FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design

Chris Chu and Yiu-Chung Wong

***Abstract*—In this paper, we present a very fast and accurate rectilinear Steiner minimal tree (RSMT) algorithm called fast lookup table estimation (FLUTE). FLUTE is based on a precomputed lookup table to make RSMT construction very fast and very accurate for low-degree[1] nets. For high-degree nets, a net-breaking technique is proposed to reduce the net size until the table can be used. A scheme is also presented to allow users to control the tradeoff between accuracy and runtime. FLUTE is optimal for low-degree nets (up to degree 9 in our current implementation) and is still very accurate for nets up to degree 100. Therefore, it is particularly suitable for very large scale integration applications in which most nets have a degree of 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 algorithm.**

***Index Terms*—Interconnect optimization, rectilinear Steiner minimal tree (RSMT) algorithm, routing, wirelength estimation, wirelength minimization.**

I. Introduction

A RECTILINEAR Steiner minimal tree (RSMT) 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. RSMT construction is a fundamental problem that has many applications in very large scale integration (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 [1]. Therefore, 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 [2] showed that the 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. Therefore, this inaccuracy is tolerable in early design stages.

At later stages in which better wirelength is required, RSMT construction is necessary. Hwang *et al.* [3] provided a comprehensive discussion of various RSMT algorithms. For optimal RSMT algorithm, the fastest implementation is currently the GeoSteiner package [4], [5]. Griffith *et al.* [6] (Batched 1-Steiner heuristic) and Mandoiu *et al.* [7] 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.* [8] presented an $O(n^2)$ time algorithm in which a spanning tree is iteratively improved by connecting a point to a nearby edge and by deleting the longest edge on the created cycle. An $O(n \log n)$ time but very complicated alternative implementation was also proposed. Zhou [9] used spanning graph [10] to help in both generating the initial spanning tree and finding good candidates for the edge-substitution idea in [8]. The resulting algorithm runs in $O(n \log n)$ time and produces a better solution in a slightly less runtime than the one in [8]. Kahng *et al.* [11] gave a practical $O(n \log^2 n)$ heuristic called batched greedy algorithm (BGA) based on a batched version of the greedy triple-contraction algorithm. This algorithm produces a better solution quality and requires a slightly shorter runtime than [8] and [9] in practice.

Most signal nets in VLSI circuits have a low degree. Therefore, in VLSI applications, rather than having a low runtime complexity, it is more important for RSMT algorithms to be simple so that they 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 trunk 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, we present a very fast and accurate lookup table based RSMT algorithm called fast lookup table estimation (FLUTE). We show that the set of all degree-n nets can be partitioned into $n!$ groups according to the relative positions

Manuscript received September 21, 2006; revised March 3, 2007. This paper was recommended by Associate Editor C. J. Alpert.

C. Chu is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010 USA (e-mail: cnchu@iastate.edu).

Y.-C. Wong resides in Palo Alto, CA 94305 USA (e-mail: ycwong@cs.usc.edu).

Digital Object Identifier 10.1109/TCAD.2007.907068

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

0278-0070/$25.00 © 2008 IEEE

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

of their pins. For each group, the optimal wirelength of any net can be found based on a few vectors called potentially optimal wirelength vectors (POWVs). Each POWV corresponds to a linear combination of distances between adjacent pins. We precompute the few POWVs for each group and store them into a table. We also store one corresponding Steiner tree, which we called potentially optimal Steiner tree (POST) associated with each POWV. To find the optimal RSMT of a net, we just need to compute the wirelengths corresponding to the POWVs for the group where the net belongs and, then, return the POST associated with the POWV with minimum wirelength. This lookup table idea can optimally and efficiently handle low-degree nets (up to degree 9 in our implementation). For high-degree nets, we proposed a net-breaking technique to recursively break a net until the table can be used. A scheme is also presented to allow users to control the tradeoff between accuracy and runtime during net breaking. The runtime complexity of FLUTE with fixed accuracy is $O(n \log n)$ for a net of degree n.

Since FLUTE is extremely fast and accurate for low-degree nets, it is particularly 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 [6] 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., 3.1 times of error reduction with 2.0 times of 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 II, we present the lookup table idea to find RSMTs for low-degree nets. In Section III, we describe the algorithm to generate the POWVs and the POSTs. In Section IV, we show how the lookup table size can be reduced. In Section V, we derive a very efficient technique to evaluate all the POWVs for a given net. In Section VI, we present the net-breaking technique for high-degree nets. In Section VII, we show the experimental results. The paper is concluded in Section VIII.

II. Lookup Table Approach for Low-Degree Nets

We define a net of degree n to be a set of n pins. In this paper, we only consider Steiner trees along the Hanan grid as Hanan [16] pointed out that an optimal RSMT can always be constructed based on the Hanan grid. Given a net, the Hanan grid is formed by drawing one horizontal line and one vertical line through each pin. Let x_i be the x-coordinate of the ith 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 the jth horizontal Hanan-grid line such that $y_1 \le y_2 \le \cdots \le y_n$. Assume that the pins are indexed in ascending order of the y-coordinate. Let s_i be the rank of pin i if all pins are sorted in ascending order of the x-coordinate (ties are broken arbitrarily for both x- and y-coordinates). Therefore, the coordinates of pin i are (x_{s_i}, y_i). The notations are shown in Fig. 1. $s_1 s_2 \ldots s_n$ is called the position sequence of the net. For the net in Fig. 1, its position sequence is 3142. The position sequence completely specifies the relative positions of the pins.

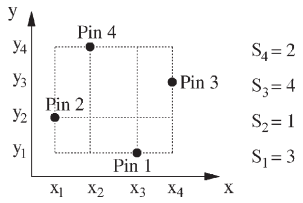


Fig. 1. Illustration of some notations.

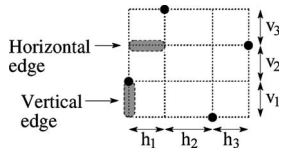


Fig. 2. Illustration of horizontal- and vertical-edge lengths.

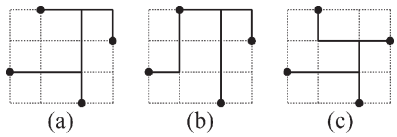


Fig. 3. Three possible Steiner trees for the net in Fig. 1.

Note that the length of a horizontal (respectively, vertical) edge in the Hanan grid is equal to the distance between two adjacent vertical (respectively, horizontal) Hanan-grid lines. We denote the horizontal-edge length as $h_i = x_{i+1} - x_i$ and the vertical-edge length as $v_i = y_{i+1} - y_i$ for $1 \le i \le n-1$. These definitions are shown in Fig. 2.

A Steiner tree on the Hanan grid can be decomposed into a collection of Hanan-grid edges. Therefore, the wirelength of any Steiner tree can always be written as a linear combination of edge lengths such that all coefficients are positive integers. For example, for the net in Fig. 1, the wirelength of the three possible Steiner trees shown in Fig. 3(a)–(c) can be written as $h_1 + 2h_2 + h_3 + v_1 + v_2 + 2v_3$, $h_1 + h_2 + h_3 + v_1 + 2v_2 + 3v_3$, and $h_1 + 2h_2 + h_3 + v_1 + v_2 + v_3$, respectively. For simplicity, we will express a wirelength as a vector of the coefficients and call it a wirelength vector. For the Steiner trees in Fig. 3(a)–(c), the wirelength vectors are (1, 2, 1, 1, 1, 2), (1, 1, 1, 1, 2, 3), and (1, 2, 1, 1, 1, 1), respectively.

In order to find the optimal wirelength for a given net, we can enumerate all possible wirelength vectors. Note that, although the number of the possible Steiner trees is huge, the number of the possible wirelength vectors is much less. More importantly, we notice that not all the wirelength vectors have the potential to produce the optimal wirelength. Most vectors are redundant because they have a larger or equal value than that of another vector in all coefficients. For example, we can ignore the wirelength vector (1, 2, 1, 1, 1, 2) because the wirelength produced by the vector (1, 2, 1, 1, 1, 1) is always v_3 less. We called a vector that can potentially produce the optimal wirelength (i.e., cannot be ignored) a POWV. We observe that for every low-degree net, there are only a few POWVs. For example, for all degree-3 nets, the only optimal wirelength vector is (1, 1, 1, 1), which corresponds to the half-perimeter wirelength (HPWL). For the net in Fig. 1, the only two POWVs

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

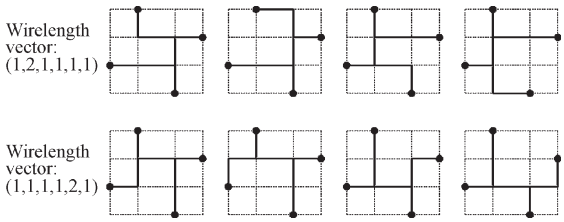


Fig. 4. All POSTs for the net in Fig. 1.

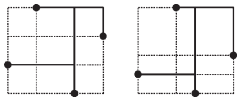


Fig. 5. Topologically equivalent Steiner trees for two different nets.

are (1, 2, 1, 1, 1, 1) and (1, 1, 1, 1, 2, 1). Which one is optimal depends on which of h_2 and v_2 is smaller. All possible Steiner trees corresponding to these two wirelength vectors are shown in Fig. 4. Each of these trees is called a POST. Some statistics on the number of POWVs will be given later in Table I.

