Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design

Chris Chu
Electrical and Computer Engineering
Iowa State University
Ames, IA 50010
Email: cnchu@iastate.edu

Yiu-Chung Wong
Rio Design Automation
Santa Clara, CA 95054
Email: ycwong@rio-da.com

Abstract

In this paper, we present a very fast and accurate rectilinear Steiner minimal tree (RSMT)[1] algorithm called FLUTE. The algorithm is an extension of the wirelength estimation approach by fast lookup table [1]. The main contribution of this paper is a new net breaking technique which is much better than the one in [1]. A scheme is also presented to allow users to control the tradeoff between accuracy and runtime.

FLUTE is optimal for nets up to degree 9 and is still very accurate for nets up to degree 100. So it is particularly suitable for VLSI applications in which most nets have a degree 30 or less. We show experimentally that over 18 industrial circuits in the ISPD98 benchmark suite, FLUTE with default accuracy is more accurate than the Batched 1-Steiner heuristic and is almost as fast as a very efficient implementation of Prim's rectilinear minimum spanning tree (RMST) algorithm. By adjusting the accuracy parameter, the error can be further reduced with only a small increase in runtime (e.g., $2.7\times$ error reduction with $2.2\times$ runtime increase).

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Design Aids – Placement and Routing

General Terms
Algorithms, Performance, Design

Keywords
Rectilinear Steiner minimal tree algorithm, routing, wirelength estimation

[1]A rectilinear Steiner minimal tree is a tree with minimum total edge length in Manhattan distance to connect a given set of nodes possibly through some extra (i.e., Steiner) nodes.

1 Introduction

Rectilinear Steiner minimal tree (RSMT) construction is a fundamental problem that has many applications in VLSI design. In early design stages like physical synthesis, floorplanning, interconnect planning and placement, it can be used to estimate wireload, routing congestion and interconnect delay. In global and detailed routing stages, it is used to generate the routing topology of each net.

RSMT problem is NP-complete [2]. So, in practice, rectilinear minimum spanning tree (RMST) is often used instead of RSMT. This approach is particularly common in early design stages in which the design space is being explored and hence a fast tree construction algorithm is crucial. The disadvantage of this approach is that the length of RMST may be much longer than that of RSMT since Steiner node is not allowed. Hwang [3] showed that length of RMST can be as much as 1.5 times that of RSMT. However, the difference is typically far less than 50% in practice. So this inaccuracy is tolerable in early design stages.

At later stages in which better wirelength is required, RSMT construction is necessary. Hwang et al. [4] provided a comprehensive discussion of various RSMT algorithms. For optimal RSMT algorithm, the fastest implementation is currently the GeoSteiner package [5, 6]. Griffith et al. [7] (Batched 1-Steiner heuristic) and Mandoiu et al. [8] are two well-known near-optimal algorithms. However, these optimal and near-optimal algorithms are computationally too expensive to be used in VLSI design applications. Many attempts have been made to design RSMT algorithms with lower runtime complexity. Borah et al. [9] presented an $O(n^2)$ time algorithm in which a spanning tree is iteratively improved by connecting a point to a nearby edge and deleting the longest edge on the created cycle. An $O(n \log n)$ time but very complicated alternative implementation was also proposed. Zhou [10] used spanning graph [11] to help both generating the initial spanning tree and finding good candidates for the edge substitution idea in [9]. The resulting algorithm runs in $O(n \log n)$ time, and produces better solution in slightly less runtime than the one in [9].

Most signal nets in VLSI circuits have a low degree[2]. So in VLSI applications, rather than having a low runtime complexity, it is more important for RSMT algorithms to be simple so that it can be efficient for small nets. An example of such an approach is the single trunk Steiner tree (STST), which is constructed by connecting each pin to a truck that goes either horizontally or vertically through the median position of all pins [12]. However, the length of STST is far from optimal even for medium size nets (e.g., degree 10-15). Hence its application is limited. Chen et al. [13] proposed an algorithm called Refined Single Trunk Tree (RST-T) to reduce the length of STST by a refining procedure. RST-T is proved to be optimal for nets up to degree 4 and is experimentally shown to be optimal for degree 5 nets. It is reasonably accurate for medium size nets too. RST-T runs in $O(n \log n)$ time with a fairly small constant.

In this paper, a very fast and accurate RSMT algorithm is presented. The algorithm is obtained by extending the wirelength estimation technique FLUTE [1] to perform RSMT construction. We also called the resulting RSMT algorithm FLUTE. In FLUTE, low-degree nets (up to degree 9 in our implementation) are handled optimally and efficiently by a lookup table approach. High-degree nets are recursively broken down until lookup table can be used. The main contribution of this paper is a new net breaking technique which is much better than the one in [1]. An optimal algorithm and three heuristics are proposed to collectively determine the best ways to break a net. A scheme is also presented to allow users to control the tradeoff between accuracy and runtime. The runtime complexity of FLUTE with fixed accuracy is $O(n \log n)$ for a degree n net.

