ELEKTRİK, VOL.1, NO.2, June 1993, ©TÜBİTAK EMO

Performance Oriented Rectilinear Steiner Trees*

Andrew Lim
Information Technology Institute, National Computer Board
71 Science Park Drive, Singapore 0511
Siu Wing Cheng
Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
Ching Ting Wu
Department of Computer Science, University of Minnesota
3M, St. Paul, Minnesota 55144, USA

Abstract

In this paper, we present a new formulation of the rectilinear Steiner tree problem (MRST) that takes performance into consideration, and we call it the performance oriented minimum rectilinear Steiner tree problem (POMRST). The POMRST problem is a more general version of the minimum rectilinear Steiner tree problem. This formulation is especially useful in the case of net connection high performance circuits in VLSI CAD. Recently, a related but less general problem has been addressed in [4].Since the POMRST problem is also NP-hard, we provide an effective heuristic for it. When we apply our POMRST heuristic to solve the MRST problem, our experimental results indicate an average of 10.64% improvement over the minimum spanning tree. This compares favorably with the existing techniques cited in [17]. Optimal solutions are obtained for test cases of 3 to 6 points. In the context of the POMRST problem, we tried test cases with widely different percentages of critical source-sink pairs and our experimental results indicate that a small increase in the total interconnection length can greatly enhance the circuit performance.

1. Introduction

Let $S = \{p_1, p_2, ..., p_n\}$ be a set of n points in the plane. A Minimum Rectilinear Steiner Tree for the set S, denoted by MRST(S), is a minimum length tree with rectilinear edges that connect all the points in the set S. Unlike spanning trees, there are some vertices with degree 3 or 4 in the resulting MRST(S) that do not belong to S. These points are known as *Steiner points.*

The rectilinear minimum Steiner tree problem can be used in the layout of electrical wires, pipes, etc., and in building constructions and connecting signal points of a net together in VLSI routing. An excellent survey of the Steiner tree problem can be found in [9]. Unfortunately, the MRST problem is found to be NP-complete [7]. This implies that an efficient optimal algorithm for the MRST problem is unlikely to be found. However, efficient optimal algorithms have been developed for some special cases [1, 2, 3, 16].

* This work was done while the first author was at the Department of Computer and Information and Computer Sciences, University of Florida, Gainesville; and the second and third authors at the Department of Computer Science, University of Minnesota, Twin Cities.

Hwang [8] was able to establish that

$$\frac{L_{\text{MRSpT(S)}}}{L_{\text{MRST(S)}}} \leq \frac{3}{2}$$

where $L_{\text{MRSpT(S)}}$ is defined to be the length of the minimum rectilinear spanning tree and $L_{\text{MRST(S)}}$ to be length of the minimum rectilinear Steiner tree. It is an open question if there exists any heuristic that has a lower worst case bound.

Up until now, only one exact algorithm for the MRST problem has been proposed [20, 21]. Yang provided a straightforward branch and bound algorithm that can only solve very small test cases. Due to the immense computing time required to obtain the optimal solution for the MRST problem, all heuristics tabulate their performance by comparing themselves against the MRSpT result. The performance of existing techniques to find approximate solutions to the MRST problem are surveyed in [17].

The MRST is commonly used in VLSI for connecting n points of a net. A net has a designated signal source. The rest of the points in the net are signal destinations (or signal sinks). The performance of a circuit is determined by the critical paths in the circuit. So, if a source-sink pair is on a critical path, the connection between them should be made as short as possible. While the MRST solution connects the net with the minimum total wire length, the issue of performance of the connection is not considered. It was reported in [5,12,18,19] that in some high performance circuits, the interconnection delay may contribute up to 70 % of the total delay of the circuits. Since, performance of a design is becoming the differentiating factor in the success/failure of a VLSI product, constructing a Steiner tree with consideration of performance is important. Recent progress on timing driven global routing [6,10,14] is cited in [4], while a solution that exhibits trade-off between interconnection delay and routing cost is first proposed in [4].

In Section 2, we first provide the formulation of the performance oriented minimum rectilinear Steiner tree problem, denoted by POMRST, and discuss its complexity. We assume a commonly used delay model where the propagation delay of a signal between 2 points is proportional to the distance between them. In simple terms, we introduce additional maximum distance constraints on the distance between each source-sink pair in the resulting Steiner tree. Note that due to possibly different requirements, we allow the distance constraint for one source-sink pair to be different from the distance constraint for another pair. In [4],*a nonuniform bounded radius routing tree (NBRMRT) problem* is proposed (as well as an approximation algorithm) which can be expressed as the POMRST problem with a maximum distance allowed from source to a sink d_i is equal to $(1+\epsilon_i).R$. R is the distance between the source and the furthest sink and ϵ_i is a constant greater than 0. This implies that the tightest maximum distance constraint imposed on any source-sink pair is greater than R. However, the sink of a very critical source-sink pair may lie anywhere in the routing area. Therefore, the NBRMRT formulation fails to cater for critical source-sink pairs in which the sinks are not at the greatest distance from the source. Our formulation provides more flexibility in this regard.

In Section 3, we present our heuristic to solve the POMRST problem. The heuristic works iteratively to grow a Steiner tree in a greedy fashion.