If all the POWVs and the corresponding POSTs are precomputed and stored in a lookup table, the RSMT will be easy to find. However, the number of different nets is infinite as the pin coordinates can take different infinite values. To handle this problem, we try to group together the nets which can share the same set of POWVs. To see which nets can be grouped together, we first introduce the following definition. Two Steiner trees for two different nets are said to be topologically equivalent if they can be transformed to each other by changing the edge lengths (or equivalently, the distance between adjacent Hanan-grid lines), with the restriction that their values remain positive. This concept is shown in Fig. 5.

Lemma 1: If two nets have the same position sequence, then every Steiner tree of one net is topologically equivalent to a Steiner tree of the other net.

Proof: Suppose we shift the grid lines of the two Hanan grids for two nets so that they become identical. Since they have the same position sequence, the pins of the two nets are in the same locations in the Hanan grid. Therefore, every Steiner tree of one net will also be a Steiner tree of the other. ■

Theorem 1: The set of all degree-n nets can be divided into $n!$ groups according to the position sequence such that all nets in each group share the same set of POWVs.

Proof: Observe that the wirelengths of topologically equivalent Steiner trees can be expressed by the same wirelength vector. For example, the wirelengths of the two trees in Fig. 5 can both be represented by (1, 2, 1, 1, 1, 2), although the values of h_i's and v_i's are different for the two nets. Based on this observation and Lemma 1, nets with the same position sequence can be grouped together to share the set of POWVs. Since the position sequence of a degree-n net is a permutation of $1, 2, \ldots, n$, there should be $n!$ groups. ■

Our RSMT approach precomputes the set of POWVs associated with each group and one[2] POST associated with each POWV. The POWVs and POSTs are stored in a lookup table. To compute the RSMT for a given net, we find out the position sequence of the net and then obtain the vectors for the corresponding group from the table. Each vector generates a wirelength by summing up the product of the vector entries with h_i's and v_i's. The minimum value over all vectors gives the optimal wirelength. The POST corresponding to the vector with the minimum wirelength gives the RSMT.

[2]In general, more than one POST can be stored. Then, different RSMTs of the same wirelength can be constructed. Routers may explore the alternatives to optimize some other objectives like congestion or timing.

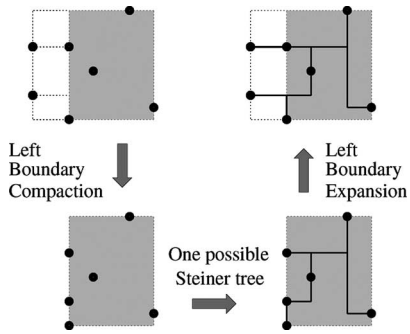


Fig. 6. Illustration of left boundary compaction.

III. Generation of Lookup Table

In this section, we discuss the generation of the sets of POWVs and the associated POSTs. For each small net degree and for each group (i.e., position sequence), we may generate all possible Steiner trees on the Hanan grid, find the corresponding wirelength vectors, and prune away the redundant ones. The remaining vectors and trees are the POWVs and POSTs for the group. A trivial approach to generate all possible Steiner trees is to enumerate all possible combinations of using and not using each edge in the Hanan grid and check if the resulting subgraph is a Steiner tree covering all the pins. However, this approach is extremely expensive. Even for degree 5, we need to enumerate a Hanan grid consisting of 40 edges for each of the 120 groups.

We propose a much more efficient algorithm based on a boundary-compaction technique. For a given group, the boundary-compaction technique reduces the grid size by compacting one of the four boundaries, i.e., shifting all pins on a boundary to the grid line adjacent to that boundary. The set of Steiner trees of the original problem can be generated by expanding the Steiner trees of the reduced grid back to the original grid. Fig. 6 uses the compaction of the left boundary as an example to illustrate the idea. Note that, in Section II, we assume that each Hanan-grid line is associated with only one pin so that the concept of position sequence is well defined. This assumption is not necessary unless we consider the grouping problem of a net. In this section, we assume that a grid line may contain more than one pin so that the grid lines can be combined and the grid size can be reduced by boundary compaction.

We can route a net by performing boundary compaction and expansion recursively. By compacting the four boundaries in a different order, a set of different Steiner trees with different wirelength vectors can be generated. Since we are performing the routing in a restricted way, it is possible that some Steiner trees and, hence, some wirelength vectors will not be generated. We define a grid G to be compactable if, for each POWV V

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

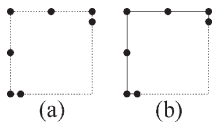


Fig. 7. Example of noncompactable grid.

Algorithm Gen-LUT(G)
Input: A grid G with some pins at grid nodes
Output: One POST for each POWV of the group associated with G
begin
1. If G is simple enough,
2. generate and return the set of POSTs for G
3. else if any boundary b contains only one pin,
4. return Expand-b(Gen-LUT(Compact-b(G)))
5. else if there is a corner with one pin such that
6. both its adjacent boundaries $b1$ and $b2$ have one other pin,
7. return Prune(Expand-$b1$(Gen-LUT(Compact-$b1$(G)))
8. $\cup$ Expand-$b2$(Gen-LUT(Compact-$b2$(G))))
9. else
10. if there are 7 pins with all 7 pins on boundaries,
11. S = {Trees with near-ring structure connecting all pins}
12. else if there are ≥ 8 pins with ≥ 7 pins on boundaries,
13. S = Connect-adj-pins(G, d) where d = # of pins $- 3$
14. return Prune(S $\cup$ Expand-left(Gen-LUT(Compact-left(G)))
15. $\cup$ Expand-right(Gen-LUT(Compact-right(G)))
16. $\cup$ Expand-top(Gen-LUT(Compact-top(G)))
17. $\cup$ Expand-bot(Gen-LUT(Compact-bot(G))))
end

Fig. 8. POST-generation algorithm for a given group. For $b \in \{\text{left}, \text{right}, \text{top}, \text{bottom}\}$, Compact-$b$() and Expand-$b$() perform compaction and expansion of boundary b, respectively. Prune() performs pruning of redundant trees not corresponding to POWVs. Connect-adj-pins() is used to generate extra trees not producible by boundary compaction.

of G, there exists a boundary b such that V can be generated by expanding some POWVs of the reduced grid obtained by compacting G at b. In other words, we can always reduce the size of a compactable grid without worrying about missing some POWVs. Lemmas 2, 3, and 4 in the succeeding parts of this paper give several situations that show that a grid is compactable. The proofs of the lemmas are in Appendix I. An example of noncompactable grid is shown in Fig. 7(a). Fig. 7(b) shows the optimal Steiner tree, which cannot be generated by boundary compaction.

Lemma 2: A grid G is compactable if it has a boundary with only one pin.

Lemma 3: A grid G is compactable if it has a corner with one pin P and if both boundaries adjacent to P have exactly one other pin.

Lemma 4: A grid G is compactable if it has up to six pins at the four boundaries.

The algorithm to generate one POST for each POWV in a given group is shown in Fig. 8. With the POSTs, the corresponding POWVs can be easily computed. Instead of enumerating all the Steiner trees first and pruning the redundant ones (i.e., those that do not correspond to POWVs) at the end, we prune the redundant trees for each subproblem. By performing pruning as early as possible, the efficiency of the algorithm can be significantly improved.

In steps 1–2, we directly generate the POSTs when G consists of a single (horizontal or vertical) grid line or is a 2×2 grid. Steps 3–4 are based on Lemma 2, and steps 5–8 are based on Lemma 3. Note that the proofs of these lemmas actually identify which boundaries to compact without missing any POWV. Since one or two (instead of four) recursive calls are made and these cases occur frequently for low-degree nets, the runtime of the algorithm can be dramatically reduced. If Lemmas 2 and 3 cannot be applied, we try compacting all four boundaries in steps 14–17. Lemma 4 guarantees that for nets with up to six pins, all the POWVs will be generated.

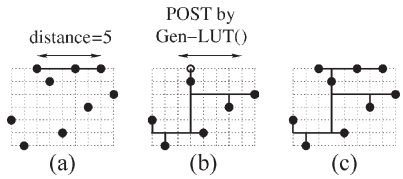


Fig. 9. Illustration for Connect-adj-pins(G, d) with $d \geq 5$.

For grids with seven or more pins, some POWVs may be missed by boundary compaction. Therefore, some extra Steiner trees are included in steps 10–13. In step 11, there are seven trees in S. Each tree is a near-ring structure, which is the bounding box that surrounds the grid with edges connecting one of the seven pairs of the adjacent pins removed. Lemma 5 proves that boundary compaction together with the near-ring structures is sufficient to generate all the POWVs for degree-7 nets. The proof of Lemma 5 is in Appendix I.

Lemma 5: For a grid with seven pins, boundary compaction together with the near-ring structures can generate all the POWVs.

For nets with eight or more pins, we used the function Connect-adj-pins() to generate some extra trees. Connect-adj-pins(G, d) connects two or more adjacent pins on the same boundary by introducing a branch along the boundary. Those pins can be at a distance at most d grid lines away from each other [see Fig. 9(a) for an illustration]. Then, those pins are replaced by a pseudo-pin located somewhere on the branch. For each possible location of the pseudo-pin, Gen-LUT() is recursively called to generate the POSTs of the reduced grid [as shown in Fig. 9(b)]. The POSTs of G can be constructed by combining the branch with the POSTs of all the reduced grids [see Fig. 9(c)].