Since FLUTE is extremely fast and accurate for low-degree nets, it is especially suitable for VLSI applications. We show experimentally that over 18 industrial circuits in the ISPD98 benchmark suite [14], FLUTE with default accuracy is more accurate than the Batched 1-Steiner heuristic [7] and is almost as fast as a very efficient implementation of Prim's RMST algorithm [15]. By adjusting the accuracy parameter, the error can be further reduced with only a small increase in runtime (e.g., $2.7\times$ error reduction with $2.2\times$ runtime increase). In addition, we show that even for high-degree nets (up to degree 100), it is still very fast and accurate.

The remainder of the paper is organized as follows. In Section 2, a brief review of the original FLUTE algorithm is provided. In Section 3, the extension for RSMT construction is discussed. In Section 4, the new net breaking technique is presented. In Section 5, experimental results are shown. The paper is concluded in Section 6.

2 The Original FLUTE Algorithm

FLUTE is a lookup table based technique originally designed for wirelength estimation [1]. In [1], it is shown that the set of all degree n nets can be partitioned into $n!$ groups according to the relative positions of their pins. For each group, the wirelength of all possibly optimal routing topologies along the Hanan grid [16] can be written as a small number of linear combinations of distances between adjacent Hanan grid lines. Each linear combination can be expressed as a vector of the coefficients which is called a potentially optimal wirelength vector (POWV). The few POWVs for each group can be generated once by an efficient algorithm based on a boundary compaction technique, and stored into a lookup table. To evaluate the wirelength of a net, we just need to compute the wirelengths corresponding to the POWVs for the group the net belongs to, and then report the one with minimum wirelength. To speed up the wirelength evaluation of a net (i.e., evaluation of all POWVs in a group), a minimum spanning tree based algorithm is presented to explore the similarity among the POWVs in a group.

[2]The *degree* of a net is the number of pins in the net.

Degree	# of groups	# of POWVs in a group			Ave. # of op.
n	$n!$	Min.	Ave.	Max.	per net
2	2	1	1	1	0
3	6	1	1	1	0
4	24	1	1.667	2	1.333
5	120	1	2.467	3	4.267
6	720	1	4.433	8	10.333
7	5040	1	7.932	15	20.025
8	40320	1	15.251	33	38.561
9	362880	1	30.039	79	74.155

Table 1: Number of groups, number of POWVs in a group, and average number of addition/subtraction operations to evaluate a net.

This idea works well for low-degree nets. Table 1 gives the number of groups, the number of POWVs in a group, and the average number of addition/subtraction operations to evaluate a net by the MST-based algorithm for nets with degree up to 9. For low-degree nets, the number of groups and the average number of POWVs per group are both small. Hence, the size of the lookup table is also small. The table size for all nets up to degree 9 is only 2.75 MB. In addition, it is extremely economical to evaluate a low degree net.

For high-degree nets, both the table size and the number of operations to evaluate a net will be impractically large. So in [1], a lookup table is constructed for nets with degree up to a user-defined parameter D.[3] Nets with higher degree are recursively divided into sub-nets by a net breaking technique until the lookup table can be used. The net breaking technique tries to divide the net at all pin positions and in both directions (i.e., horizontal and vertical). Then the best solution is picked. For a given pin and breaking direction, pins with a smaller coordinate than the given pin in the breaking direction form one sub-net and other pins form another sub-net. The given pin is also included into both sub-nets as shown

[3]In [1], $D = 7$ or 8 is used. In this paper, $D = 9$ is used. We have constructed a lookup table that can be proved to be optimal for degree up to 6 and experimentally verified to be optimal for degree 7, 8, and 9 by 5 million random nets each. The table construction is based on a modified boundary compaction technique but it is not reported here due to space limitation.

in Figure 1(a). However, if recursive calls are really made to evaluate each of the possible pins and directions, the runtime will be very significant. Thus, the total half-perimeter wirelength (HPWL) of the two sub-nets is used to predict the wirelength of each possibilities. For example, according to HPWL prediction, Figure 1(c) is a better selection than Figure 1(b). Then only one pin in each direction is selected to really break the net. The better of the two wirelengths will be returned.

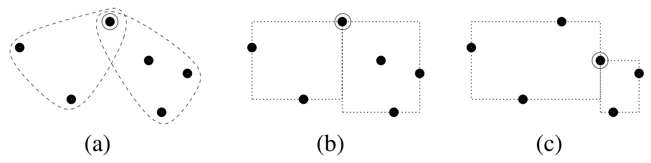

Figure 1: Illustration of the original net breaking technique.

3 Extension for RSMT Construction

In this paper, we extend the original FLUTE idea to construct RSMT. When building the lookup table, for each POWV, one of the routing topologies generated by the boundary compaction based algorithm is stored. Then to construct RSMT, the Steiner trees corresponding to the two sub-nets created by net breaking is combined and returned.

It is enough to store one routing topology per POWV. If more than one are stored, then different RSMTs can be constructed. All the resulting RSMTs have the same wirelength. However, routers may explore the alternatives to optimize some other objectives like congestion or coupling between different nets.