In Section 4, we present experimental results on our heuristic using randomly generated test cases. The experiments involve test cases with and without maximum distance constraints (i.e., the MRST problem). For the MRST problem, we obtain an average of 10.64% improvement over the minimum spanning tree. This compares favorably with the existing techniques cited in [17]. For comparison, we also program a branch-and-bound algorithm for the MRST problem. We ran our branch-and-bound algorithm on some randomly generated test cases of 3 to 6 points which verifies the optimality of the solutions obtained by our heuristic for the test cases. In the context of the POMRST problem, we tried test cases with widely different

percentages of critical source-sink pairs and our experimental results indicate that a small increase in the total interconnection length can greatly enhance the circuit performance.

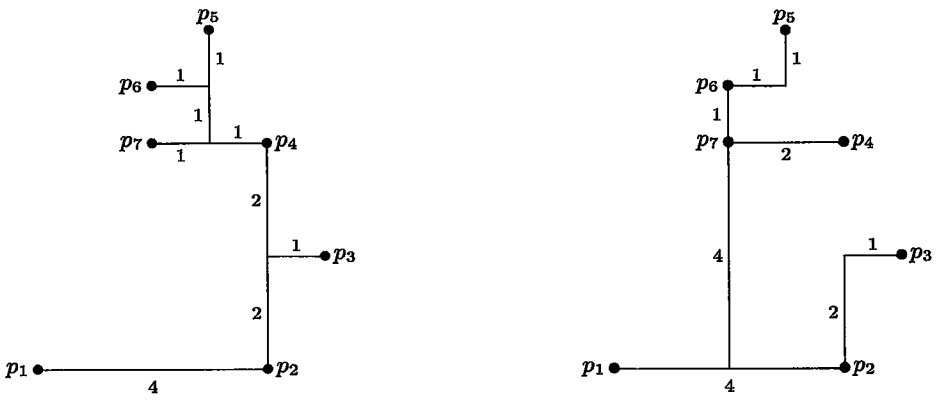
We also list our input generation procedures so as to facilitate comparison of our work with new results to be obtained in future.

2. Problem Formulation

Below is the formulation of the POMRST problem:
Given a set, $S = \{p_1, p_2, \ldots, p_n\}$, where S is a set of points on a plane, a designated source $s \in S$, a maximum distance function $md : s \times (S \setminus s) \longrightarrow Z^+$, and an edge distance function $\lambda(p,q) = |p_x - q_x| + |p_y - q_y|$, i.e., the rectilinear distance between points p and q. Let S' be all the intersection points when horizontal and vertical lines are drawn through all the points in S. The POMRST problem is to find a spanning tree, $G^t = (T, E_T)$ which is a subgraph of $G = (S', S' \times S')$ and $S \subseteq T$ such that:

$$\begin{aligned} &\min \textstyle\sum_{e \in E_T} \lambda(e) \\ &\text{st:} \\ &\quad \mathit{distance}(s,p) \leq md(s,p), \quad \forall p \in S \setminus s \end{aligned}$$

where
the $\mathit{distance}(s,p)$ is the length of the path from s to p in the tree constructed and
$md(s,p)$ is the maximum possible distance between s and p.



(a) **MRST** (b) **POMRST**

p_1 is the source and $p_2, \ldots, p_7$ are destinations. Assume the rectilinear distances of p_1 to $p_2, \ldots, p_7$ are 4,7,8,9,7,6 respectively (minimum possible distances). For **MRST**, the total length of interconnection is 14 and the distances between p_1 and $p_2; p_3, p_4, p_5, p_6, p_7$ are $4, 7, 8, 11, 11, 10$ respectively. The total length of interconnection for **POMRST** is 16. The distances between p_1 and $p_2, p_3, p_4, p_5, p_6, p_7$ are $4, 7, 8, 9, 7, 6$ respectively.

Figure 1. MRST and POMRST solutions for connecting 7 points.

In the MRST problem, we set *md (s,p)* to ∞ or a very large number. This implies the removal of the maximum distance (or maximum delay in the context of VLSI) constraints.

In the performance oriented minimum rectilinear Steiner tree (POMRST) problem, *md(s,p)* is set to a certain value, depending on whether the source-sink pair (s,p) is on any critical path. Since, the MRST

problem is NP-complete, and the MRST problem is a special case of the POMRST problem, the POMRST problem is also NP-hard

Figure 1 illustrates the difference in the construction for MRST and POMRST solutions. Note that by increasing the total wire length by 2 units, the distance between $(p_1, p_5), (p_1, p_6)$, and (p_1, p_7) is decreased by 2,4 and 4, respectively.