Note that this technique is complementary to boundary compaction. It produces tree branches along a boundary that cannot be produced by boundary compaction. Lemma 6 proves that boundary compaction together with Connect-adj-pins() is sufficient to generate all the POWVs for nets with a degree of up to 10. The proof of Lemma 6 is in Appendix I.

Lemma 6: For a net with n pins where $7 \leq n \leq 10$, boundary compaction together with Connect-adj-pins() with distance $d = n - 3$ can generate all the POWVs.

Note that Connect-adj-pins() can also be used to handle nets with seven pins. However, Connect-adj-pins() is very slow because one recursive call to Gen-LUT() is made for each possible location of the pseudo-pin. Thus, the near-ring structure is used instead.

The completeness of the algorithm Gen-LUT() is summarized in the following theorem.

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

TABLE I
NUMBER OF POWVS IN A GROUP FOR NETS OF A GIVEN DEGREE

Degree n	# of groups $n!$	# of POWVs in a group		
		Min.	Ave.	Max.
2	2	1	1	1
3	6	1	1	1
4	24	1	1.667	2
5	120	1	2.467	3
6	720	1	4.433	8
7	5040	1	7.932	15
8	40320	1	15.251	33
9	362880	1	30.039	79

Theorem 2: The algorithm Gen-LUT() generates one POST for each POWV for nets with a degree of 10 or less.

Proof: This theorem follows directly from Lemmas 4, 5, and 6. ■

The number of POWVs in a group is listed in Table I. We only generate the lookup table up to degree 9. The computation time for lookup table generation will be discussed at the end of Section IV as it is affected by the table-size reduction techniques presented in Section IV.

IV. REDUCTION OF LOOKUP TABLE SIZE

According to Table I, for degree 9 alone, there are 10.9 million POWVs. If 1 byte is used to store each of the 16 entries in a POWV, the POWV storage requirement for degree 9 will be 166.3 MB. The POST associated with each POWV should have up to seven Steiner nodes and $9 + 7 - 1 = 15$ branches. If 1 byte is used to store each branch in a POST, the POST storage requirement for degree 9 will be 155.9 MB. The total storage requirement for both POWVs and POSTs and for all the degrees up to 9 will be prohibitively large.

A smaller table will reduce the usage of the hard disk, main memory, and cache. It will also reduce the time of loading the lookup table from the hard disk to the memory. Therefore, it is desirable to reduce the size of the lookup table.

One technique to reduce the POWV storage requirement is to explore the similarity among POWVs in a group and to store the differences between the POWVs according to the minimum spanning tree (MST) computed in Section V. For this method, instead of using $2 \times (d-1)$ byte for each POWV of degree d, we only need 2.5 byte or less, as shown in Table III. However, this method does not reduce the number of POWVs or the POST storage requirement.

Another technique is to explore the equivalence of different groups and to show that the POWVs and POSTs of only a small fraction of all the groups need to be generated and stored. Note that the table-generation time will also be reduced by this technique.

Groups are equivalent for two reasons. First, observe that even though the nets in Fig. 10(a) and (b) belong to two different groups, both will become the grid in Fig. 10(c) if the top boundary is compacted. Note that, by Lemma 2, both grids are compactable at the top boundary. Hence, the two groups for these nets have the same set of POWVs. Moreover, even the POSTs can be shared between the groups. For example, POSTs corresponding to the POWV (1, 2, 1, 1, 1, 1) for the nets in Fig. 10(a) and (b) are shown in Fig. 10(d) and (e), respectively.

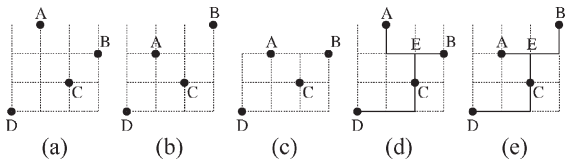


Fig. 10. Equivalence of different groups due to boundary compaction.

It is clear that both POSTs have the same topology (consisting of branches AE, BE, EC, and CD). The same argument can be applied to all four boundaries. Therefore, up to $2^4 = 16$ different groups can share a set of POWVs and POSTs (the number of equivalent groups may be less than 16 because pins can be shared by adjacent boundaries, and therefore, not all combinations exist). Second, if two nets are symmetrical horizontally, vertically, or diagonally, the POWVs and POSTs of one group can be transformed to those of the other. Due to the overhead in solution transformation, only horizontal symmetry is considered in our implementation. This allows two groups to share the POWVs and POSTs.

Some implementation details are described in the succeeding part of the paper. For any group of degree n such that the corresponding position sequence is $s_1 s_2 \ldots s_n$, we define a modified position sequence $p_1 p_2 \ldots p_n$ as follows:

$$p_i = \left|\{s_j : 1 \le j < i \text{ and } s_j < s_i\}\right|, \qquad \text{for } 1 \le i \le n.$$

For the example in Fig. 1, $p_1 p_2 p_3 p_4 = 0021$. According to the definition given previously, it is not hard to see that p_i can take any integral value between 0 and $i-1$. We define a group index for the group as

$$k = \prod_{j=1}^{n} \frac{n!}{j!} \times p_j.$$

We prove in Lemma 7 that group index can be used as the array index for the lookup table organized as an array of $n!$ groups. Then, we prove in Lemma 8 that it is sufficient for the lookup table to be an array only for the first $n!/4$ groups. The proofs of both lemmas are in Appendix II.

Lemma 7: Group index is a one-to-one mapping from the groups of degree n to an integral value between 0 and $n! - 1$.

Lemma 8: Any group of degree n is equivalent to a group with a group index between 0 and $n!/4 - 1$.

Some statistics of the lookup table are listed in Table II. We generate the lookup table up to degree 9. By exploring the equivalence of the groups, we can reduce the number of groups generated and stored by a factor of 25.8 (the table-generation time should also be reduced by a similar factor). The total table size is only 9.00 MB, which can be easily handled by today's computers.

The last column of Table II is the lookup table generation time in a PC with a 3.4-GHz Pentium 4 processor. It is extremely fast to generate the table up to degree 7 because of the boundary-compaction technique and the near-ring structure presented in Section III. For degrees 8 and 9, the generation time is much longer because of the function Connect-adj-pins(). Note that we have several ideas to significantly reduce the table-generation time (e.g., storing the solutions of a grid instead

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

TABLE II
SOME STATISTICS OF THE LOOKUP TABLE

Degree	# of groups			Table size (MB)		Gen.
n	$n!$	generated	$n!$/gen.	POWV	POST	time
2	2	1	2	0.00	0.00	0.0 s
3	6	1	6	0.00	0.00	0.0 s
4	24	2	12	0.00	0.00	0.0 s
5	120	8	15	0.00	0.00	0.0 s
6	720	36	20	0.00	0.00	0.0 s
7	5040	222	22.70	0.01	0.02	0.0 s
8	40320	1638	24.62	0.17	0.31	50.7 s
9	362880	13950	26.01	2.56	5.93	58.2 hr
Total	409112	15858	25.80	2.75	6.26	58.2 hr

TABLE III
AVERAGE NUMBER OF ADDITION/SUBTRACTION REQUIRED

Degree	Average # of ADD/SUB			
	per group		per POWV	
n	Independent	MST	Independent	MST
2	0	0	0	0
3	0	0	0	0
4	1.333	1.333	0.8	0.8
5	4.267	4.267	1.73	1.73
6	14.422	10.333	3.253	2.331
7	39.651	20.025	4.999	2.525
8	109.136	38.561	7.156	2.528
9	288.060	74.155	9.590	2.469

of recomputing them so that they can be reused in different recursive calls). However, as the lookup table only needs to be generated once, we did not implement those ideas.

V. SPEEDUP OF MINIMUM-WIRELENGTH COMPUTATION

To find the optimal RSMT of a given net, we need to consider the set of POWVs in the corresponding group. A straightforward approach is to evaluate the POWVs independently. For each POWV $(\alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \beta_1, \beta_2, \ldots, \beta_{n-1})$, we compute the expression $\text{WL} = \sum_{i=1}^{n-1} \alpha_i h_i + \sum_{i=1}^{n-1} \beta_i v_i$. Since entries in POWVs are typically small integers and addition is computationally much less expensive than multiplication, it is more efficient to add the edge length several times instead of using multiplication. In addition, each of the edge lengths should be used at least once. Therefore, it is better to evaluate the expression as $\text{WL} = \text{HPWL} + \sum_{i=1}^{n-1}(\alpha_i - 1)h_i + \sum_{i=1}^{n-1}(\beta_i - 1)v_i$. Then, we have $2(n-1)$ less terms to add.

However, we observe that most POWVs shared by a group of nets are very similar to one another. Many of them differ from others in only one or two entries. Hence, some POWVs can be efficiently evaluated by adding or subtracting some terms from some other previously computed POWVs. By exploring the dependence among the POWVs, the evaluation of all the POWVs for a net can be made more efficient than the independent approach.

