A Novel Ultra-Fast Heuristic for VLSI CAD Steiner Trees

Bharat Krishna
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052
bharat.krishna@intel.com

C.Y. Roger Chen
Syracuse University
Dept. of Elec. & Comp. Engineering
Syracuse, NY 13244
crchen@syr.edu

Naresh K. Sehgal
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052
naresh.k.sehgal@intel.com

ABSTRACT

In all stages of VLSI chip design, routing estimation is required to account for the effect of interconnects. We propose a fast Steiner tree construction algorithm, which is 3-180 times faster for 10-300 point Steiner trees, and within 2.5% of the length of the Batched-1-Steiner tree. The proposed method can be used as a fast net length estimation tool in VLSI CAD applications, e.g. in the inner cycle of a floorplanning/placement engine.

Categories & Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids - *Layout, Placement and Routing*.

General Terms: Algorithms

Keywords: Steiner trees, Routing, Interconnect estimation

1. INTRODUCTION

In VLSI integrated circuits, electrically equivalent points are usually connected by routing the wires in Manhattan geometries, i.e. the wires are routed in horizontal and vertical directions. The most common solution to the rectilinear trees are the Steiner tree algorithms. The problem of finding a minimum total length routing tree referred to as the Minimum Rectilinear Steiner Tree (MRST) is NP-complete [1] . The shortest non-rectilinear tree is the minimum spanning tree (MST), which is the starting point for most of the Steiner algorithms that have been published.

In all stages of VLSI chip design, routing estimation is required to account for the effect of interconnects. For example, routing is estimated to calculate signal delay due to resistance and capacitance of wires for better circuit design, or to estimate the routing feasibility by analyzing congestion in the placement phase. All placement algorithms include the net cost as a key factor in their optimization algorithm. Although the available Steiner tree solutions provide a good estimate of the wire length for a net, their slow runtime makes them un-usable for many applications. Some applications use half perimeter [2] as the estimation of net lengths, which is a fast but inaccurate method when the number of points to be connected is more than 3. In this paper, we present two variations of a novel algorithm. One option of the algorithm has a runtime that is upto 180X faster and the tree length is 2.5% more than the Batched-1-Steiner algorithm [4] . The second variation has a faster runtime (250X), but the tree length is 4% more. In real circuits, it is common to have about 5% of the nets having at least 10 nodes. Our method can be used to quickly and more accurately estimate the wire length of these nets.

2. RELATED WORK

There has been extensive research and published work [3] [4] [5] [6] [9] [10] [11] on generating minimum length Steiner trees. A Steiner tree is a rectilinear tree for a given set of vertices and some additional vertices to create the minimum length spanning tree. These additional vertices are known as "Steiner points".

Various graph-based methods [3] [4] [5] [6] [9] have been proposed to find the Steiner points. They all use the earlier works of Hanan [7] to identify the Steiner points. The basic concept of these algorithms is to find the "gain in tree cost", i.e. the length of the MST with the added point is less than the tree without the added point. Some of them are the Iterative-1-Steiner (I1S) heuristic by Kahng et al [3] and the edge based heuristic [9] by Borah et. al., etc. The cost of the tree is the sum of the length of all the edges in the tree. The reduction in the tree cost is obtained by adding the Steiner points in the tree. A fast implementation of the I1S algorithm is the Batched-1-Steiner algorithm by Griffith et. al. [4] , which is widely accepted as the best heuristic Steiner tree algorithm and is thus used for comparing the results obtained from the proposed algorithm. The authors in [8] presented a hybridized Genetic Algorithm for finding near-optimal trees. A thorough survey of the various algorithms for computing optimal Steiner trees can be found in [12] .

3. PROPOSED ALGORITHM

A new ultra fast algorithm is proposed to create Steiner trees. The input to the new algorithm is a Minimum Spanning Tree (MST) on the set of vertices. For each vertex, the proposed algorithm identifies a group of edges that can have overlapping rectilinear edges, such that the tree length of the rectilinear sub-tree of these edges is smaller than the sum of their rectilinear translations. This is illustrated in **Figure 1**. The amount of overlap is defined as the gain in the length of the tree as shown. Therefore, for a tree with two edges, e_1 and e_2, if the rectilinear lengths of two edges are $L(e_1)$ and $L(e_2)$ respectively, then the tree length is $L(e_1) + L(e_2) - (L(e_1) \cap L(e_2))$, where $L(e_1) \cap L(e_2)$ is the overlap between the rectilinear connections of e_1 and e_2.