One of the causes of the non-optimality of the original FLUTE algorithm is that the two sub-nets are considered independently. The two corresponding Steiner sub-trees may share some wire segments as shown in Figure 2(a). However, if wirelength only is returned as in original FLUTE, this redundancy cannot be detected. In this extension that returns a RSMT, the redundant segment is detected and removed in linear time. So the wirelength can be improved and the runtime complexity will remain the same.

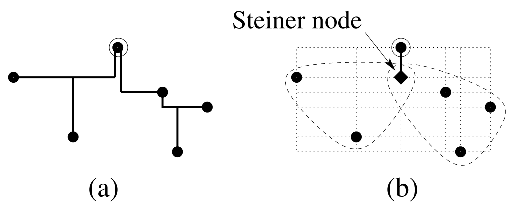

Figure 2: Redundant segment when combining two Steiner sub-trees.

4 Improved Net Breaking Technique

In this section, we present an improved technique to recursively break high-degree nets. In this technique, if a net satisfies certain conditions, it will be broken optimally. Otherwise, three heuristics are applied to collectively determine a score for each way of breaking. Then several ways corresponding to the highest scores are tried by making recursive calls. Note that in original FLUTE, one pin is selected for each direction to break the net. However, in the new technique, there is no restriction on the number of pins selected for each direction. In this technique, a scheme is also introduced to allow users to control the tradeoff between accuracy and runtime.

Let us introduce some notations. Consider an n-pin net. Let x_i be the x-coordinate of some vertical Hanan grid line such that $x_1 \le x_2 \le \cdots \le x_n$. Similarly, let y_j be the y-coordinate of some horizontal Hanan grid line such that $y_1 \le y_2 \le \cdots \le y_n$. Assume the pins are indexed in ascending order of y-coordinate. Let s_i be the rank of pin i if all pins are sorted in ascending order of x-coordinate.[4] Therefore, the coordinates of the pin i is (x_{s_i}, y_i). The notations are illustrated in Figure 3.

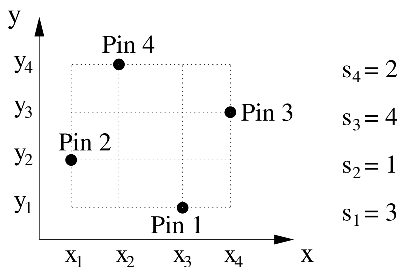

Figure 3: Illustration of some notations.

4.1 Optimal Net Breaking Algorithm

Theorem 1 *For any net, if there exists r such that $s_i \le r$ for all $i \in \{1, \ldots, r\}$ (see Figure 4(a) for an example with $r = 3$), then an optimal RSMT can be constructed by merging the optimal RSMTs of $\{Pin\ 1, \ldots, Pin\ r, (x_r, y_r)\}$ and $\{(x_r, y_r), Pin\ r + 1, \ldots, Pin\ n\}$.*

Proof: $s_i \le r$ for all $i \in \{1, \ldots, r\}$ implies $s_i \ge r + 1$ for all $i \in \{r + 1, \ldots, n\}$. In other words, the x-coordinates of the first r pins are always less than the x-coordinates of the remaining $n - r$ pins. The first r pins and the other $n - r$ pins naturally form two clusters. In any optimal RSMT, there should be at least one[5] "bridge" connecting the two clusters (Figure 4(b)). An optimal RSMT T^* that passes through the node (x_r, y_r) can be constructed by shifting the segments of each bridge without changing the wirelength (Figure 4(c)). Another RSMT T with the same or less wirelength to T^* can

[4]Note that $s_1 s_2 \ldots s_n$ is called vertical sequence in [1].

[5]It can be proved that there is always exactly one bridge.

be obtained by merging the optimal RSMTs for the two clusters with the node (x_r, y_r) added to both. Hence, T should also be optimal. □

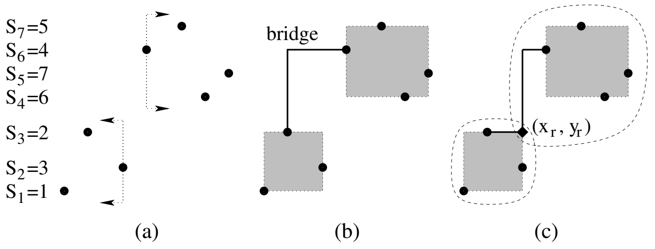

Figure 4: Illustration of the optimal net breaking algorithm.

Theorem 2 *For any net, if there exists r such that $s_i \geq n - r + 1$ for all $i \in \{1, \ldots, r\}$, then an optimal RSMT can be constructed by merging the optimal RSMTs of $\{Pin\,1, \ldots, Pin\,r, (x_{n-r+1}, y_r)\}$ and $\{(x_{n-r+1}, y_r), Pin\,r+1, \ldots, Pin\,n\}$.*

Proof: Similar to Theorem 1. □

The optimal net breaking algorithm will break a net according to Theorem 1 and 2 if there exists $r \in \{2, \ldots, n-2\}$ satisfying either one of the two conditions. Note the the size of the two sub-nets are $r+1$ and $n-r+1$. So it will not be useful to break the net if $r = 1$ or $n-1$.