The problem of determining the best dependence among POWVs for a given group can be transformed into an MST problem. Consider a group associated with a set of q POWVs. We construct a complete graph with $q+1$ nodes. q of these nodes correspond to the q POWVs in the set, and one more node corresponds to the wirelength vector $(1, \ldots, 1, 1, \ldots, 1)$ (i.e., HPWL). The weight of each edge is set to one-norm of the difference of the two corresponding wirelength vectors. In other words, the edge weight is equal to the number of addition/subtraction required to convert from the wirelength of one vector to that of the other. Given an MST of the graph, we can evaluate the POWVs in an order defined by a breath-first traversal of the tree starting from the node corresponding to the HPWL. The total edge weight of the MST gives the number of addition/subtraction required to compute all q POWVs.

The average number of addition/subtraction required for the independent and MST-based approaches is listed in Table III. Columns 2 and 3 give the average number per group, which is proportional to the average runtime to evaluate a net. It is clear that the MST-based approach can significantly speed up the evaluation of high-degree nets. The last two columns give the average number per POWV, which is proportional to the average runtime to compute a POWV. It shows that, for the independent approach, a lot more entries need to be added for POWVs of high-degree nets, whereas, for the MST-based approach, the number of entries to be added/subtracted first increases slowly with net degree and then remains around 2.5.

VI. NET BREAKING FOR HIGH-DEGREE NETS

For high-degree nets, both the table size and the number of operations to evaluate a net will be impractically large. Therefore, the table lookup approach is practical only for low-degree nets.

In FLUTE, we have a user-defined parameter D. A lookup table is constructed up to degree D ($D = 9$ in the current implementation). Nets with a degree higher than D are broken into several subnets with a degree ranging from 2 to D to which the table lookup estimation can be applied.

In this section, we present a technique to recursively break high-degree nets. In this technique, if a net satisfies certain conditions, it will be broken optimally. Otherwise, four 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. In this technique, a scheme is also introduced to allow users to control the tradeoff between accuracy and runtime.

A. Optimal Net-Breaking Algorithm

Theorem 3: For any net, if the set of pins can be partitioned into two subsets $L = \{\text{Pin } 1, \ldots, \text{Pin } r\}$ and $R = \{\text{Pin } r+1, \ldots, \text{Pin } n\}$ such that the x-coordinate of any pin in L is less than or equal to that of any pin in R [see Fig. 11(a) for an example with $r = 3$], then an optimal RSMT can be constructed by merging the optimal RSMTs of $L \cup \{(x_r, y_r)\}$ and $\{(x_r, y_r)\} \cup R$.

Proof: In any optimal RSMT, there should be at least one[3] "bridge" connecting the two subsets [Fig. 11(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 [Fig. 11(c)]. Another RSMT T with the same or less wirelength to T^* can be obtained by merging

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

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

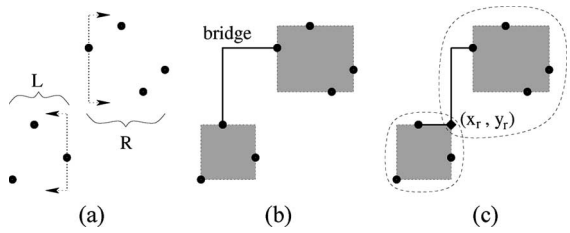


Fig. 11. Illustration of the optimal net-breaking algorithm.

the optimal RSMTs for the two subsets with the node (x_r, y_r) added to both. Hence, T should also be optimal. ■

Theorem 4: 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 $\{\text{Pin } 1, \ldots, \text{Pin } r, (x_{n-r+1}, y_r)\}$ and $\{(x_{n-r+1}, y_r), \text{Pin } r + 1, \ldots, \text{Pin } n\}$.

Proof: Similar to Theorem 3. ■

The optimal net-breaking algorithm will break a net according to Theorems 3 and 4 if there exists $r \in \{2, \ldots, n-2\}$ satisfying either one of the two conditions. Note that the sizes of the two subnets are $r + 1$ and $n - r + 1$. Therefore, it will not be useful to break the net if $r = 1$ or $n - 1$.

B. Net-Breaking Heuristics

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

$$\text{Score } S(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r).$$

A larger score means a more desirable way of breaking. Therefore, it is better for $S_1(r)$ to be large and for $S_2(r)$, $S_3(r)$, and $S_4(r)$ to be small.

The first component is

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

If we break the net at pin r, pin r will become the only pin at the bottom (respectively, top) boundary of the upper (respectively, lower) subnet. According to Lemma 2, the edge length $y_{r+1} - y_r$ (respectively, $y_r - y_{r-1}$) will be counted once in the wirelength of the upper (respectively, lower) subnet. 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. Therefore, 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 and upper subnets, the pins on the left of pin r need to be connected to those on the right (unless in the rare cases that there is no pin either on the left or on the right of pin r in a subnet). Therefore, the edge lengths $x_{s_r+1} - x_{s_r}$ and $x_{s_r} - x_{s_r-1}$ will be counted in both the upper and lower subnets. 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 edge length $x_2 - x_1$ (respectively, $x_n - x_{n-1}$) will always be counted once for any way of breaking according to Lemma 2, 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

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

where $\overline{h} = (x_{n-1} - x_2)/(n-3)$, and $\overline{v} = (y_{n-1} - y_2)/(n-3)$. In general, it is better to have the breaking pin closer to the center of the net. If pin r is close to the 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 the center horizontally (i.e., s_r is close to $(n+1)/2$), the other pins are closer to pin r on average in both the upper and lower subnets. In here, we use the distance of pin r from the center (in terms of number of edges in Hanan grid) to predict how many extra edges need to be used. $\overline{h}$ and $\overline{v}$ are the average edge 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 the average length of extra edges. In principle, we can use different weights for the horizontal and vertical parts of S_3 to form the score. However, we observe that a single weight β works just as well.

The fourth component $S_4(r)$ is the total HPWL of the two subnets. This is a direct way to predict the resulting wirelength.

We experimentally determined that it is good to set α to 0.3, β to $7.4/(n+10)$, and γ to $4.8/(n-1)$. S_1 is the most important of the four components. It produces significantly better results with the single term S_1 than with any one of the other three. The result is even better by combining all four.

After subtrees for the two subnets are constructed, they are combined to form a Steiner tree for the original net. Note that the two subtrees may share some edges, as shown in Fig. 12(a). These redundant edges can be detected in constant time and will be removed by introducing an extra Steiner node, as shown in Fig. 12(b). To further reduce the wirelength, a local refinement technique can be applied to improve the subtree in the neighborhood of the breaking pin. This technique uses FLUTE to reconstruct the subtree connecting all the pins that are directly reachable from the breaking pin without passing through the other pins [as shown in Fig. 12(c)]. To minimize the runtime overhead, the local refinement technique is applied only if the subtree around the breaking pin has up to D pins.

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

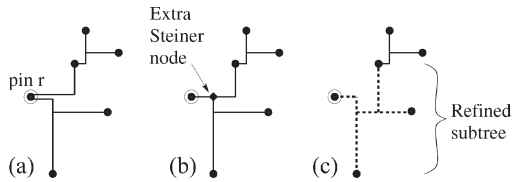


Fig. 12. Merging two Steiner subtrees.

C. 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 subnets generated by recursive calls are handled with the same accuracy. A better tradeoff between accuracy and runtime can be obtained if lower level subnets are handled with less accuracy. We introduce a user-defined accuracy parameter A. The original net is handled with accuracy A. That means that A different ways of breaking are tried. Then, for each recursive call, the accuracy is set to $\max\{\lfloor A/2 \rfloor, 1\}$. We notice that a small A is already enough to obtain very accurate solutions. We set the default value of A to 3.

D. Time Complexity of FLUTE

The time complexity is analyzed as follows. Consider $A = 1$. We first need to sort all the pins according to the 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. Therefore, the total runtime is $O(n \log n)$. Note that the optimal net-breaking algorithm may not break the net in an even manner. However, we can implement the algorithm to search for clusters simultaneously starting from all four corners (instead of only the lower left and lower right corners as suggested by Theorems 3 and 4, respectively). Then, if the net is not broken evenly (i.e., a small cluster exists), the checking time will also be small. Therefore, 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 $O(A^{(\log A+1)/2} n \log n)$.

VII. Experimental Results

The FLUTE algorithm described in this paper is implemented with the C programming language in the software package FLUTE-2.5. For our implementation, the runtime complexity is $O(n^2)$ because a simple $O(n^2)$-sorting algorithm is used, and the net-breaking pin is searched in the range $3 \leq r \leq n-2$. To minimize runtime, the local refinement technique introduced in Section VI-B is not applied for low accuracy (i.e., when $A \leq 4$). The source code of FLUTE is posted in the "rectilinear spanning and Steiner trees" slot of the Gigascale Systems Research Center (GSRC) bookshelf [17].

We perform all experiments in a 3.4-GHz Intel Pentium 4 machine.[4] Three sets of experiments are conducted. First, we compare the following six algorithms on nets from industrial circuits: an efficient $O(n^2)$ implementation of Prim's algorithm (RMST) [15], RST-T [13], the spanning graph-based RSMT algorithm (SPAN) [9], BGA [11], the near-optimal Batched Iterated 1-Steiner (BI1S) heuristic [6], and FLUTE with default accuracy $A = 3$. The exact RSMT software GeoSteiner 3.1 [5] is used to generate the optimal solutions. Source codes of RMST, BGA, BI1S, and GeoSteiner are downloaded from the GSRC bookshelf [20]. 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 on the benchmark circuits is given in Table IV. There are 1.57 million nets in total. The placement is generated by FastPlace [21].