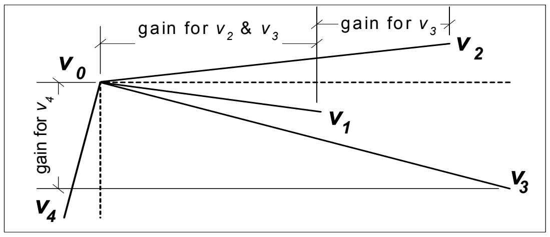


Figure 1: Gain definition

Sub-groups of edges are identified from the set of non-rectilinear edges that have non-zero gain. These groups are translated into rectilinear edges in a descending order of the gain. The two methods are based on Maximum Gain Analysis (MGA) and Favored Direction Analysis (FDA). Both methods analyze the global relative location of the points connected by the edges of the MST to create the rectilinear tree.

3.1 Maximum Gain Analysis

The MGA computes, for each vertex, the maximum overlap (i.e. gain) that can be achieved from the edges incident on the vertex. The subset of edges that would give the maximum gain is found by computing the horizontal and vertical overlaps for all edges incident on each of the vertices. To compute the gain in a given direction, consider the m vertices connected to a vertex v_0 that have an Eastern orientation. Let the $m+1$ vertices be ordered such that $x_0 < x_1 < \ldots < x_m$, where x_i is the x-coordinate of v_i. First, v_0 is connected to v_1 by a rectilinear connection such that (x_1, y_0) is the bend point in the connection. The remaining vertices, i.e. v_2 to v_m, can now be connected to v_0 by starting at (x_1, y_0), thus making use of the existing connection between v_0 and v_1. This results in reducing the length of each of the rectilinear connections between v_0 and v_i $(2 \le i \le m)$ by $(x_1 - x_0)$ for a gain of $(m-1)(x_1 - x_0)$. Next, (x_1, y_0) is connected to v_2 by the rectilinear connection with bend (x_2, y_0). The remaining vertices, i.e. v_3 to v_m can then be connected to v_0 by starting at (x_2, y_0), thus making use of the existing connection between v_0 and v_2. This will result in reducing the length of each of the rectilinear connections between v_0 and v_i $(3 \le i \le m)$ by $(x_2 - x_1)$, for an additional gain of $(m-2)(x_2 - x_1)$. This computation can be extended for all vertices such that:

$$\text{Total gain from all connections} = \sum_{i=1}^{m-1} (x_i - x_{i-1}) * (m - i)$$

Lemma 1: The gain in a given direction is the sum of the length of all edges minus the length of the longest edge.

Proof: For $m = 2$, the gain is $x_1 - x_0$. (1)

For $m = 3$, the gain is $(x_1 - x_0)(3-1) + (x_2 - x_1)(3-2) = 2x_1 - 2x_0 + x_2 - x_1$
$= x_1 - 2x_0 + x_2$
$= (x_1 - x_0) + (x_2 - x_0)$ (2)

For $m = 4$, the gain
$= (x_1-x_0)(4-1) + (x_2-x_1)(4-2) + (x_3-x_2)(4-3)$
$= 3x_1 - 3x_0 + 2x_2 - 2x_1 + x_3 - x_2 = x_1 - 3x_0 + x_2 + x_3$
$= (x_1 - x_0) + (x_2 - x_0) + (x_3 - x_0)$ (3)

Note that $(x_i - x_0)$ is the horizontal length of edge between v_0 and v_i. From (1) (2) and (3), a pattern can be observed. For 3 edges (i.e. $m = 3$), the gain is sum of the horizontal length of the shortest 2 edges. For 4 edges (i.e. $m = 4$), the gain is sum of the length of the shortest 3 edges. Therefore, for m edges, the gain is sum of the length of the shortest $(m-1)$ edges, which is equivalent to the sum of the length of all edges minus the length of the longest edge. Q.E.D.