3. The POMRST Heuristic

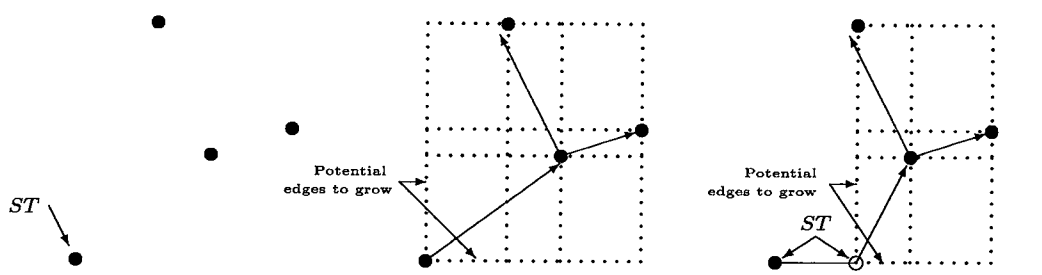
3.1. An Overview

Our heuristic works iteratively by growing the Steiner tree one edge at a time. The initial Steiner tree contains the source only. Each iteration is divided into two stages. Let ST be the current Steiner tree at the beginning of each iteration and let V_1 be the set of vertices contained in ST.

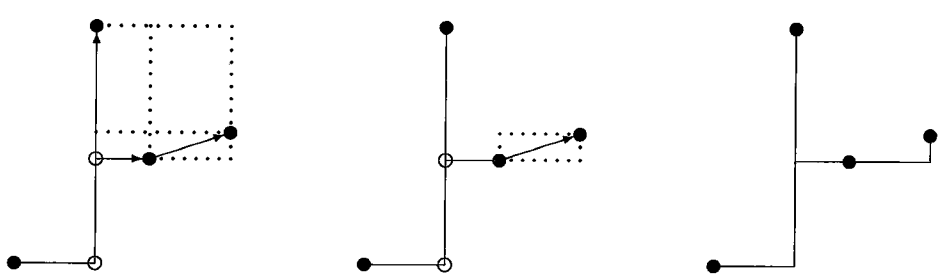
In the first stage, we run a *performance-oriented spanning tree* construction procedure to connect the points in $S - V_1$ to ST. We denote each new edge introduced at this stage as a *spanning tree edge.* Note that every spanning tree edge is directed.

In the second stage, we grow ST by one *grid edge.* The grid edge is selected from the grid formed by drawing horizontal and vertical lines through the points in $S - V_1$. The spanning tree edges discovered in the first stage will be used to guide our selection. Note that the grid edge selected may not necessarily be a spanning tree edge. Intuitively, the selection process involves a weighting scheme that favors the grid edge that will allow a lot of sharing and bring the points in $S - V_1$ "closer" to the new ST.

Figure 2 illustrates the growing of the Steiner tree. In Section 3.2., we formulate the performance-oriented spanning tree problem and describe a heuristic for it. The details of the weighting scheme for selecting a grid edge are presented in Section 3.3.



(a) Pts to be connected (b)POMSpT & grid plane (c) Grow ST along a grid edge



(d) Repeat and grow (e) Repeat and grow (f) POMRST generated

Figure 2. Growing of Steiner tree.

3.2. Performance-oriented Spanning Tree

The performance-oriented minimum spanning tree problem (POMSpT) is formulated below. In simple terms, the goal is to construct a minimum length spanning tree such that the maximum distance constraint for each source-sink pair is satisfied. In general, the edge distance function can be arbitrary, but it is specialized to be the rectilinear distance in this context.

Given a graph $G = (V, E)$, where V is the set of vertices, E the set of edges, a source vertex $s \in V$, an edge distance function $\lambda : E \longrightarrow R^+$, a maximum distance function between any vertex and s, $md : s \times (V \setminus s) \longrightarrow R^+$, and a value $B \in R^+$. Find a spanning tree T, if possible, that satisfies,

$$\lambda(T) = \sum_{e \in E_T} \lambda(e) \leq B$$
$$distance(s, i) \leq md(s, i), \ \forall i \in V \setminus s$$

Each spanning tree edge (u, v) is drawn as a directed arc from u to v, but the length of (u, v) is the rectilinear distance between u and v.

In [11], it is proved that the POMSpT problem is NP-hard for an arbitrary distance function λ. It is also proved that the POMSpT problem remains to be NP-hard in the Euclidean plane. It is also demonstrated that the ratio of the lengths of a POMSpT and the MST is not bounded by any constant.

We extend the well-known Prim's minimum spanning tree algorithm to a heuristic for the POMSpT problem. Figure 3 provides the details of the procedure.

Procedure POMSpT($Init\_T, V_1, V_2, md$);
$SPT := Init\_T$; done := **false**
while $V_2 \neq \emptyset$ **and not** done **do**
begin
  let (u, v) be a least cost edge such that $u \in V_1$ and $v \in V_2$
  and $dist(s, u) + |u_x - v_x| + |u_y - v_y| \leq md((s, v))$;
  if (no such edge exist) **then** done := **true**
  else add v to V_1, add (u, v) to SPT, remove v from V_2
  $dist(s, v) := dist(s, u) + |u_x - v_x| + |u_y - v_y|$;
end;
if $V_2 \neq \emptyset$
  then writeln('No solution can be found');

Figure 3. Heuristic for the POMSpT problem.

In order to be used as a subroutine to solve the POMRST problem, the procedure accepts three input parameters: (a) $Init\_T$ – the starting tree to be grown which is represented as a set of tree edges, (b) V_1 – the vertices contained in $Init\_T$, and (c) V_2 – the vertices to be connected. The rectilinear distance function is used in the procedure.