TABLE IV
Benchmark Information

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 V
Percentage Error in Wirelength

Circuit	Wirelength error (%)					
	RMST	RST-T	SPAN	BGA	BI1S	FLUTE
ibm01	4.092	1.933	0.251	0.129	0.106	0.074
ibm02	5.849	3.780	0.331	0.143	0.115	0.209
ibm03	4.637	1.919	0.271	0.125	0.095	0.062
ibm04	4.048	1.255	0.203	0.084	0.060	0.051
ibm05	4.489	3.134	0.329	0.153	0.112	0.106
ibm06	5.964	2.822	0.381	0.182	0.134	0.084
ibm07	4.720	1.704	0.268	0.116	0.084	0.046
ibm08	4.784	4.445	0.328	0.162	0.123	0.261
ibm09	4.331	1.804	0.235	0.105	0.075	0.042
ibm10	4.104	1.790	0.252	0.104	0.080	0.051
ibm11	4.018	1.227	0.219	0.087	0.062	0.024
ibm12	3.783	1.908	0.248	0.106	0.077	0.054
ibm13	4.782	2.002	0.292	0.135	0.102	0.053
ibm14	3.908	1.540	0.221	0.095	0.068	0.040
ibm15	4.201	1.941	0.266	0.106	0.077	0.062
ibm16	4.231	2.421	0.279	0.124	0.090	0.068
ibm17	3.905	2.188	0.263	0.110	0.082	0.056
ibm18	4.432	3.353	0.300	0.134	0.100	0.147
All	4.232	2.261	0.269	0.117	0.086	0.075

The wirelength comparison is shown in Table V. FLUTE is the best among the six algorithms. The average wirelength error over all the nets is only 0.075%. FLUTE produces the best wirelength for all the 15 circuits in which all nets have a degree of 55 or less. BI1S is the best for the remaining three circuits (ibm02, ibm08, and ibm18).

The breakdown of the wirelength estimation for the nets with a different degree is shown in Table VI. A summary of all the

[4]In earlier versions of this paper [18], [19], experiments are performed in a Sun Sparc-2 machine. For unknown reasons, BI1S is significantly slower in Sun machines.

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
BREAKDOWN OF THE WIRELENGTH ESTIMATION ACCORDING TO DEGREE FOR NETS OF ALL 18 CIRCUITS

Degree	Net breakdown		Wirelength error (%)					
	#	WL	RMST	RST-T	SPAN	BGA	BI1S	FLUTE
2	54.92%	27.98%	0.00	0.00	0.00	0.00	0.00	0.00
3	14.40%	10.26%	2.50	0.00	0.03	0.00	0.00	0.00
4	7.68%	7.84%	3.89	0.00	0.11	0.00	0.00	0.00
5	5.61%	8.18%	4.74	0.00	0.21	0.07	0.05	0.00
6	3.20%	5.65%	5.40	0.49	0.29	0.12	0.07	0.00
7	2.28%	4.82%	5.91	1.02	0.37	0.13	0.09	0.00
8	1.98%	4.61%	6.25	1.91	0.42	0.16	0.12	0.00
9	1.81%	4.46%	6.79	2.65	0.48	0.21	0.15	0.00
10–17	6.98%	21.72%	7.81	6.21	0.60	0.29	0.22	0.16
≥18	1.15%	4.48%	9.04	14.05	0.75	0.40	0.32	0.87

TABLE VII
RUNTIME COMPARISON. THE OVERALL RUNTIMES IN THE LAST ROW ARE NORMALIZED WITH RESPECT TO FLUTE RUNTIME

Circuit	Runtime (s)					
	RMST	RST-T	SPAN	BGA	BI1S	FLUTE
ibm01	0.02	0.09	0.55	0.75	1.01	0.02
ibm02	0.02	0.14	1.05	1.50	4.32	0.03
ibm03	0.02	0.18	1.02	1.38	1.95	0.03
ibm04	0.04	0.20	1.07	1.44	2.24	0.02
ibm05	0.03	0.20	1.71	2.40	2.69	0.05
ibm06	0.03	0.23	1.45	1.95	2.53	0.04
ibm07	0.05	0.32	1.96	2.59	3.26	0.04
ibm08	0.06	0.35	2.63	3.74	6.60	0.09
ibm09	0.07	0.40	2.42	3.19	4.13	0.06
ibm10	0.08	0.53	3.59	4.77	5.85	0.09
ibm11	0.06	0.53	2.87	3.76	5.16	0.05
ibm12	0.10	0.54	3.94	5.33	6.25	0.10
ibm13	0.10	0.66	3.89	5.18	6.68	0.09
ibm14	0.15	1.02	5.91	7.84	10.11	0.14
ibm15	0.21	1.27	8.18	10.86	13.96	0.22
ibm16	0.23	1.33	9.33	12.47	14.75	0.26
ibm17	0.28	1.39	11.06	15.06	16.63	0.31
ibm18	0.26	1.40	9.81	13.28	17.82	0.30
All	0.93	5.56	37.34	50.25	64.92	1.0

TABLE VIII
WIRELENGTH ERROR AND RUNTIME OF FLUTE FOR DIFFERENT ACCURACY A. THE ROW IN BOLD IS THE DEFAULT

Algorithm		WL error (%)	Runtime (s)	Runtime Normalized
FLUTE (return RSMT)	$A=1$	0.2313	1.27	0.65
	$A=2$	0.1092	1.60	0.82
	$A=3$	**0.0747**	**1.94**	**1.00**
	$A=4$	0.0396	2.65	1.37
	$A=5$	0.0243	3.88	2.00
	$A=6$	0.0174	5.22	2.69
	$A=7$	0.0154	5.93	3.06
	$A=8$	0.0113	8.70	4.48
	$A=9$	0.0104	9.63	4.96
	$A=10$	0.0090	12.29	6.34
	$A=11$	0.0086	13.39	6.90
	$A=12$	0.0073	19.07	9.83
FLUTE (no RSMT)	$A=1$	0.2721	0.98	0.51
	$A=2$	0.1318	1.16	0.60
	$A=3$	0.0917	1.37	0.71
	$A=4$	0.0513	1.84	0.95
	$A=5$	0.0430	2.08	1.07
	$A=6$	0.0322	2.69	1.39
	$A=7$	0.0292	3.02	1.56
	$A=8$	0.0222	4.27	2.20
	$A=9$	0.0209	4.76	2.45
	$A=10$	0.0186	5.90	3.04
	$A=11$	0.0178	6.42	3.31
	$A=12$	0.0157	8.98	4.63
HPWL		-8.7710	0.33	0.17

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 2 or 3, there is still a substantial proportion of higher degree nets, and the contribution of those nets to the wirelength is very significant. For example, nets with a degree of 10 or more account for 8.13% of all nets and contribute 26.2% of the total wirelength. Columns 4–9 report the percentage error in wirelength. As the table shows, all six techniques have more errors for nets with a 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 four RSMT algorithms.

The runtime comparison is listed in Table VII. Note that FLUTE is much faster than all the other Steiner-tree algorithms, although it is the most accurate. The FLUTE is only 7% slower than RMST.

Second, we show the effect of the accuracy parameter A to the tradeoff between wirelength error and runtime. A is varying from 1 to 12. A potential application of FLUTE is wirelength estimation. Therefore, an implementation of FLUTE with RSMT construction disabled and the widely used HPWL is also compared. The average percentage error and total runtime for all the nets in 18 IBM circuits are reported in Table VIII.

Table VIII shows 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 $A^{(\log A+1)/2}$ because most nets have a low degree. We notice that if RSMT is not constructed, the runtime is decreased by roughly 1.3–2.1 times. However, because the redundant edge removal and the local refinement techniques described at the end of Section VI-B cannot be used, the error is increased. For applications in which only wirelength estimation is required, such an implementation provides a much better tradeoff between accuracy and runtime unless extremely accurate solutions are desired. For extremely accurate solutions, the implementation with RSMT construction is more efficient even if the RSMT returned is not used.

Even with RSMT construction and a relatively high accuracy of $A=3$, FLUTE is only 5.88 times slower than HPWL while much more accurate. If RSMT is not required and if an accuracy of $A=1$ is sufficient, FLUTE is less than three times slower than HPWL.