An example of gain computation is shown in **Figure 2**. From lemma 1, it becomes unnecessary to sort the vertices based on their coordinates. Since only vertex v_1 has more than 1 edge incident on it, gains are computed for this vertex only. The example shows that the gain in East direction, EG(v_1), is due to the overlap between the connections from v_1 to v_2, v_3 and v_4. This is computed as follows:

$\text{EG}(v_1) = (x_2-x_1)+(x_3-x_1)+(x_4-x_1)-\max((x_2-x_1),(x_3-x_1),(x_4-x_1))$
$= (16-8)+(20-8)+(14-8)-\max((16-8),(20-8),(14-8))$
$= (8) + (12) + (6) - \max(8, 12, 6) = 26 - 12 = 14$

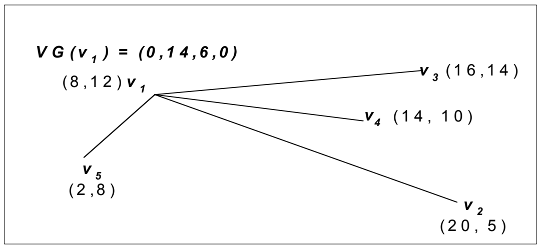


Figure 2: Example of Gain Calculation w.r.t vertex v_1

Similarly, the gain in the South direction (SG(v_1)) is 6. From this, it is determined that the maximum gain realizable for v_1 is 8 and is possible when the edges with an East direction component are converted. The edges that do not overlap with any other edge are identified as “no-impact” edges. Thus, the gain is computed for vertices with 2 or more edges. The vertex with the maximum gain is selected and all edges incident on it that have a component in the maximum gain direction are converted into rectilinear shapes. The process is repeated until all edges are converted into Manhattan edges.

3.1.1 Definitions

Let V be the set of vertices and E be the set of edges in the MST. Let e_{ij} be the edge connecting v_i and v_j, where v_i , $v_j \in V$ and $e_{ij} \in E$. The vertex gain $VG(v_i)$ is modeled as a vector of 4 elements, enumerated as the four geographical directions, i.e. {North, East, South, West}. Each element in the set is the gain in the respective direction for the vertex.

$VG(v_i)$: the gains in 4 geographical directions w.r.t. v_i. See example in **Figure 2**.

$MG(v_i) = \underset{vi \in V}{\mathrm{MAX}} \; (VG(v_i))$

$MGD(v_i)$ = the direction of $MG(v_i)$

$GMG = \underset{vi \in V}{\mathrm{MAX}} (MG(v_i))$, is the global maximum gain.

3.1.2 Pseudo Code for MGA algorithm

The first step is to compute the *MG* and *MGD* pairs for all the vertices in the given MST. The second step is to find the vertex with the global maximum gain and translate all edges that contribute to the *GMG* to rectilinear edges. The *MG* for all the vertices affected by the rectilinearization is updated by subtracting the respective contribution of the converted edges. These steps are repeated until all edges are rectilinearized. After this step completes, only the no impact edges remain as non-rectilinear and these are rectilinearized trivially. The pseudo code for the flow is given in **Figure 3**.

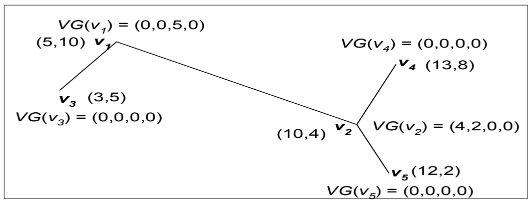
```
Proc RectilinearizeWithMGA {
    Compute MG for each vertex;
    While (non-rectilinear edges){
        Find the vertex with maximum gain //i.e. GMG
        Translate all edges which contribute to MG;
        Update MG and MGD of all affected vertices;
    }
    Convert no impact edges;
endProc;
```

Figure 3: Pseudo code for the MGA algorithm

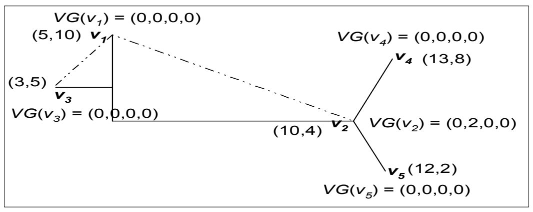
3.1.3 Runtime Complexity of MGA

A tree of n nodes has $n-1$ edges. The computation of *MG* step for all vertices involves computing the rectilinear length of $2(n-1)$ edges, which is $O(n)$. As each rectilinear edge is created just once, the time to create all the edges is also $O(n)$. To find the *MG* vertex in the first loop, the set of vertices is sorted in $O(nlogn)$. The successive loops update the VG of the affected vertices only, and the *MG* vertex can be found in O(1) from the pre-sorted list. Therefore, the overall algorithm using MGA is of $O(nlogn)$.