3.3. The Heuristic and the Growing Strategy

We describe in this section the remaining details of our POMRST heuristic. Figure 4 illustrates the framework of the heuristic. The heuristic starts with a partially formed Steiner tree, ST, which contains only the source s. It will continually grow the partially formed Steiner tree until V_2 becomes empty (i.e., all the points in S are in the Steiner tree ST).

```
Algorithm POMRST(S, s, md);
   /* To find Minimum Rectilinear Steiner Tree that satisfy
      the maximum distance constraints, S is the set of points,
      s is the source and md are the distance constraints */
begin
   ST := ∅;
   V1 := {s};    /* Steiner tree with vertex s */
   V2 := S − {s};
   while V2 ≠ ∅ do
   begin
      call POMSpT(ST, V1, V2, md);
      perform no-choice-grow;
      if no-choice-grow did nothing then
      begin
         grid plane;
         calculate the weight for the grid edges;
         grow ST: select a grid edge (a, b), ST := ST ∪ {(a, b)},
                  V1 := V1 ∪ {b}, V2 := V2 − {b};
      end;
   end
   remove dangling edges;
end;
```

Figure 4. Outline of the heuristic.

To decide where and how to grow ST, the POMSpT procedure of the previous chapter is used. Using the POMSpT procedure, unconnected points in S are joined to ST with spanning tree edges. Suppose that the POMSpT procedure has been invoked to introduce the spanning tree edges. If there is a vertical or horizontal spanning tree edge (a, b) for some $a \in V_1$ and some $b \in V_2$, and the distance from the source s to a plus $length((a, b))$ [1] is equal to $md(s, b)$, then we invoke the no-choice-grow subroutine to grow the Steiner tree by the edge (a, b). In the event that there are several such points in V_2, the point b with the smallest $md(s, b)$ is selected. The justification for performing the no-choice-grow is that since these spanning tree edges will inevitably be included in the Steiner tree, we may as well do it as early as possible so that this may help in connecting other unconnected sinks in the future.

If the no-choice-grow procedure is not invoked, then we have to consider other alternatives to grow ST. One immediate way to obtain a solution to the POMRST problem is to replace each spanning tree edge (a, b) by two edges (a, c) and (c, b), where $c = (b_x, a_y)$ and b_x and a_y are the x-coordinate and y-coordinate of b and a, respectively. However, this method is too greedy and does not exploit the fact that a new Steiner tree edge introduced may be shared to improve the total length. So, we shall grow our Steiner tree in a less greedy manner and in a "consistent direction" with some of the spanning tree edges generated. For example, let (a, b) be a spanning tree edge as shown in Figure 5.

[1] $length((a, b))$ denotes the length of the spanning tree edge (a, b).

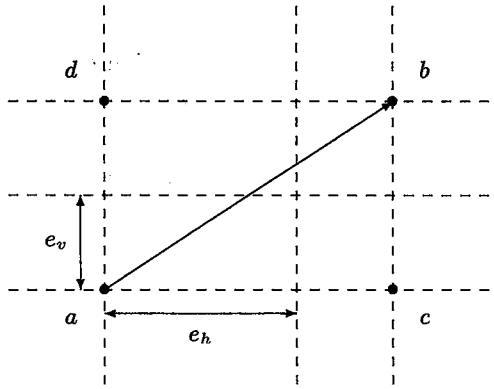


Figure 5.

If we choose to grow in a direction consistent with the spanning tree edge (a, b), we should grow from a towards c or d. To avoid being overly greedy, we shall grow by the grid edge e_h or e_v instead of (a, c) and (a, d). In reality, we do not explicitly select a spanning tree edge to guide the selection of the grid edge. Instead, we define the weight of each grid edge by all the incident spanning tree edges and select the grid edge with the maximum weight to be the new Steiner tree edge. The above is the sketch of the process. We describe the details below.

Let horizontal and vertical lines be drawn through the endpoints of all spanning tree edges to form a grid. Then we select a grid edge e that satisfies the following criteria:

1. e has exactly one endpoint u in V_1 (i.e., e is attached to ST).
2. u is also the endpoint of some spanning tree edge.
3. e has the maximum edge weight, $weight(e)$, among all the grid edges that satisfy (1) and (2).

To define $weight(\cdot)$, we first define a notion of $score(\cdot)$ for grid edges and grid points. At the beginning, $score(e)$ and $score(gp)$ are initialized to zero for all grid edge e and grid point gp.

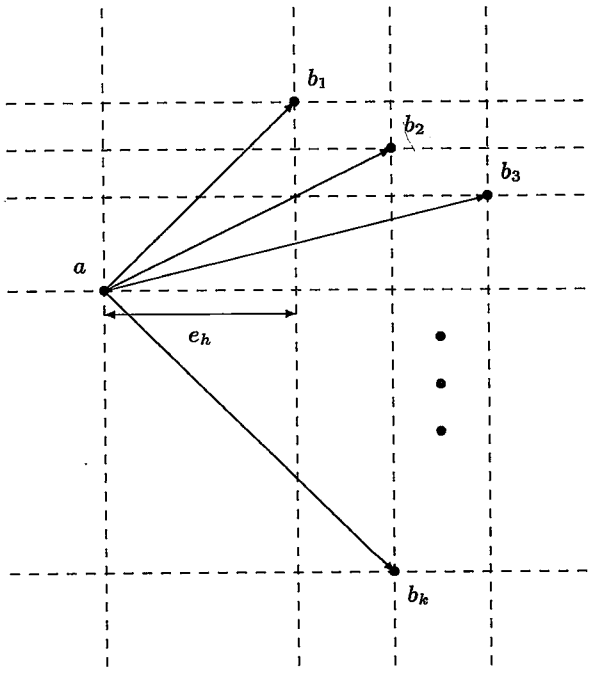


Figure 6.