Third, we investigate the accuracy and runtime of different algorithms for nets with a degree ranging from 10 to 100. We notice that out of 1.57 million nets in 18 IBM circuits, only 1212 (0.077%) have a degree of more than 30, and only 80

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

TABLE IX
PERCENTAGE ERROR IN WIRELENGTH FOR NETS OF A DIFFERENT DEGREE

	Wirelength error (%)												
						FLUTE							
Degree	RMST	RST-T	SPAN	BGA	BI1S	$A=1$	$A=2$	$A=3$	$A=4$	$A=6$	$A=8$	$A=10$	$A=12$
10	11.982	5.091	0.949	0.443	0.349	0.684	0.236	0.112	0.072	0.027	0.020	0.020	0.020
20	12.168	14.370	1.019	0.518	0.421	2.181	1.265	0.961	0.590	0.281	0.150	0.119	0.098
30	12.551	21.896	1.136	0.619	0.552	2.992	2.171	1.846	1.161	0.642	0.430	0.357	0.292
40	12.727	28.987	1.121	0.624	0.556	3.516	2.718	2.388	1.709	1.096	0.751	0.670	0.554
50	12.684	35.346	1.143	0.628	0.567	3.955	3.214	2.867	2.193	1.475	1.044	0.931	0.766
60	12.729	42.110	1.192	0.647	0.580	4.288	3.571	3.252	2.557	1.839	1.280	1.160	0.971
70	12.848	47.984	1.148	0.630	0.557	4.553	3.865	3.558	2.912	2.136	1.578	1.442	1.185
80	12.862	53.404	1.195	0.639	0.573	4.762	4.168	3.813	3.149	2.344	1.712	1.587	1.361
90	12.889	59.007	1.201	0.669	0.590	4.896	4.339	4.027	3.411	2.582	1.926	1.809	1.563
100	12.867	64.770	1.210	0.678	0.599	5.098	4.523	4.270	3.658	2.790	2.126	2.000	1.721

TABLE X
TOTAL RUNTIME FOR 1000 NETS OF A DIFFERENT DEGREE

	Runtime (s)												
						FLUTE							
Degree	RMST	RST-T	SPAN	BGA	BI1S	$A=1$	$A=2$	$A=3$	$A=4$	$A=6$	$A=8$	$A=10$	$A=12$
10	0.00	0.01	0.19	0.28	0.18	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01
20	0.01	0.02	0.51	0.81	0.93	0.01	0.02	0.02	0.04	0.11	0.21	0.32	0.53
30	0.02	0.03	0.85	1.44	2.81	0.02	0.03	0.04	0.07	0.23	0.52	0.82	1.60
40	0.03	0.04	1.19	2.14	6.48	0.02	0.04	0.06	0.12	0.38	0.92	1.43	2.77
50	0.05	0.04	1.55	2.91	12.43	0.04	0.06	0.08	0.16	0.53	1.37	2.12	4.16
60	0.07	0.04	1.92	3.74	21.27	0.04	0.07	0.11	0.21	0.70	1.86	2.87	5.67
70	0.09	0.06	2.29	4.67	33.29	0.06	0.10	0.12	0.25	0.88	2.39	3.67	7.32
80	0.11	0.06	2.69	5.59	49.12	0.07	0.10	0.15	0.30	1.05	2.94	4.51	9.08
90	0.13	0.07	3.24	6.54	70.96	0.08	0.12	0.17	0.35	1.22	3.47	5.38	10.83
100	0.16	0.08	3.85	7.65	97.64	0.10	0.15	0.19	0.41	1.41	4.07	6.28	12.76

(0.005%) have a degree of more than 60. Therefore, for VLSI applications, it should be enough to observe the behavior of algorithms for a degree of up to 100. One thousand nets are randomly generated for each degree. The average wirelength error and total runtime are reported in Tables IX and X, respectively.

From Tables IX and X, for nets with degree 10 to 30, FLUTE is clearly the best algorithm. It can be as fast as extremely fast algorithms (RMST and RST-T) yet much more accurate. It can also be more accurate than very accurate algorithms (SPAN, BGA, and BI1S) yet much faster (note that the advantages of FLUTE over other algorithms in both accuracy and runtime are even more significant for nets with degree 9 or less as solutions can be obtained directly from the lookup table).

For higher degree nets, FLUTE with a small A value can generate reasonably accurate solutions in a very short runtime. Other algorithms are either far less accurate or much slower. Therefore, FLUTE is still the most suitable algorithm for higher degree nets if moderate accuracy is enough. If very accurate solutions (say $< 2\%$ error) are desired for nets with a degree of 50 or more, a large A value is required for FLUTE. In that case, FLUTE may not be the fastest algorithm.

VIII. CONCLUSION

In this paper, we introduced a fast and accurate lookup table based RSMT algorithm called FLUTE. The table stores for low-degree nets the set of POWVs associated with each position sequence and an RSMT topology associated with each POWV. We proposed an algorithm based on boundary compaction to generate the sets of POWVs up to degree 9. We designed an MST-based approach to determine the most efficient way to evaluate each set of POWVs. We presented a net-breaking technique to divide a high-degree net into low-degree nets so that the table lookup estimation can be used. We also presented a scheme to allow users to control the tradeoff between accuracy and runtime. The experimental results with industrial nets showed that FLUTE with default accuracy is more accurate than the Batched 1-Steiner heuristic and is almost as fast as RMST construction.

APPENDIX I
PROOFS FOR SECTION III

This appendix contains the proofs of the lemmas regarding the optimality of the lookup table generation algorithm described in Section III. Lemmas 2–6 are directly used in Section III. However, in order to prove these lemmas, two additional lemmas (Lemmas 9 and 10) are required. They are added at the end of the appendix.

Lemma 2: A grid G is compactable if it has a boundary with only one pin.

Proof: Assume without loss of generality that the left boundary of G has only one pin P. Let G' be the reduced grid obtained by compacting G at the left boundary. Therefore, the first entry in the POWVs of G corresponds to the compacted edges. We show that every POWV V of G must be in the form $(1, V')$, where V' is a POWV of G'. Consider any POST T associated with V. We can prove that it has exactly one branch from P to other pins. If there are multiple branches from P to other pins [as shown in Fig. 13(a)], another Steiner tree with a single branch can be constructed as follows. Let l be the second Hanan-grid line from the left boundary. The edges of T on the left of l can be replaced with a vertical segment along l connecting the subtrees of T on the right of l and a horizontal edge

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

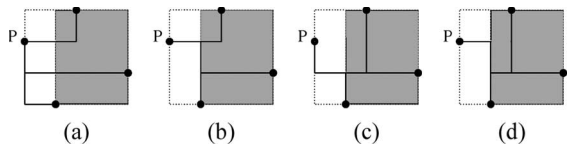


Fig. 13. Illustrations for the proof of Lemma 2.

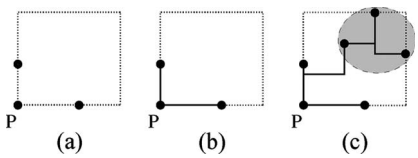


Fig. 14. Illustrations for the proof of Lemma 3.

from P to the segment [as shown in Fig. 13(b)]. The POWV of this tree is better than V in the first entry and is at least as good in all other entries, contrary to the fact that V is potentially optimal. Hence, any POST must have a single branch from P, which implies that the first entry of V should be one. Moreover, if the branch does not go horizontally from P [as shown in Fig. 13(c)], it can be "flipped" [as shown in Fig. 13(d)] to obtain a tree with the same wirelength vector as V. By shifting P along the horizontal branch until the next Hanan-grid line, the grid becomes G'. Hence, the remaining entries of V should form a POWV of G'. ■

Lemma 3: A grid G is compactable if it has a corner with one pin P and if both boundaries adjacent to P have exactly one other pin.

Proof: Assume without loss of generality that P is at the lower left corner, as shown in Fig. 14(a). Assume on the contrary that there is a POWV of G such that its entries associated with all four boundaries are better than those obtained by boundary compaction. Consider any Steiner tree associated with this POWV. By Lemma 9, for both the left and the bottom boundaries, the two pins should be connected by a branch along the boundary, as shown in Fig. 14(b). If G has no other pins besides the three, G is obviously compactable. Otherwise, these three pins should be connected to the rest of the tree by a branch. Suppose without loss of generality that the branch is originated from the left boundary, as shown in Fig. 14(c). Such a solution is not better than those obtained by compacting the left boundary. It contradicts the assumption. Hence, G must be compactable. ■

Lemma 4: A grid G is compactable if it has up to six pins at the four boundaries.

Proof: If G has a boundary with only one pin, then Lemma 2 shows that it is compactable. Therefore, we focus on G with at least two pins on each boundary. As G has at most six pins on the boundaries and each boundary has at least two pins, at least two corners should have a pin so that it can be shared by two boundaries. All cases that satisfy the aforementioned conditions are shown in Fig. 15. Note that only pins on the boundaries are considered. In addition, note that cases which are symmetrical to one of those in Fig. 15 are not shown.

Lemma 3 can be applied to show that all cases except case (f) are compactable (the pin P can be the one at the lower left corner). Lemma 10 can be applied to show that case (f) is also compactable. Therefore, a grid with six or less pins at the boundaries is always compactable. ■

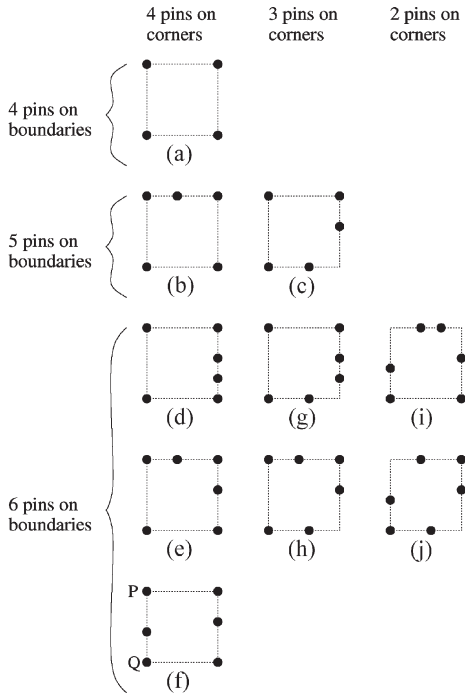


Fig. 15. Illustrations for the proof of Lemma 4.