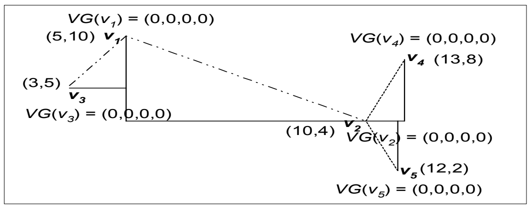
A step-by-step example describing the flow of the algorithm is illustrated in **Figure 4**. An input MST with calculated *VGs* for each of the vertices is shown in **Figure 4**(a). In the first iteration, v_1 is identified as having the global maximum gain in the Southern direction. All edges incident on v_1 and connecting to pairing vertices in the South are translated to rectilinear edges. The converted non-rectilinear edges are shown as dotted lines. The *VGs* are updated for the affected vertices (i.e. v_1, v_2 and v_3) by subtracting the contribution of the converted edges. The resulting picture is shown in **Figure 4**(b). In the next iteration, vertex v_2 is found to have the global maximum gain in the Eastern direction. Therefore, all the edges incident on v_2 and connecting to pairing vertices in the East are translated to rectilinear edges and the *VG*s are updated for the affected vertices resulting in the picture shown in **Figure 4**(c). At this point all the edges have been converted to rectilinear paths and the algorithm stops.



(a) An example with calculated VGs



(b) The results after 1st iteration



(c) The final Steiner Tree after 2nd iteration

Figure 4: A step-by-step example using MGA

3.2 Favored Direction Analysis

The Favored Direction Analysis (FDA) identifies a vertex where the maximum number of edges favors a certain direction. For a large number of points, this method is less computation intensive and is therefore faster. In FDA, the number of edges overlapping is computed, not the actual amount of overlap as in MGA.

3.2.1 Definitions

Let V be the set of vertices and E be the set of edges in the MST. Let e_{ij} be the edge connecting v_i and v_j, where v_i , $v_j \in V$ and $e_{ij} \in E$. The relative location (*RL*) of a vertex with respect to another vertex is defined as a vector of 4 elements, enumerated as the four geographical directions, i.e. {North, East, South, West}. Each element is set to 1 if the pair point is in that global location else it is 0. Therefore,

$RL(v_i, v_j)$: the relative location of v_j w.r.t. v_i. see **Figure 5**.
$RL(v_j, v_i)$: the relative location of v_i w.r.t. v_j. see **Figure 5**.
$SRL(v_i) = \sum_{e_{ij} \in E} (RL(v_i, v_j))$, is a vector of 4 elements, each indicating the number of pairing points that exist in each of the geographical locations relative to v_i. See example in **Figure 6**.
$FD(v_i) = \mathrm{MAX}_{v_i \in V} (SRL(v_i))$, is the direction favored by most of the edges incident on v_i and is referred to as the Favored Direction for v_i.

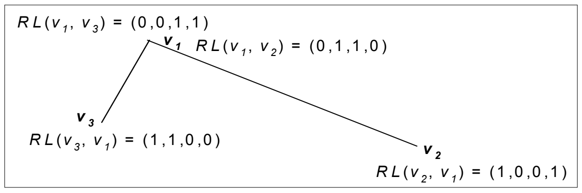


Figure 5: Example of Relative Location (*RL*)

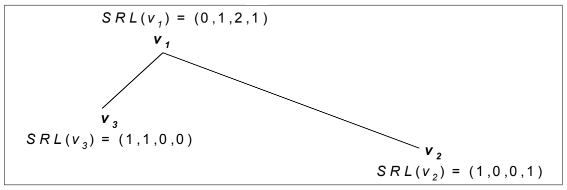


Figure 6: Ex. of Summed Relative Locations (*SRL*)

3.2.2 Pseudo Code for FDA Algorithm

The rectilinearization process is an iterative algorithm. It first computes *SRL* for all the vertices in the given MST. Next, it finds a vertex that has a favored direction and translates all the edges that contributed to the *SRL* into rectilinear connections. The *SRL* for all the vertices affected by the rectilinearization is updated by subtracting the respective contribution of the converted edges. The process is repeated until all the edges are translated. If a vertex with a favored direction cannot be found, a vertex with the maximum *SRL* (which is not unique) is picked and the edges in one of the directions are translated. The pseudo code for the flow is given in **Figure 7**.