Suppose that there are k spanning tree edges (a, b_i), $1 \leq i \leq k$, that are incident to a vertex a in ST and lie in the first or the fourth quadrant around a (refer to Figure 6). Let e_h be the horizontal grid edge that is incident to a and points to the right. Then $score(e_h)$ is set to $k\alpha$, where α is an experimentally determined parameter. The intuitive justification is that for large value of k, e_h will be more heavily shared and thus the future increase in total length is expected to be reduced. The scores of grid edges of other orientations can be defined symmetrically.

In addition, it is also important to consider where a grid edge leads us to. So we also give scores to the endpoints of the grid edges not belonging to V_1. Each spanning tree edge (a, b) defines a smallest box enclosing it. First, we prefer grid points that fall into more such boxes because it implies that a grid edge with such an endpoint will have direction "consistent" with more spanning tree edges. So, if a grid point is enclosed by m such boxes, its score will first be set to m. Second, we should prefer grid points that lie "closer" to the remaining unconnected points in V_2. Therefore, each grid point (gp_x, gp_y) receives an additional score of

$$\sum_{p=(p_x,p_y)\in V_2} \frac{\gamma}{\sqrt{(p_x - gp_x)^2 + (p_y - gp_y)^2} + \delta}$$

where γ and δ are experimentally determined parameters.

Finally, $weight((a, b))$ for a grid edge (a, b) is set to be:

$$length((a, b)) \cdot (score((a, b)) + score(b) - \sigma)$$

where σ is an experimentally determined parameter.

This completes the description of the weighting scheme. After we have connected all the points in S to ST, we need to remove all dangling grid edges that have been introduced previously. An edge is dangling if one of its end points is not in S. Note that this has to be applied recursively since the removal of a dangling edge may generate a new dangling edge.

4. Experimental Results

We implemented the POMRST heuristic in the C programming language and conducted experiments on some randomly generated test cases. To facilitate comparison of our work with future new results, we list in Figure 7 the procedures that we used to generate test cases. We used the random number generator in [13] (see Figure 8). The initial seed value is set to 13957.

```
procedure generate(seed, mx, my, v, n);
  /* To generate random n points on a plane mx by my plane.
     v stores the x and y co-ordinates */
begin
  for i := 1 to n do
  begin
    done := false;
    while done = false do
    begin
      done := true;
      v[i].x := trunc(mx * rnd(seed));
      v[i].y := trunc(my * rnd(seed));
      for j := 1 to i - 1 do
```

```
            if (v[j].x = v[i].x) and (v[j].y = v[i].y)
                then done := false;
        end;
    end;
end;
```

Figure 7. Procedure generate.

```
function rnd(seed):real;
    /* The random number generator for generating test cases */
begin
    hi := trunc(seed/127773);
    lo := seed − 127773 * hi;
    test := 16807 * lo − 2836 * hi;
    if test > 0.0
    then
        seed = test;
    else
        seed := test + 2147483647;
    rnd:= seed/m;
end;
```

Figure 8. Function random.

All experiments were run on a SPARCstation *SLC*. Since MRST is a special case of POMRST and there are a lot of previous work done on the MRST problem, we first apply our POMRST heuristic (with some improvement) to solve some MRST test cases. Then we also test our heuristic with some randomly generated POMRST test cases.

4.1. Experimental Results for MRST

We first tested our heuristic using $3 \leq n \leq 40$ for the POMRST problem with no maximum distance constraint. Without the maximum distance constraints, the POMRST is just the traditional minimum rectilinear Steiner tree problem.

Since there is no maximum distance constraint, any point can be the source. We first divide the gridded plane[2] into four quadrants by using the middle vertical and horizontal grid lines. By middle we mean that the number of grid lines on the left (top) and right (bottom) of the chosen vertical (horizontal) grid line differs by at most 1. A point is in a quadrant if it is located in the quadrant or on the boundary of the quadrant. Four different source points are chosen, one from each quadrant. The heuristic will be activated four times using each of the four different selected source points once. The best of the four solutions is reported.

The experimental results are summarized in Table 1. Figures 9 and 10 provide a graphical representation of our experimental results. For each n, 100 randomly generated test cases are generated and evaluated. This is done by generating n points randomly on a 100 unit by 100 unit plane. The second and third columns are the maximum and minimum improvement in total length of the rectilinear Steiner

[2] The grid is formed by drawing horizontal and vertical lines through all the points in S.

tree obtained by our heuristic over the minimum spanning tree (MRSpT), respectively. The fourth and fifth columns are the average improvement of 100 test cases and the best average improvement of the first $k > 20$ test cases over the MRSpT, respectively.