Lemma 5: Boundary compaction together with the near-ring structures can generate all the POWVs for a grid with seven pins.

Proof: Consider a grid G with seven pins, which is not compactable. By Lemma 4, all seven pins should be on the boundaries. By Lemma 2, there should be at least two pins on each boundary. As G has seven pins at the boundaries and each boundary has at least two pins, at least one corner should have a pin so that it can be shared by two boundaries. All cases that satisfy the previously mentioned conditions are shown in Fig. 16. Note that cases which are symmetrical to one of those in Fig. 16 are not shown.

Lemma 3 can be applied to show that cases (a), (b), (e), (h), (i), (j), (k), (m), (n), and (o) are compactable. Lemma 10 can be applied to show that cases (c), (g), and (l) are compactable.

It is not hard to see that cases (d) and (f) are not compactable. However, we can prove that the POWVs missed by boundary compaction are all covered by the near-ring structures. Assume that it is not the case. In other words, there is a POWV missed by boundary compaction such that the associated Steiner tree has some branches not along the boundaries. We consider the following two cases.

Case 1) Those branches only connect adjacent boundaries. Then, those branches can be "flipped" such that all branches of the Steiner tree are along the boundaries. Moreover, the resulting POWV is the same or better. Hence, the POWV can be generated by the near-ring structures.

Case 2) Those branches also connect two nonadjacent (i.e., opposite) boundaries. Consider case (f). Assume without loss of generality that the left and right boundaries are connected by branches not along the boundaries. By Lemma 9, the two pins at the bottom boundary should be connected by a branch

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

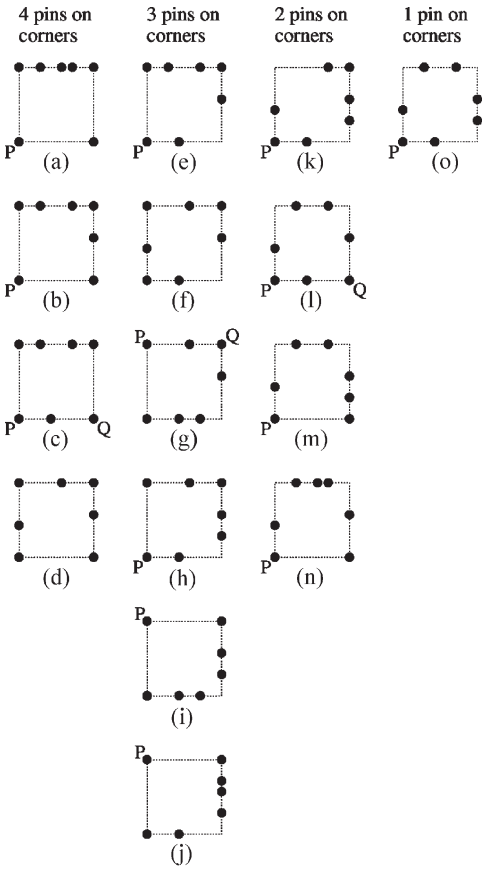


Fig. 16. Illustrations for the proof of Lemma 5.

along the boundary. In addition, at least two of the three pins at the top boundary should be connected by a branch along the boundary. If the left two pins are connected, such a solution is not better than those obtained by compacting the grid at the left boundary. If the right two pins are connected, such a solution is not better than those obtained by compacting the grid at the right boundary. Similar arguments can be applied to handle case (d).

Lemma 6: For a net with n pins where $7 \leq n \leq 10$, boundary compaction together with Connect-adj-pins() with distance $d = n - 3$ can generate all POWVs.

Proof: A net with n pins corresponds to an $n \times n$ Hanan grid such that each grid line has one pin. By Lemma 2, all four boundaries can be compacted once so that an $(n-2) \times (n-2)$ grid G is formed. Any two pins on the same boundary of G are at a distance at most $n-3$ grid lines apart. Hence, Connect-adj-pins$(G, n-3)$ can generate any branch along any boundary of G.

The only remaining issue is that boundary compaction may not be able to generate the branches originating from a branch introduced by Connect-adj-pins(). The reason is that after Connect-adj-pins() connects several pins on a boundary by a branch B, those pins are replaced by a single pseudo-pin. If there is more than one branch connecting B to the remaining pins in a POST, compacting that boundary will not generate this POST [see Fig. 17(a) for an illustration].

We show in the following that if a net has ten pins or less, there always exists a boundary such that boundary compaction

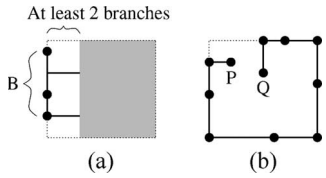


Fig. 17. Illustrations for the proof of Lemma 6.

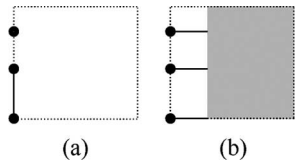


Fig. 18. Illustrations for the proof of Lemma 9.

can be applied. For any branch B introduced by Connect-adj-pins() in a boundary that cannot be compacted, the number of pins on B should be more than the number of branches connecting B to the remaining pins. Otherwise, this boundary can be compacted directly without even applying Connect-adj-pins(). Therefore, there should be at least three pins on B. As there are at most ten pins in the grid, it is impossible to have at least three pins on each boundary unless some corner pins are shared. It is impossible to share all four corners because a ring (i.e., nontree) structure will be formed. Consider the case that three corner pins are shared, as shown in Fig. 17(b). There should be at least nine pins on the boundaries. Furthermore, there should be at least two other pins (P and Q) not on the boundaries. This case is impossible as the total number of pins is at least 11. It is not hard to see that if less than three corner pins are shared, even more pins are required to make the grid not compactable. ■

The following lemma is used in the proof of Lemmas 3, 5, and 10.

Lemma 9: If a grid G is not compactable, then, for any POST associated with any POWV missed by boundary compaction, there should be a branch connecting at least two pins along each of the four boundaries.

Proof: By Lemma 2, there should be at least two pins on each boundary. Without loss of generality, consider the pins on the left boundary. The lemma claims that at least two pins are connected by a branch along the left boundary, as shown in Fig. 18(a). Otherwise, each pin should be connected to the rest of the tree by a separate branch, as shown in Fig. 18(b). Such a solution is not better than those generated by compacting the left boundary. ■

The following lemma is used in the proof of Lemmas 4 and 5.

Lemma 10: A grid G is compactable if it has two adjacent corners with pins P and Q and each of the three boundaries involving P and Q has exactly one other pin.

Proof: Assume without loss of generality that P is at the lower left corner and Q is at the lower right corner, as shown in Fig. 19(a). Assume on the contrary that there is a POWV of G such that its entries associated with all four boundaries are better than those obtained by boundary compaction. Consider any Steiner tree associated with this POWV. By Lemma 9, for both the left and right boundaries, the two pins should be connected by a branch along the boundary, as shown in Fig. 14(b).

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

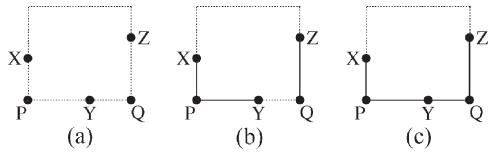


Fig. 19. Illustrations for the proof of Lemma 10.

Moreover, pin Y should be connected to at least one of the corner pins P and Q by a branch along the bottom boundary. Without loss of generality, assume that Y is connected to P, as shown in Fig. 19(b).

The subtree consisting of P, X, and Y should be connected to the rest of the tree by a branch. If the branch is originated from the left boundary, such a solution is not better than those obtained by compacting the left boundary. If the branch is originated from the bottom boundary and is not along the bottom boundary, such a solution is not better than those obtained by compacting the bottom boundary. If the branch is originated from the bottom boundary and is along the bottom boundary [i.e., the branch connects Y and Q, as shown in Fig. 19(c)], we consider two cases based on whether there are other pins besides the five. If there is no other pin, G is obviously compactable from the top boundary. Otherwise, the subtree consisting of the five pins should be connected to the other pins by a branch. If the branch is originated from the left/bottom/right boundary, such a solution is not better than those obtained by compacting the left/bottom/right boundary. ■

APPENDIX II
PROOFS FOR SECTION IV

This appendix contains the proofs of the Lemmas 7 and 8 regarding the lookup table size reduction techniques described in Section IV. Lemma 11, at the end of the appendix, is required by the other lemmas.

Lemma 7: Group index is a one-to-one mapping from the groups of degree n to an integral value between 0 and $n! - 1$.

Proof: As $p_j \geq 0$ for all j, it is obvious that any group index $k \geq 0$. In addition, by the fact that $p_j \leq j - 1$ for all j and Lemma 11 with $i = 1$, it is easy to prove that $k \leq n! - 1$.