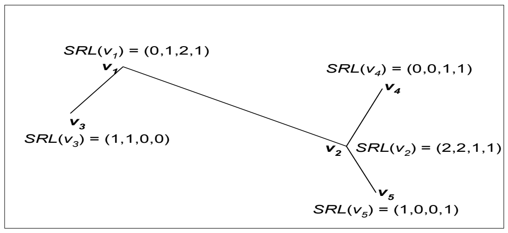
```
Proc RectilinerizeWithFDA {
    Compute SRL for each vertex;
    While (non-rectilinear edges){
        Find a vertex with FD;
        If (found){
            Translate all edges which contribute to FD;
            Update SRL of all affected vertices;
        } else {
            Find a vertex with maximum SRL;
            Translate all edges which contribute to FD;
            Update SRL of all affected vertices;
        }
    }
endProc;
```

Figure 7: Pseudo code for the FDA algorithm

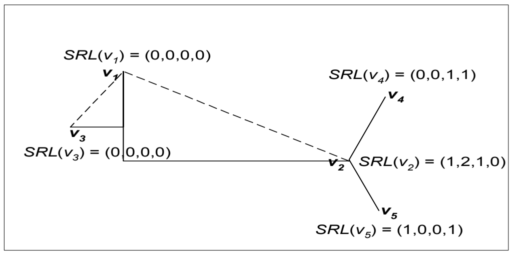
3.2.3 Runtime Complexity for FDA

A tree of n nodes has $n-1$ edges. The computation of *SRL* step for all vertices involves calculating the relative directions of $2(n-1)$ edges, which is $O(n)$. As each rectilinear edge is created just once, the time to create all the edges is also $O(n)$. To find the maximum *FD* vertex in the first loop, the set of vertices is sorted in $O(nlogn)$. The successive loops update the *SRL* of the affected vertices only, and the maximum *FD* vertex can be found in $O(1)$ from the pre-sorted list. Therefore, the overall algorithm using FDA is of $O(nlogn)$.

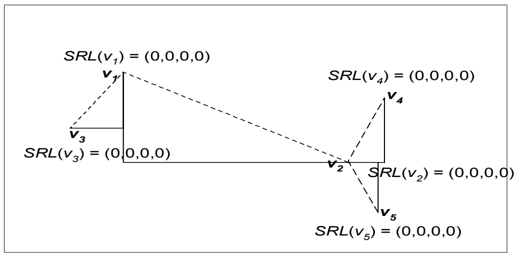
A step-by-step example describing the flow of the FDA algorithm is illustrated in **Figure 8**. An input MST with the calculated *SRL* for each of the vertices is shown in **Figure 8**(a). In the first iteration, v_1 is identified as having an *FD* of South. Therefore, all edges incident on v_1 that connect to a pairing vertex in the South are translated to rectilinear edges. The resulting picture is shown in **Figure 8**(b). The converted non-rectilinear edges are shown as dotted lines. The *SRL*s are updated for the affected vertices (i.e. v_1, v_2 and v_3) by subtracting the contribution of the converted edges.



(a) An example MST with computed SRLs



(b) The example tree after 1st iteration



(c) The final Steiner tree after 2nd iteration

Figure 8: A step-by-step example using FDA

In the next iteration, vertex v_2 is found to have an *FD* of East. All the edges incident on v_2 that connect to a pairing vertex in the East are translated to rectilinear edges and the *SRL*s are updated for the affected vertices resulting in the picture shown in **Figure 8**(c). At this point, all the edges have been converted to rectilinear paths and the algorithm stops.

4. RESULTS

Both flavors of the proposed Steiner tree algorithm, Maximum Gain Analysis (MGA) and Favored Direction Analysis (FDA), were implemented. The Steiner trees generated by the proposed algorithms are compared to the Batched-1-Steiner (B1S) [4] implementation. All experiments are run on an IBM AIX workstation. The metrics used for comparison are runtime and tree length. The data presented is calculated by averaging the runtime and tree length for 1000 randomly generated testcases in each category. The runtime results are summarized in **Table 1**.

