graphs, work so poorly when applied to the Hanan grid graph. It appears that the lower bounds used in these algorithms are far less tight in the Hanan grid graph than in other types of graphs. It would be interesting to determine why this is the case and perhaps to use these insights to develop lower bounds better suited to the RST problem.

A user faces a bit of a quandary when trying to select an exact RST algorithm. On one hand, the algorithm of Salowe and Warme [21] is the fastest in practice for randomly generated instances, and probably for all instances. However, in the absence of good worst-case bounds on its time complexity, it is impossible to know whether there might be pathological instances for which its performance is much worse than it is on average. On the other hand, several of the other algorithms, such as the algorithm of Dreyfus and Wagner [6] and the algorithms of Ganley and Cohoon [8–10] have good worst-case time bounds, but are outperformed in practice by the algorithm of Salowe and Warme. One possible solution, suggested by Robins [20], is to run the Salowe and Warme algorithm in parallel with one of the algorithms with good time bounds and to stop both when one terminates. In this way, one achieves both the good running time in practice and the good bound on worst-case time complexity.

Such a solution should probably be viewed as an interim measure; hopefully future work on the SFDP algorithm [9, 10], the Salowe and Warme algorithm [21], or some entirely new algorithm will result in an algorithm that is fast in practice *and* has a good worst-case time complexity.

Note added in proof. While this paper was being reviewed, Warme [25] made substantial improvements to the algorithm described in Salowe and Warme [21]. The improvements result partly from reordering and interleaving the various full-set screening tests used in the first phase of the algorithm, but mostly from the replacement of the second phase with a sophisticated integer-programming-based branch-and-cut algorithm. The new algorithm can solve problems containing *hundreds* of terminals within the 15 minute time limit used in this paper!

References

[1] J.E. Beasley, An SST-based algorithm for the Steiner problem in graphs, Networks 19 (1989) 1–16.
[2] J.L. Bentley, H.T. Kung, M. Schkolnick, C.D. Thompson, On the average number of maxima in a set of vectors and applications, J. ACM, 25 (1978) 536–543.
[3] M. Bern, Faster exact algorithms for Steiner trees in planar networks, Networks 20 (1990) 109–120.

[4] L.L. Deneen, J.B. Dezell, Using partitioning and clustering techniques to generate rectilinear Steiner trees, in: Proc. 2nd Canadian Conference on Computational Geometry, (1990) pp. 315–318.
[5] L.L. Deneen, G.M. Shute, C.D. Thomborson, A probably fast, provably optimal algorithm for rectilinear Steiner trees, Random Structures and Algorithms 5 (1994) 535–557.
[6] S.E. Dreyfus, R.A. Wagner, The Steiner problem in graphs, Networks 1 (1972) 195–207.
[7] J.L. Ganley, Geometric interconnection and placement algorithms, PhD thesis, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 1995.
[8] J.L. Ganley, J.P. Cohoon, A faster dynamic programming algorithm for exact rectilinear Steiner minimal trees, in: Proc. 4th Great Lakes Symposium on VLSI (1994) pp. 238–241..
[9] J.L. Ganley, J.P. Cohoon, Optimal rectilinear Steiner minimal trees in $O(n^2 2.62^n)$ time, in: Proc. 6th Canadian Conference on Computational Geometry (1994) pp. 308–313.
[10] J.L. Ganley, J.P. Cohoon, Improved computation of optimal rectilinear steiner minimal trees, Int. J. Comput. Geometry Appl. 7 (1997) 457–472.
[11] M.R. Garey, D.S. Johnson, The rectilinear Steiner tree problem is NP-complete, SIAM J. Appl. Math. 32 (1977) 826–834.
[12] S.L. Hakimi, Steiner's problem in graphs and its implications, Networks 1 (1971) 113–133.
[13] M. Hanan, On Steiner's problem with rectilinear distance, SIAM J. Appl. Math. 14 (1996) 255–265.
[14] F.K. Hwang, On Steiner minimal trees with rectilinear distance, SIAM J. Appl. Math. 30 (1976) 104–114.
[15] F.K. Hwang, D.S. Richards, P. Winter, The Steiner Tree Problem, Vol. 53, Annals of Discrete Mathematics, North-Holland, Amsterdam, Netherlands, 1992.
[16] D.M. Inada, private communication, (1995).
[17] J. Komlós, M.T. Shing, Probabilistic partitioning algorithms for the rectilinear Steiner problem, Networks 15 (1985) 413–423.
[18] F.D. Lewis, W.C. Pong, N. Van Cleave, Optimum Steiner tree generation, in: Proc. 2nd Great Lakes Symposium on VLSI (1992) pp. 207–212.
[19] J.S. Provan, Convexity and the Steiner tree problem, Networks 18 (1988) 55–72.
[20] G. Robins, private communication, 1995.
[21] J.S. Salowe, D.M. Warme, 35-point rectilinear Steiner minimal trees in a day, Networks 25 (1995) 69–87.
[22] M.L. Shore, L.R. Foulds, P.B. Gibbons, An algorithm for the Steiner problem in graphs, Networks 12 (1982) 323–333.
[23] A.F. Sidorenko, On minimal rectilinear Steiner trees, Diskretnaya Matematika, 1 (1989) 28–37 (in Russian).
[24] C.D. Thomborson, B. Alpern, L. Carter, Rectilinear Steiner tree minimization on a workstation, In N. Dean, G. E. Shannon (eds.) Computational Support for Discrete Mathematics, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 15, American Mathematical Society, Providence, RI, 1994 pp. 119–136.
[25] D.M. Warme, A new exact algorithm for rectilinear Steiner trees, Presented at the 1996 INFORMS Conf.; available at `http://www.s3i.com/~warme/`.
[26] P. Winter, Reductions for the rectilinear Steiner tree problem, Networks 26 (1995) 187–198.
[27] Y.T. Wong, M. Pecht, A solution for Steiner's problem, in M. Pecht (ed), Placement and Routing of Electronic Modules, Marcel Dekker, New York, NY (1993) pp. 261–304..
[28] Y.Y. Yang, O. Wing, Optimal and suboptimal solution algorithms for the wiring problem, in: Proc. Int. Symp. on Circuit Theory (1972) pp. 154–158.