4.2 Net Breaking Heuristics

Without loss of generality, consider breaking the net according to y-coordinate. If the net is broken at pin r, then pin 1 to pin r will form one sub-net, and pin r to pin n will form another sub-net. To ensure that both sub-nets are at least a constant factor smaller than the original net, we require $\gamma n \leq r \leq n - \gamma n + 1$ for some positive constant γ. We compute a score which is a weighted sum of three components:

$$\text{Score } S(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r).$$

A larger score means a more desirable way of breaking. So it is better for $S_1(r)$ to be large, and for $S_2(r)$ and $S_3(r)$ to be small.

The first component is:

$$S_1(r) = y_{r+1} - y_{r-1}$$

If we break the net at pin r, according to [1], the segment length $y_{r+1} - y_r$ (respectively, $y_r - y_{r-1}$) will be counted once in the wirelength of the upper (respectively, lower) sub-net because pin r is the only pin at the bottom (respectively, top) boundary of the sub-net. Otherwise, both $y_{r+1} - y_r$ and $y_r - y_{r-1}$ are likely to be counted more than once in the total wirelength. So it is better to break the net at pin r if $y_{r+1} - y_{r-1}$ is large.

The second component is:

$$S_2(r) = \begin{cases} 2(x_3 - x_2) & \text{if } s_r = 1 \text{ or } 2 \\ x_{s_r+1} - x_{s_r-1} & \text{if } 3 \leq s_r \leq n-2 \\ 2(x_{n-1} - x_{n-2}) & \text{if } s_r = n-1 \text{ or } n \end{cases}$$

When $3 \leq s_r \leq n-2$, x_{s_r+1} and x_{s_r-1} are the x-coordinates of the pins just right and just left of pin r, respectively. If we break the net at pin r, in both the lower sub-net and the upper sub-net, the pins on the left of pin r needs to be connected to those on the right (unless for the rare cases that there is no pin either on the left or on the right of pin r in a sub-net). So the segment lengths $x_{s_r+1} - x_{s_r}$ and $x_{s_r} - x_{s_r-1}$ will be counted in both the upper and the lower sub-nets. Therefore, it is less desirable to break the net at a pin with a large $x_{s_r+1} - x_{s_r-1}$. When $s_r = 1$ (respectively, n), pin r is at the left (respectively, right) boundary and x_{s_r-1} (respectively, x_{s_r+1}) is not defined. When $s_r = 2$ (respectively, $n-1$), as the segment length $x_2 - x_1$ (respectively, $x_n - x_{n-1}$) will always be counted once for any way of breaking according to [1], it is less effective to use $x_{s_r+1} - x_{s_r-1}$ as a prediction. For these cases, we observe that it is good in practice to set the second component to either $2(x_3 - x_2)$ or $2(x_{n-1} - x_{n-2})$.

The third component is:

$$S_3(r) = \left| s_r - \frac{n+1}{2} \right| \times d_x + \left| r - \frac{n+1}{2} \right| \times d_y$$

where $d_x = \dfrac{x_{n-1} - x_2}{n-3}$ and $d_y = \dfrac{y_{n-1} - y_2}{n-3}$. In general, it is better to have the breaking pin closer to the center of the net. If pin r is close to center vertically (i.e., r is close to $(n+1)/2$), the net will be evenly divided and hence less recursive calls are likely to be made later. Both accuracy and runtime will be improved as a result. If pin r is close to center horizontally (i.e., s_r is close to $(n+1)/2$), other pins are closer to pin r on average in both upper and lower sub-nets. In here, we use the distance of pin r from the center (in terms of number of segments in Hanan grid) to predict how many extra segments need to be used. d_x and d_y are the average segment lengths in the Hanan grid. Because $x_n - x_{n-1}$, $x_2 - x_1$, $y_n - y_{n-1}$, and $y_2 - y_1$ are always counted once for any solutions, they are not included in the computation of average length of extra segments. In principle, we can use different weights for the horizontal part and the vertical part of S_3 to form the score. However, we observe that a single weight β works just as well.

We experimentally determined that it is good to set both α and β to 0.3. S_1 is the most important of the three components. S_1 by itself is already significantly better than the HPWL heuristic in the original FLUTE. The result is even better by combining the three. We have also tried to incorporate the HPWL heuristic into the score. Even better wirelength can be obtained. However, the HPWL heuristic is relatively expensive to compute. (It can consider all pins in linear time, which is the same complexity as other heuristics, but it comes with a bigger constant.) We note that to achieve higher accuracy, it is more effective by utilizing the accuracy control scheme described below.