Table 1: Runtime Comparison

Test case	No of points	B1S runtime (sec)	FDA runtime (sec)	FDA Speed Up	MGA runtime (sec)	MGA Speed Up
St1	10	0.02	0.005	4.00	0.006	3.33
St4	20	0.125	0.013	9.62	0.016	7.81
St5	40	0.897	0.038	23.61	0.05	17.94
St50	50	1.69	0.05	33.80	0.07	24.14
St100	100	13.23	0.19	69.63	0.24	55.13
St200	200	95.6	0.7	136.57	0.9	106.22
St300	300	349.8	1.4	249.86	1.9	184.11

The results show that the runtime of the proposed algorithm increases linearly with the number of points in the tree. Also, the absolute runtime of the proposed FDA algorithm is 250x faster than that of the Batched-1-Steiner. It is also seen that while both the proposed algorithms scale linearly with the number of points, the MGA has a runtime approximately 1.3x that of the FDA algorithm.

Table 2: Tree Length Comparison

Test case	No of points	B1Steiner average tree length	FDA average tree length	FDA % tree length increase	MGA average tree length	MGA % tree length increase
St1	10	2702.45	2792.23	3.32	2757.65	2.04
St4	20	3813.44	3952.85	3.66	3898.36	2.23
St5	40	5339.06	5545.49	3.87	5469.48	2.44
St50	50	5922.35	6147.13	3.80	6073.24	2.55
St100	100	8264.94	8582.78	3.85	8462.55	2.39
St200	200	11545.56	12000.84	3.94	11830.74	2.47
St300	300	14058.97	14631.66	4.07	14419.97	2.57

The tree length comparison results are shown in **Table 2**. With 1000 testcases used in each category, the average tree length is worse by about 2.5% using MGA and 4% for FDA computed trees.

The tree length comparison shows that the MGA algorithm produces trees that are closer to the B1S Steiner trees. But this comes at the cost of increased runtime, as the MGA is about 50% slower than the FDA algorithm. A chart comparing the speed up and accuracy in results is shown in **Figure 9**. It is clear that the FDA provides higher speedup but lesser accurate results compared to the MGA method.

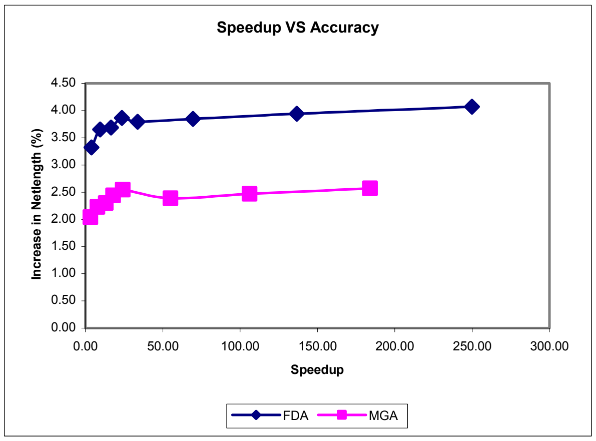


Figure 9: Speedup vs. accuracy comparison between MGA and FDA

Figure 10 shows the Steiner tree generated by the Batched-1-Steiner implementation from [4] for 200 points. The runtime for the example is 95 sec and the tree length is 11058 units. For the same set of points, **Figure 11** shows the Steiner tree generated by the MGA based algorithm proposed here. The length of this tree is 11360 units (2.7% more) but the runtime to generate it is only 0.9 sec (106x faster). **Figure 12** shows the Steiner tree generated by the FDA based algorithm proposed here. The length of this tree is 11465 units (3.6% more) but the runtime to compute it is only 0.7 sec (136x faster).

5. CONCLUSION

We presented two variations of a novel and ultra fast algorithm to generate Steiner trees. Many Steiner tree algorithms have been proposed in literature, but the existing solutions are very slow for use in VLSI CAD applications. The Maximum Gain Analysis (MGA) algorithm presented here compares well to Batched-1-Steiner implementation with the tree cost (or total edge length) being at most 2.5% more while the runtime is upto 180x faster for trees with 300 points. As the runtime of the proposed algorithm is $O(nlogn)$, the runtime speedup is dependent on the number of points connected by the tree, but the increase in average net length is independent of the number of points.

The proposed Steiner tree algorithm can be used to quickly and accurately estimate the wire length of a net in VLSI CAD applications. The fast runtime and accurate net length estimation make the algorithm very usable in a variety of intermediate physical design steps where the net lengths are estimated repeatedly in all iterations of an algorithm, e.g. placement algorithm.