Table 1. Experimental result for MRST problem.

Size	*Impro. over all tries*		*Average Improv.*		*Time*
n	*Max*	*Min*	*All tries*	*Best > 20 tries*	*sec.*
3	24.15	0.00	6.98	7.47	0.001
4	20.56	0.00	8.53	9.23	0.002
5	24.05	0.00	9.70	10.41	0.004
6	21.80	1.89	9.98	11.51	0.007
7	18.06	3.12	10.18	10.91	0.011
8	18.99	2.78	10.80	11.00	0.017
9	20.00	3.61	10.43	10.88	0.023
10	16.33	3.66	10.41	11.71	0.034
11	19.63	4.82	10.44	11.24	0.047
12	19.38	4.86	10.38	11.67	0.060
13	15.79	3.91	10.20	10.67	0.075
14	16.40	5.38	10.60	11.26	0.094
15	15.94	4.01	10.59	11.08	0.120
16	16.38	2.98	10.73	11.39	0.151
17	16.98	6.10	10.83	11.31	0.185
18	15.31	6.77	10.88	10.97	0.210
19	14.67	5.56	10.28	10.73	0.248
20	16.79	5.84	10.60	10.93	0.301
21	14.36	6.46	10.57	10.98	0.380
22	15.33	6.39	10.74	10.97	0.459
23	15.14	6.67	10.64	11.10	0.535
24	15.14	6.57	10.68	11.33	0.609
25	16.22	5.09	10.48	11.26	0.695
26	14.78	5.87	10.63	10.94	0.790
27	14.93	5.82	10.69	10.84	0.896
28	14.17	3.96	10.77	11.30	1.05
29	15.12	6.70	10.95	11.48	1.22
30	15.79	7.26	10.74	11.50	1.40
31	16.43	7.33	10.62	11.12	1.62
32	15.23	5.31	10.54	10.85	1.87
33	15.37	6.92	10.81	11.56	2.10
34	16.07	7.63	11.02	11.52	2.41
35	17.06	7.08	10.81	11.45	2.73
36	14.29	6.69	10.69	11.52	3.05
37	13.92	7.66	10.88	11.26	3.67
38	14.03	7.40	10.67	10.89	4.06
39	14.29	7.05	10.78	11.02	4.54
40	13.34	7.51	10.81	11.27	5.12

For comparison, we also program a branch-and-bound algorithm to find the exact minimum rectilinear Steiner tree for a set of points. The details of our branch-and-bound algorithm are given in the appendix. Our branch-and-bound algorithm verifies that our POMRST heuristic obtains the optimal solution for our test cases with 3 to 6 points. For $7 \leq n \leq 40$, our heuristic has an average of 10.64% improvement over the MRSpT solutions. This compares favorably with existing results cited in [17].

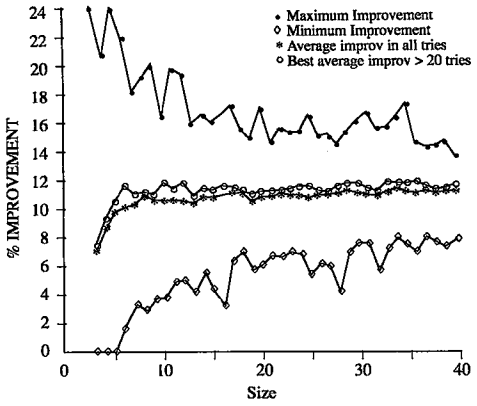


Figure 9. Graph of % Improvement vs size

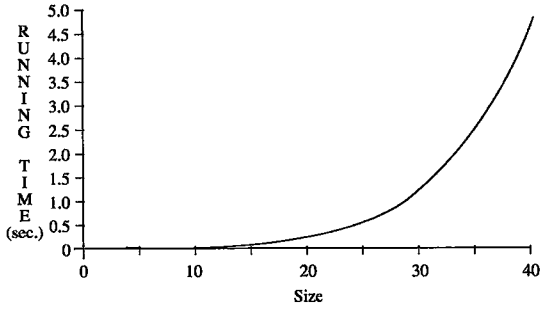


Figure 10. Graph of Running Time vs size

4.2. Experimental Results for POMRST

In this section, we present the experimental results of our heuristic on POMRST problems. We evaluated our heuristic for $n = 10, 15, 20, 25, 30, 35$ and 40. For each n, 20 randomly generated test cases are evaluated. For each n, n points are generated on a 250 by 250 unit plane. If a source-sink pair is critical, we set its maximum distance bound to be the rectilinear distance between them. We ran our POMRST heuristic on test cases with $0\%, 10\%, 25\%, 50\%$ and 75% of the source-sink pairs set to be critical. A very large maximum distance bound is prescribed to those non-critical source-sink pairs (i.e., there is no constraint on the maximum distance for non-critical source-sink pairs).

Table 2. POMSET results with different number of critical paths.