4.3 Accuracy Control Scheme

We can control the accuracy of FLUTE by changing the number of ways of breaking each net. However, we observe that it is not as good if all sub-nets generated by recursive calls are handled with the same accuracy. A better tradeoff between accuracy and runtime can be obtained if lower-level sub-nets are handled with less accuracy. We introduce a user-defined accuracy parameter A. The original net is handled with accuracy A. That means A different ways of breaking are tried. Then for each level of recursive call, the accuracy is reduced by 1 unless it is already 1. We notice that a small A is already enough to obtain very accurate solutions. We set the default value of A to 3.

4.4 Time Complexity of FLUTE

The time complexity is analyzed as follows. Consider $A = 1$. We first need to sort all pins according to x- and y-coordinates. Then we recursively break the net into two subnets in a roughly even manner. In each recursive call, it takes linear time to check the optimal breaking conditions and to compute the scores. So the total runtime is $O(n \log n)$. Note that the optimal net breaking algorithm may not break the net in a even manner. However, we can implement the algorithm to search for clusters simultaneously starting from all four corners (instead of only lower-left and lower-right corners as suggested by Theorem 1 and 2, respectively). Then, if the net is not broken evenly (i.e., a small cluster exists), the checking time will also be small. So the total runtime will still be $O(n \log n)$. For accuracy A, it is not hard to show by mathematical induction on A that the time complexity of FLUTE is $O(A!\, n \log n)$.

5 Experimental Results

We have implemented FLUTE in C. Our implementation has a time complexity of $O(n^2)$ because a simple $O(n^2)$ sorting algorithm is used, and the net breaking pin is searched in the range $2 \leq r \leq n-1$. The source code of FLUTE is posted in the "Rectilinear Spanning and Steiner Trees" slot of the GSRC bookshelf [17].

We perform all experiments in a 750 MHz Sun Sparc-2 machine. Three sets of experiments are conducted. First, we compare the following five algorithms on nets from industrial circuits: an efficient $O(n^2)$ implementation of Prim's algorithm (RMST) [15], Refined Single Trunk Tree (RST-T) [13], the spanning graph based RSMT algorithm (SPAN) [10], the near-optimal Batched Iterated 1-Steiner (BI1S) heuristic [7], and FLUTE with default accuracy $A = 3$. The exact RSMT software GeoSteiner 3.1 [6] is used to generate the optimal solutions. Source codes of RMST, BI1S, and GeoSteiner are downloaded from the GSRC Bookshelf [18]. Source codes of SPAN and RST-T are obtained from the authors. The 18 IBM circuits in the ISPD98 benchmark suite are used. Some information of the benchmark circuits are given in Table 2. There are totally 1.57 million nets. The placement is generated by FastPlace [19].

Circuit	# of nets	Ave. degree	Max. degree
ibm01	14111	3.58	42
ibm02	19584	4.15	134
ibm03	27401	3.41	55
ibm04	31970	3.31	46
ibm05	28446	4.44	17
ibm06	34826	3.68	35
ibm07	48117	3.65	25
ibm08	50513	4.06	75
ibm09	60902	3.65	39
ibm10	75196	3.96	41
ibm11	81454	3.45	24
ibm12	77240	4.11	28
ibm13	99666	3.58	24
ibm14	152772	3.58	33
ibm15	186608	3.84	36
ibm16	190048	4.10	40
ibm17	189581	4.54	36
ibm18	201920	4.06	66
All	1570355	3.92	134

Table 2: Benchmark information.

The wirelength comparison is shown in Table 3. FLUTE is the best among the five algorithms. The average wirelength error over all nets is only 0.07%. FLUTE produces the best wirelength for all 15 circuits in which all nets have degree 55 or less. BI1S is the best for the remaining three circuits (ibm02, ibm08 and ibm18).

The breakdown of the wirelength estimation for nets with different degree is shown in Table 4. A summary of all 18 circuits is given. Columns 2 and 3 provide a breakdown on the number of nets and the wirelength. Notice that although most nets are of degree two or three, there are still a substantial proportion of higher degree nets and the contribution of those nets to the wirelength is very significant. For example, nets with degree 10 or more account for 8.13% of all nets and contribute 26.2% of total wirelength. Columns 4 to 8 report the percentage error in wirelength. As the table shows, all five techniques have more error for nets with higher degree. FLUTE is exact for nets up to degree 9 and is still very accurate for higher degree nets. Note that although RST-T is exact up to degree 5, it performs badly for high-degree nets. As a result, the overall accuracy is far worse than the other three RSMT algorithms.

The runtime comparison is listed in Table 5. Note that except for FLUTE, the more accurate algorithms require significantly more runtime. FLUTE is only 2.4 times slower than RMST (the fastest) but is the most accurate.