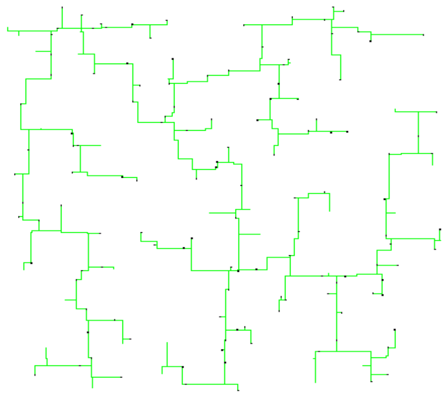


Figure 10: Steiner Tree by Batched-1-Steiner for 200 points (Runtime = 97 sec)

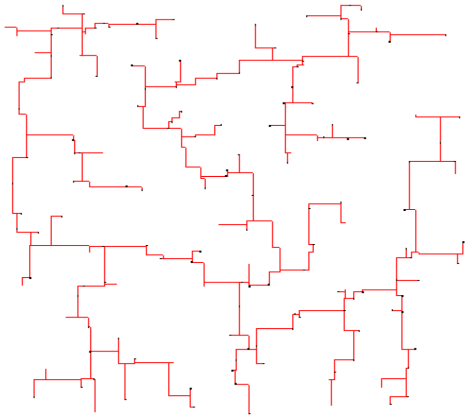


Figure 11: Steiner Tree by MGA for 200 points (Runtime = 0.9 sec)

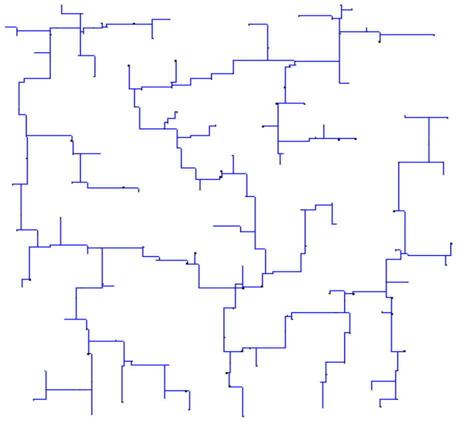


Figure 12: Steiner Tree by FDA for 200 points (Runtime = 0.7 sec)

REFERENCES

[1] Garry, M. R., and Johnson, D.S., "The Rectilinear Steiner Tree Problem is NP-complete," *SIAM Journal on Applied Mathematics*, Vol. 32, No. 4, pp 826-834, 1977.

[2] Gerez, S. H., "Algorithms for VLSI Design Automation," *John Wiley & Sons Ltd.*, pp.105, 1999.

[3] Kahng, A., Robins, G., "A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach," *IEEE International Conference on Computer-Aided Design, Digest of Technical Papers*, pp. 428-431, 1990.

[4] Griffith, J., Robins, G., Salowe, J.S., Tongtong, Z., "Closing the gap: near-optimal Steiner trees in polynomial time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.13, pp. 1351-1365, 1994.

[5] Hwang, F.K., "An O(n log n) Algorithm for Suboptimal Rectilinear Steiner trees," *IEEE Transactions on Circuits and Systems*, Vol. CAS26, pp. 75-77, Jan 1979.

[6] Ganley, J.L., Cohoon, J.P., "A faster dynamic programming algorithm for exact rectilinear Steiner minimal trees," *Proceedings of Fourth Great Lakes Symposium on VLSI,* pp. 238-241, 1994.

[7] Hanan, M., "On Steiner's problem with rectilinear distance," *SIAM Journal of Applied Mathematics*, 14, pp. 255-265, 1966.

[8] Saltouros, M., Theologou, M., Angelopoulos, M., Ricudis, C. S., "An efficient evolutionary algorithm for (near-) optimal Steiner tree calculation: an approach to routing of multipoint connections," *Proceedings of the Third International Conference on Computational Intelligence and Multimedia Applications*, pp. 448-453, 1999.

[9] Borah, M., Owens, R.M., Irwin, M.J., "A fast and simple Steiner routing heuristic [for VLSI]," *Discrete Applied Mathematics 90*, pp 51-67, Jan. 1999.

[10] Areibi, S., Xie, M., Vannelli, A., "An efficient rectilinear Steiner tree algorithm for VLSI global routing," *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, pp. 1067-1072, 2001.

[11] Mandoiu, I. I., Vazirani, V. V., Ganley, J. L., "A new heuristic for rectilinear Steiner trees," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pp. 1129-39, Oct. 2000.

[12] Ganley, J. L., "Computing optimal rectilinear Steiner trees: a survey and experimental evaluation," *Discrete Applied Mathematics 90*, pp. 161-171, Jan. 1999.