Size	POMRST with diff. # CSP				
n	0%	10%	25%	50%	75%
10	588	602	608	638	652
15	745	754	789	812	820
20	866	898	943	981	1051
25	978	1019	1065	1128	1191
30	1067	1121	1165	1271	1322
35	1142	1195	1257	1395	1415
40	1226	1301	1372	1445	1504

Table 3. Number of violations of maximum distance constraints.

n	Violations of maximum distance constraints											
	10%			25%			50%			75%		
	# CSP	# V	% V	# CSP	# V	% V	# CSP	# V	% V	# CSP	# V	% V
10	1	0.5	50	2	0.9	45	5	2.5	50	7	3.9	55.71
15	1	0.4	40	3	1.9	63.33	7	4.1	58.57	11	6.4	58.18
20	2	1.5	75	5	3.4	68	10	6.9	69	15	10.2	68
25	2	1.5	75	6	4.6	76.66	12	9.2	76.66	18	13.4	74.44
30	3	2.3	76.66	7	5.1	72.86	15	10.9	72.66	22	17.7	80.45
35	3	2.4	80	8	6.2	77.5	17	13.6	80	26	19.9	76.54
40	4	3.3	82.5	10	8.3	83	20	16.2	81	30	24.2	80.67

The experimental results are summarized in Tables 2 and 3. In Table 2, the second, third, fourth, fifth and sixth columns give the lengths of the rectilinear Steiner tree generated with 0%, 10%, 25%, 50% and 75% of the source-sink pairs set to be critical, respectively. Note that the POMRST problem becomes the MRST problem when there is 0% critical source-sink pairs. In Table 2, we observe only a marginal increase in the total length when we compare other columns with the column for 0% critical source-sink pairs. Since our heuristic has demonstrated a favorable performance for the MRST problem, we suspect that for randomly generated test cases, each critical source-sink can be connected (using our heuristic) via a shortest rectilinear path between the source and sink with only a relatively small increase in the total length when compared to the MRST.

We can interpret the above observation in two different ways. It may mean that our POMRST heuristic works effectively to keep the increase in the total length of the Steiner tree small. On the other hand, it may happen that our test cases are easy. That is, even if we ignore the maximum distance constraints and solve the MRST problem using our heuristic, the maximum distance constraints of only very few critical source-sink pairs will be violated in the Steiner tree obtained. Therefore, our heuristic may not need to work very hard and the increase in total length is small.

To investigate the second possibility, we conducted the following experiment. For each percentage, we also find out how many critical source-sink pairs will be at distances further than their rectilinear distances apart, if we ignore the critical source-sink pairs and just run our POMRST heuristic with no maximum distance constraints. The results are tabulated in Table 3. #CSP represents the number of critical source-sink pairs. #V is the average number of critical source-sink pairs that are at distances greater than their rectilinear distances apart. %V is the percentage of violation. The results show that the average percentage of violation is greater than 68% and the percentage of violation increases with the size of test cases to above 80% in all columns. Thus, this shows that our test cases are not easy and gives support to the favorable performance of our heuristic.

5. Conclusions

We have developed a new formulation of the rectilinear Steiner tree problem known as the performance oriented rectilinear Steiner problem that takes performance into consideration. Since the problem is NP-hard, we provided a heuristic for it. Our heuristic is able to generate solutions for nets of size $n \leq 40$ quickly. When we restrict our problem to the traditional minimum rectilinear Steiner tree problem, our heuristic is able to generate solutions that are, on the average, 10.64% better than the minimum rectilinear

spanning tree solution, for all $7 \leq n \leq 40$. When $3 \leq n \leq 6$, our heuristic generated optimal solutions for the test cases. Our experiments on POMRST problem indicate that a small increase in the total length of interconnection can improve the performance of the net tremendously. The POMRST problem is very useful in the routing of high performance circuits.

References

[1] P. Agarwal and M. Shing, "Algorithms for Special Cases of Rectilinear Steiner Trees: I. Points on the Boundary of a Rectilinear Rectangle," *Networks*, vol. 20, pp. 453–485, 1990.

[2] S.W. Cheng, A. Lim, and C.T. Wu, "Optimal Steiner Tree for Extremal Point Sets," in preparation, 1992.

[3] J. Cohoon, D. Richards, and J. Salowe, "A Linear-Time Steiner Tree Routing Algorithm For Terminals on the Boundary of a Rectangle," in *ICCAD*, pp. 402–405, 1988.

[4] J. Cong, A. Kahng, G. Robins, M. Sarrafzadeh and C.K. Wong. "Provably Good Performance-Driven Global Routing," IEEE Transactions on Computer Aided Design, 1992.

[5] W.E. Donath, R.J. Norman, B.K. Agarwal, and S.E. Bello, "Timing driven placement using complete path delays," in *Proc. IEEE Design Automation Conference*, 1990, pp. 84–89.

[6] A.E. Dunlop *et al.*, "Chip layout optimization using critical path weighting," in *Proc. IEEE Design Automation Conference*, 1984, pp. 133–136.

[7] M. Garey and D. Johnson, "The Rectilinear Steiner Tree Problem is NP-Complete," *SIAM J. Applied Math.*, no. 4, pp. 826–834, 1977.

[8] F. Hwang, "On Steiner Minimal Trees with Rectilinear Distance," *SIAM J. Applied Math.*, vol. 30, no. 1, pp. 104–114, 1976.