Second, we show the effect of the accuracy parameter A to the tradeoff between wirelength error and runtime. A is varying from 1 to 9. An implementation of FLUTE with

Degree	Net breakdown		Wirelength error (%)				
	#	WL	RMST	RST-T	SPAN	BI1S	FLUTE
2	54.92%	27.98%	0.00	0.00	0.00	0.00	0.00
3	14.40%	10.26%	2.50	0.00	0.04	0.00	0.00
4	7.68%	7.84%	3.89	0.00	0.09	0.00	0.00
5	5.61%	8.18%	4.74	0.00	0.30	0.05	0.00
6	3.20%	5.65%	5.40	0.49	0.33	0.07	0.00
7	2.28%	4.82%	5.91	1.02	0.46	0.09	0.00
8	1.98%	4.61%	6.25	1.90	0.44	0.11	0.00
9	1.81%	4.46%	6.79	2.64	0.52	0.15	0.00
10-17	6.98%	21.72%	7.81	6.22	0.66	0.22	0.16
$\geq$18	1.15%	4.48%	9.04	14.05	0.75	0.32	0.77

Table 4: Breakdown of the wirelength estimation according to degree for nets of all 18 circuits.

Circuit	Wirelength error (%)				
	RMST	RST-T	SPAN	BI1S	FLUTE
ibm01	4.092	1.942	0.258	0.098	0.076
ibm02	5.849	3.750	0.335	0.117	0.221
ibm03	4.637	1.925	0.267	0.099	0.060
ibm04	4.048	1.270	0.207	0.061	0.050
ibm05	4.489	3.155	0.330	0.111	0.086
ibm06	5.964	2.846	0.378	0.137	0.090
ibm07	4.720	1.693	0.266	0.087	0.042
ibm08	4.784	4.446	0.325	0.119	0.250
ibm09	4.331	1.813	0.236	0.075	0.039
ibm10	4.104	1.787	0.253	0.077	0.052
ibm11	4.018	1.215	0.218	0.062	0.026
ibm12	3.783	1.912	0.246	0.074	0.056
ibm13	4.782	2.001	0.293	0.106	0.049
ibm14	3.908	1.541	0.220	0.069	0.036
ibm15	4.201	1.945	0.265	0.076	0.060
ibm16	4.231	2.426	0.278	0.089	0.061
ibm17	3.905	2.189	0.265	0.080	0.052
ibm18	4.432	3.352	0.298	0.098	0.133
All	4.232	2.263	0.269	0.085	0.070

Table 3: Percentage error in wirelength.

Circuit	Runtime (s)				
	RMST	RST-T	SPAN	BI1S	FLUTE
ibm01	0.03	0.55	3.68	72.48	0.06
ibm02	0.06	0.78	7.17	108.93	0.15
ibm03	0.05	1.05	7.04	140.00	0.11
ibm04	0.06	1.22	7.29	162.15	0.09
ibm05	0.08	1.15	11.95	146.23	0.22
ibm06	0.06	1.38	9.94	176.88	0.16
ibm07	0.09	1.94	13.76	244.50	0.20
ibm08	0.14	2.07	18.58	266.02	0.41
ibm09	0.12	2.44	17.14	308.61	0.23
ibm10	0.16	3.00	26.00	383.44	0.39
ibm11	0.14	3.17	20.08	411.29	0.23
ibm12	0.18	3.07	28.61	394.68	0.44
ibm13	0.19	3.88	28.37	504.71	0.38
ibm14	0.29	6.00	44.21	775.54	0.60
ibm15	0.40	7.37	63.72	949.99	0.95
ibm16	0.43	7.57	77.27	968.36	1.03
ibm17	0.50	7.71	92.85	973.79	1.35
ibm18	0.49	8.03	79.96	1036.36	1.31
All	0.42	7.51	67.1	965.1	1

Table 5: Runtime comparison. The overall runtimes in the last row are normalized with respect to FLUTE runtime.

Algorithm		WL error	Runtime	
		(%)	(s)	Normalized
New FLUTE (return RSMT)	$A = 1$	0.330	5.04	0.61
	$A = 2$	0.151	6.16	0.74
	$\mathbf{A = 3}$	**0.070**	**8.31**	**1**
	$A = 4$	0.039	12.10	1.46
	$A = 5$	0.026	18.36	2.21
	$A = 6$	0.020	29.60	3.56
	$A = 7$	0.016	51.38	6.18
	$A = 8$	0.013	96.35	11.59
	$A = 9$	0.012	190.67	22.94
New FLUTE (no RSMT)	$A = 1$	0.388	2.70	0.32
	$A = 2$	0.184	3.30	0.40
	$A = 3$	0.089	4.26	0.51
	$A = 4$	0.052	6.13	0.74
	$A = 5$	0.036	8.97	1.08
	$A = 6$	0.028	14.15	1.70
	$A = 7$	0.023	23.85	2.87
	$A = 8$	0.020	43.91	5.28
	$A = 9$	0.017	85.02	10.23
Orig. FLUTE	$D = 9$	0.477	3.81	0.46

Table 6: Wirelength error and runtime of FLUTE for different accuracy A. The row in bold is the default.

RSMT construction disabled (i.e., for wirelength estimation only) and the original FLUTE with $D = 9$ are also compared. The average percentage error and total runtime for all nets in 18 IBM circuits are reported in Table 6 and plotted in Figure 5.