For any two different groups, assume the corresponding modified position sequences to be $p_1 p_2 \ldots p_n$ and $p'_1 p'_2 \ldots p'_n$ and the corresponding group indexes to be k and k', respectively. Since the groups and, hence, the position sequences are different, the modified sequences should also be different. Let i be the smallest index such that $p_i \neq p'_i$. Without loss of generality, assume that $p_i > p'_i$.

$$\begin{aligned} k - k' &= \frac{n!}{i!} \times (p_i - p'_i) + \prod_{j=i+1}^{n} \frac{n!}{j!} \times (p_j - p'_j) \\ &\geq \frac{n!}{i!} + \prod_{j=i+1}^{n} \frac{n!}{j!} \times (p_j - p'_j) \\ &\geq \frac{n!}{i!} - \prod_{j=i+1}^{n} \frac{n!}{j!} \times (j - 1) \\ &= \frac{n!}{i!} - \left(\frac{n!}{i!} - 1 \right) \quad \text{by Lemma 11} \\ &= 1. \end{aligned}$$

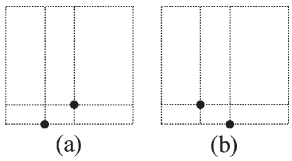


Fig. 20. Relative position of the bottom boundary pins for two equivalent groups.

Therefore, $k \neq k'$. In other words, different groups will have different group indexes.

Since there are $n!$ groups and each group is mapped to a different integer between 0 and $n! - 1$, the lemma is proved. ■

Lemma 8: Any group of degree n is equivalent to a group with a group index between 0 and $n!/4 - 1$.

Proof: For simplicity, we call a group with group index k as group k. For any group k with $k \geq n!/4$, assume that the position sequence is $s_1 s_2 \ldots s_n$ and that the modified position sequence is $p_1 p_2 \ldots p_n$. Consider the following three cases.

Case 1) $3n!/4 \leq k < n!$.

For the group k' that is horizontally symmetrical to group k, assume that the position sequence is $s'_1 s'_2 \ldots s'_n$ and that the modified position sequence is $p'_1 p'_2 \ldots p'_n$. It is clear that $s'_j = n + 1 - s_j$ for $1 \leq j \leq n$. Therefore, it follows from the definition of the modified position sequence that $p'_j = j - 1 - p_j$ for $1 \leq j \leq n$. Thus

$$\begin{aligned} k' &= \prod_{j=1}^{n} \frac{n!}{j!} \times p'_j \\ &= \prod_{j=1}^{n} \frac{n!}{j!} \times (j - 1 - p_j) \\ &= \prod_{j=1}^{n} \frac{n!}{j!} \times (j - 1) - k \\ &= n! - 1 - k \quad \text{by Lemma 11.} \end{aligned}$$

Therefore, $0 \leq k' \leq n!/4 - 1$.

Case 2) $n!/2 \leq k < 3n!/4$.

As $k \geq n!/2$, p_2 should be one, which implies that $s_1 < s_2$, as shown in Fig. 20(a). Consider the group k' in Fig. 20(b) which is the same as group k in Fig. 20(b) except for the relative position of the bottom two pins. Groups k and k' are equivalent due to boundary compaction. For group k', assume that the position sequence is $s'_1 s'_2 \ldots s'_n$ and that the modified position sequence is $p'_1 p'_2 \ldots p'_n$. Then, $s'_1 > s'_2$, which implies that $p'_2 = 0$. $p_j = p'_j$ for all $j \neq 2$. Therefore, $k' = k - n!/2$. Therefore, $0 \leq k' \leq n!/4 - 1$.

Case 3) $n!/4 \leq k < n!/2$.

We can use the same argument as Case 1) to prove that group k is equivalent to group $k'' = n! - 1 - k$. Therefore, $n!/2 \leq k'' < 3n!/4$. Then, we can use the same argument as Case 2) to prove that group k'' (i.e., group k) is equivalent to group $k' = k'' - n!/2$. Therefore, $0 \leq k' \leq n!/4 - 1$.

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.

Group k' is between 0 and $n!/4 - 1$ in all cases. Hence, the lemma is proved. ■

Lemma 11: For any i such that $1 \leq i \leq n$

$$\prod_{j=i}^{n} \frac{n!}{j!} \times (j-1) = \frac{n!}{(i-1)!} - 1.$$

Proof: The lemma can be proved by induction on i. If $i = n$, both sides are equal to $n-1$. Assume that $\prod_{j=i}^{n}(n!/j!) \times (j-1) = (n!/(i-1)!) - 1$ for some i.

$$\begin{aligned}\prod_{j=i-1}^{n} \frac{n!}{j!} \times (j-1) &= \frac{n!}{(i-1)!} \times (i-2) + \prod_{j=i}^{n} \frac{n!}{j!} \times (j-1) \\ &= \frac{n!}{(i-1)!} \times (i-2) + \frac{n!}{(i-1)!} - 1 \\ &= \frac{n!}{(i-1)!} \times (i-1) - 1 \\ &= \frac{n!}{(i-2)!} - 1.\end{aligned}$$

Hence, the lemma is proved. ■

ACKNOWLEDGMENT

The authors would like to thank Prof. C. K. Cheng for providing the source code of RST-T and Prof. H. Zhou for providing the source code of SPAN.

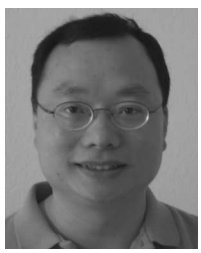
REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.
[2] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM J. Appl. Math.*, vol. 30, no. 1, pp. 104–114, Jan. 1976.
[3] F. K. Hwang, D. S. Richards, and P. Winter, "The Steiner tree problem," in *Annals of Discrete Mathematics*. Amsterdam, The Netherlands: Elsevier, 1992.
[4] D. M. Warme, P. Winter, and M. Zachariasen, "Exact algorithms for plane Steiner tree problems: A computational study," in *Advances in Steiner Trees*, D. Z. Du, J. M. Smith, and J. H. Rubinstein, Eds. Norwell, MA: Kluwer, 2000, pp. 81–116.
[5] *GeoSteiner—Software for Computing Steiner Trees*. [Online]. Available: http://www.diku.dk/geosteiner
[6] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang, "Closing the gap: Near-optimal Steiner trees in polynomial time," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 11, pp. 1351–1365, Nov. 1994.
[7] I. I. Mandoiu, V. V. Vazirani, and J. L. Ganley, "A new heuristic for rectilinear Steiner trees," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 1999, pp. 157–162.
[8] M. Borah, R. M. Owens, and M. J. Irwin, "An edge-based heuristic for Steiner routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 12, pp. 1563–1568, Dec. 1994.
[9] H. Zhou, "Efficient Steiner tree construction based on spanning graphs," in *Proc. Int. Symp. Phys. Des.*, 2003, pp. 152–157.
[10] H. Zhou, N. Shenoy, and W. Nicholls, "Efficient spanning tree construction without Delaunay triangulation," *Inf. Process. Lett.*, vol. 81, no. 5, pp. 271–276, 2002.
[11] A. Kahng, I. Mandoiu, and A. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear Steiner trees," in *Proc. Asian South Pacific Des. Autom. Conf.*, 2003, pp. 827–833.
[12] J. Soukup, "Circuit layout," *Proc. IEEE*, vol. 69, no. 10, pp. 1281–1304, Oct. 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 Int. Workshop Syst. Level Interconnect Prediction*, 2002, pp. 85–89.
[14] C. J. Alpert, "The ISPD98 circuit benchmark suite," in *Proc. Int. Symp. Phys. Des.*, 1998, pp.80–85. [Online]. Available: http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html
[15] A. B. Kahng and I. Mandoiu, *RMST-Pack: Rectilinear Minimum Spanning Tree Algorithms*. [Online]. Available: http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST
[16] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, Mar. 1966.
[17] C. Chu, *FLUTE: Fast lookup table based technique for RSMT construction and wirelength estimation*. [Online]. Available: http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT
[18] C. Chu, "FLUTE: Fast lookup table based wirelength estimation technique," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2004, pp. 696–701.
[19] C. Chu and Y.-C. Wong, "Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 28–35.
[20] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Toward CAD-IP reuse: The MARCO GSRC bookshelf of fundamental CAD algorithms," in *Proc. IEEE Des. Test*, 2002, pp. 72–81.
[21] N. Viswanathan and C. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 26–33.

Chris Chu received the B.S. degree in computer science from the University of Hong Kong, Hong Kong, in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Texas, Austin, in 1994 and 1999, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Iowa State University, Ames. His areas of expertise include computer-aided design (CAD) of very large scale integration physical design and design and analysis of algorithms. His recent research interests are performance-driven interconnect optimization and fast circuit floorplanning, placement, and routing algorithms.

Dr. Chu has served on the technical program committees of several major conferences including the International Conference on CAD, International Symposium on Physical Design (ISPD), International Symposium on Circuits and Systems, Design Automation and Test in Europe, Asia and South Pacific Design Automation Conference, and System Level Interconnect Prediction (SLIP). He has also served as an Organizer for the Association for Computing Machinery Special Interest Group on Design Automation Ph.D. Forum. He was the recipient of the IEEE TRANSACTIONS ON CAD Best Paper Award in 1999 for his work in performance-driven interconnect optimization, the Bert Kay Best Dissertation Award for 1998–1999 from the Department of Computer Sciences, University of Texas at Austin, and the the ISPD Best Paper Award in 2004 for his work in efficient placement algorithm.



Yiu-Chung Wong received the Ph.D. degree in computer science from the University of Southern California, Los Angeles, in 1997.

He resides in Palo Alto, CA, and is currently an independent consultant for various high-tech companies in the area.

Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on November 14,2024 at 14:11:05 UTC from IEEE Xplore. Restrictions apply.