[9] F. Hwang and D. Richards, "Steiner Tree Problems," *Networks*, vol. 22, pp. 55–89, 1992.

[10] M.A.B. Jackson, E.S. Kuh, and M. Marek-Sadowska, "Timing driven routing for building block layout," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1987, pp. 518–519.

[11] A. Lim, "Performance Oriented Spanning Trees," manuscript, 1992.

[12] A. Lim, Y. Chee, and C.T. Wu, "Performance Driven Placement with Global Routing For Macro Cells," in *Second Great Lakes Symposium on VLSI*, 1992.

[13] S. Park and K. Miller, "Random Number Generators: Good Ones are Hard to Find," *Communications of ACM*, vol. 31, no. 10, pp. 1192–1201, 1988.

[14] S. Prasitjutrakul and W.J. Kubitz, "A timing-driven global router for custom chip design," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 48–51.

[15] D. Richards, "Fast Heuristic Algorithms for Rectilinear Steiner Trees," *Algorithmica*, vol. 4, pp. 191–207, 1989.

[16] D.S. Richard and J.S. Salowe. "A Linear-Time Algorithm to Construct a Rectilinear Steiner Minimal Tree for k-Extremal Point Sets," *Algorithmica*, 7 (1992), 247–276.

[17] M. Sarrafzadeh and C.K. Wong. "New Directions in the Rectilinear Steiner Tree Problem," manuscript (preliminary version), 1992.

[18] S. Sutanthavibul and E. Shragowitz, "An Adaptive Timing-Driven Layout for High Speed VLSI," in *27th Design Automation Conference*, pp. 90–95,

1990.

[19] C.T. Wu, A. Lim, and D. Du, "An Effective Timing-Driven Placement Algorithm For Macro Cells," in *5th International Conference in VLSI Designs*, pp. 31–35, 1992.

[20] Y. Yang and O. Wing, "An algorithm for the wiring problem," in *Digest IEEE Int. Symp. Electrical Networks*, pp. 14–15, 1971.

[21] Y. Yang and O. Wing, "Optimal and suboptimal solution algorithms for the wring problem," in *Proc. IEEE Int. Symp. Circuit Theory*, pp. 154–158, 1972.

Appendix: The Branch-and-bound Algorithm

The first step is to grid the plane in which the points in set S are located. Let W be the set of points where the grid lines intersect. Clearly, $S \subseteq W$. A POMRST can be found by choosing its Steiner points from $W - S$.

Let n be the number of sinks to be joined to the source. Each partial solution is represented by a partial tree. A partial tree is a collection of edges. The branch-and-bound algorithm will keep growing each partial tree in a breadth first manner. A breadth first search strategy is used using a FIFO queue.

Each partial tree stored in the queue is associated with two values, lb and ub. lb stores the lower bound of the length of the best Steiner tree that contains that partial tree as a subgraph. ub stores the upper bound of the length of a possible Steiner tree that contains that partial tree as a subgraph.

Two global variables are also maintained: $best_{ub}$ and $best_{cost}$. The variable $best_{ub}$ stores the smallest upper bound discovered so far. The variable $best_{cost}$ stores the smallest length of the complete Steiner trees discovered so far.

Figures 11, 12, and 13 provide the pseudo codes for our branch-and-bound algorithm. To compute lb for each partial tree, we run Prim's algorithm to find spanning tree edges to connect the remaining unconnected points to the partial tree. Then the lower bound is computed :

$$\text{Lowerbound} = \text{length of partial tree} + \frac{2}{3} \times \text{sum of lengths of spanning tree edges}$$

The upper bound can be computed by joining the remaining unconnected points to the partial tree using the POMRST heuristic with no maximum distance constraints. Then the upper bound is computed :

$$\text{Upperbound} = \text{length of partial tree} + \text{cost for joining unconnected points}$$

```
Algorithm Branch-and-Bound(S);
   /* To find Minimum Rectilinear Steiner Tree */
begin
   while queue not empty do
   begin
      repeat
         current:=dequeue();
      until current.lb ≤ best_ub and current.lb < best_cost;
      if current connects all sinks
         then
         begin
            record current;
            update best_cost and best_ub if necessary
         end
         else if current.lb < best_ub
            then
               call Expand(current);
   end
end;
```

Figure 11. Main Program of Branch-and-bound Algorithm.

```
procedure Expand(current);
begin
    Pick a leaf node in the partial tree current;
    For each possible direction in which current can grow from that leaf do
        call Try(current, leaf, direction);
    end
end;
```

Figure 12. Procedure Expand.

```
procedure Try(current, leaf, direction);
begin
    Grow current in the given direction from that leaf;
    if a cycle is created then return;
    if the new partial tree has been produced before then return;
    if the growth creates some useless turns then return;
    /* The new partial tree passed the pruning steps */
    Compute the cost of the partial tree;
    Compute the lb value for the partial tree;
    if lb is not smaller than best_cost and best_ub then return;
    Compute the ub value for the partial tree;
    Update best_cost and best_ub if needed;
    Append this partial tree to end of queue;
end;
```

Figure 13. Procedure Try.