Table 6 and Figure 5 show that the accuracy control scheme provides a very effective way to achieve much less error in a moderate runtime increase. The runtime is increasing at a rate much slower than $A!$ because most nets have a low degree.

By comparing the two implementations of FLUTE for the same A, we notice that the runtime is roughly doubled due to RSMT construction. However, because of the removal of redundant segments as described in Section 3, the error is reduced. For applications in which only wirelength estimation is required, the implementation without RSMT construction provides a much better tradeoff between accuracy and runtime unless extremely accurate solutions are desired. For extremely accurate solutions, the implementation with RSMT

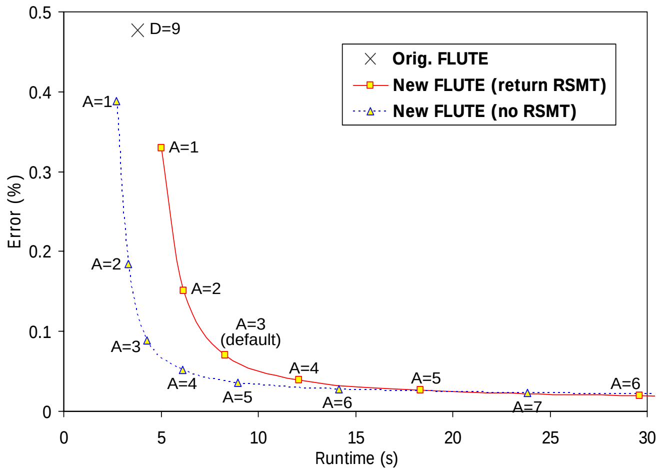

Figure 5: Wirelength error versus runtime for different A.

construction is more efficient even if the RSMT returned is not used.

Table 6 and Figure 5 also clearly show that the new net breaking technique introduced in this paper is much better than the original one. The new FLUTE without RSMT construction and with $A = 2$ is essentially the same as the original FLUTE except for the net breaking technique. The new FLUTE is 2.6× smaller in error and 13.4% less in runtime.

Third, we investigate the accuracy and runtime of different RSMT algorithms for nets with degree ranging from 10 to 100. We notice that out of 1.57 millions nets in 18 IBM circuits, only 1212 (0.077%) have a degree of more than 30, and only 80 (0.005%) have a degree of more than 60. So for VLSI applications, it should be enough to observe the behavior of algorithms for degree up to 100. 10000 nets are randomly generated for each degree. The average wirelength error and total runtime are reported in Table 7 and Table 8, respectively.

We can see from Table 7 that wirelength error is increasing sub-linearly over degree in all algorithms. For RMST, SPAN and BI1S, the error increases very slowly with respect to degree. Hence, they should be suitable for problems with large degree. FLUTE is very accurate for low-degree nets and it is still reasonably accurate for nets with degree up to 100. So it is suitable for VLSI applications. The error of RST-T increases very rapidly with respect to degree and is very substantial even for medium size nets.

From Table 8, the runtime of FLUTE with a small A value is comparable to RMST even for large nets. So it should also be suitable for applications that require a fast and moderately accurate solution for large nets. Theoretically, the time complexity is $O(n^3)$ for BI1S, $O(n^2)$ for RMST and FLUTE with fixed accuracy[6], and $O(n \log n)$ for RST-T and SPAN. But in terms of runtime scalability in the range considered, RST-T is the best. All other algorithms scale similarly. Note that for FLUTE with a large A, the runtime seems to be increasing dramatically when the degree is small. The reason is for low-degree nets, only a few level of recursive calls can be made during net breaking. Hence, the accuracy factor on runtime is not apparent. When the degree gets larger, the accuracy factor will become apparent and stabilized. Then the increase in runtime will slow down (to quadratic in our implementation).

[6] $O(n \log n)$ time implementation exists for both RMST and FLUTE.

6 Conclusion and Discussion

In this paper, we presented an extension of the wirelength estimation technique FLUTE to perform RSMT construction. A much better net breaking technique is proposed. The net breaking technique consists of an optimal net breaking algorithm and three net breaking heuristics. The intuitions behind the three heuristics are explained. However, to a certain extent, the terms and the parameters are determined experimentally. We do not have very strong arguments to fully justify them. Many other heuristics and different ways of tuning the parameters have also explored. The algorithm can be made either faster or more accurate. However, what we present in this paper provides the best tradeoff between runtime and accuracy.

Acknowledgment

We thank Prof. C. K. Cheng for providing the source code of RST-T and Prof. Hai Zhou for providing the source code of SPAN.

References

[1] Chris Chu. FLUTE: Fast lookup table based wirelength estimation technique. In *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, pages 696–701, 2004.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, NY, 1979.

[3] F. K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM Journal of Applied Mathematics*, 30:104–114, 1976.

[4] F. K. Hwang, D. S. Richards, and P. Winter. The Steiner tree problem. *Annals of Discrete Mathematics*, 1992. Elsevier Science Publishers.

[5] D. M. Warme, P. Winter, and M. Zachariasen. Exact algorithms for plane Steiner tree problems: A computational study. In D.Z. Du, J.M. Smith, and J.H. Rubinstein, editors, *Advances in Steiner Trees*, pages 81–116. Kluwer Academic Publishers, 2000.

[6] GeoSteiner – software for computing Steiner trees. `http://www.diku.dk/geosteiner/`.

Degree	Wirelength error (%)									
					FLUTE					
	RMST	RST-T	SPAN	BI1S	$A=1$	$A=2$	$A=3$	$A=4$	$A=5$	$A=6$
10	11.920	5.016	0.950	0.381	1.212	0.405	0.201	0.126	0.092	0.070
20	12.373	14.471	1.124	0.484	2.719	1.695	0.916	0.572	0.408	0.321
30	12.526	22.215	1.235	0.533	3.690	2.660	1.735	1.100	0.807	0.650
40	12.630	29.130	1.316	0.551	4.361	3.443	2.492	1.633	1.185	0.971
50	12.742	35.598	1.391	0.563	4.907	4.032	3.072	2.081	1.540	1.251
60	12.766	41.832	1.436	0.566	5.382	4.520	3.558	2.475	1.814	1.473
70	12.802	47.781	1.497	0.575	5.745	4.919	3.957	2.837	2.095	1.689
80	12.872	53.967	1.554	0.588	6.122	5.321	4.369	3.195	2.357	1.912
90	12.892	59.430	1.600	0.587	6.438	5.639	4.674	3.508	2.624	2.103
100	12.881	64.722	1.658	0.590	6.691	5.926	4.960	3.798	2.852	2.272

Table 7: Percentage error in wirelength for nets of different degree.

Degree	Runtime (s)									
					FLUTE					
	RMST	RST-T	SPAN	BI1S	$A=1$	$A=2$	$A=3$	$A=4$	$A=5$	$A=6$
10	0.07	0.48	13.69	53.76	0.13	0.17	0.19	0.23	0.27	0.30
20	0.20	0.61	38.63	76.03	0.38	0.58	1.06	1.90	3.29	5.47
30	0.41	0.74	69.32	135.03	0.68	1.08	2.09	4.70	10.19	20.89
40	0.69	0.89	109.08	242.22	1.09	1.58	3.21	8.26	20.47	46.85
50	1.05	1.04	156.53	419.09	1.34	2.17	4.41	11.81	32.28	82.49
60	1.48	1.19	217.18	680.37	1.68	2.71	5.77	15.56	46.82	130.41
70	1.99	1.34	287.67	1033.26	2.14	3.24	7.01	19.79	61.44	183.66
80	2.58	1.48	373.74	1503.89	2.49	3.91	8.45	23.90	77.01	243.95
90	3.27	1.67	493.00	2109.24	3.10	4.62	9.88	28.34	94.93	311.15
100	4.00	1.79	577.59	2839.78	3.46	5.12	11.33	32.81	112.94	384.18

Table 8: Total runtime for 10000 nets of different degree.

[7] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang. Closing the gap: Near-optimal Steiner trees in polynomial time. *IEEE Trans. Computer-Aided Design*, 13(11):1351–1365, November 1994.

[8] I. I. Mandoiu, V. V. Vazirani, and J. L. Ganley. A new heuristic for rectilinear Steiner trees. In *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, 1999.

[9] M. Borah, R. M. Owens, and M. J. Irwin. An edge-based heuristic for Steiner routing. 13(12):1563–1568, December 1994.

[10] Hai Zhou. Efficient Steiner tree construction based on spanning graphs. In *Proc. Intl. Symp. on Physical Design*, pages 152–157, 2003.

[11] H. Zhou, N. Shenoy, and W. Nicholls. Efficient spanning tree construction with delaney triangulation. *Information Processing Letters*, 81(5), 2002.

[12] J. Soukup. Circuit layout. *Proceedings of IEEE*, 69:1281–1304, October 1981.

[13] H. Chen, C. Qiao, F. Zhou, and C.-K. Cheng. Refined single trunk tree: A rectilinear Steiner tree generator for interconnect prediction. In *Proc. ACM Intl. Workshop on System Level Interconnect Prediction*, pages 85–89, 2002.

[14] C. J. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proc. Intl. Symp. on Physical Design*, pages 80–85, 1998. `http://vlsicad.cs.ucla.edu/~cheese/ispd98.html`.

[15] Andrew B. Kahng and Ion Mandoiu. RMST-Pack: Rectilinear minimum spanning tree algorithms. `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST/`.

[16] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, 14:255–265, 1966.

[17] Chris Chu. FLUTE: Fast lookup table based technique for RSMT construction and wirelength estimation. `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/`.

[18] A. E. Caldwell, A. B. Kahng, and I. L. Markov. VLSI CAD Bookshelf. `http://www.gigascale.org/bookshelf/`.

[19] Natarajan Viswanathan and Chris Chu. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proc. Intl. Symp. on Physical Design*, pages 26–33, 2004.