# FLUTE: Fast Lookup Table Based Wirelength Estimation Technique

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

## Abstract

Wirelength estimation is an important tool to guide the design optimization process in early design stages. In this paper, we present a novel wirelength estimation technique called FLUTE. Our technique is based on pre-computed lookup table to make wirelength estimation very fast and very accurate for low degree[1] nets. We show experimentally that for FLUTE, RMST, and HPWL, the average error in wirelength are 0.72%, 4.23%, and -8.71%, respectively, and the normalized runtime are 1, 1.24, and 0.16, respectively.

## 1 Introduction

In the deep sub-micron / nanometer regime, interconnect issues like delay and routability become the main concern in IC design. Thus, design optimization even in the early design stages needs to be guided by physical information of interconnects (wirelength, congestion, etc.). This paper focuses on layout stages like floorplanning and placement in which module (i.e., pin) locations are being fixed. We can evaluate each tentative floorplan or placement solution by performing real routing on it. However, it will be prohibitively expensive to incorporate routing within the optimization process of an early stage. A more realistic approach is to apply some fast yet accurate techniques to estimate the physical interconnect information.

A very popular technique is the half-perimeter wirelength (HPWL) estimation which equals the half-perimeter of the bounding rectangle of pins [1]. This technique is very efficient. It also provides exact wirelength for optimally routed two-pin nets and three-pin nets. However, it can significantly underestimate wirelength for higher-degree nets. Cheng [2] proposed a net weighting technique to scale up the HPWL estimation. The net weights are degree-dependent constants and are experimentally determined. However, even for different nets with the same degree, the error in the HPWL estimation can be very different. It is impossible to derive a single net weight to accurately scale up the HPWL estimation for all nets.

Another commonly used technique to estimate the wirelength is by rectilinear minimum spanning tree (RMST) [3,4]. This approach can produce good wirelength estimation in reasonable runtime. The best time complexity of RMST is $O(n \log n)$ [3]. However, a simple $O(n^2)$ time implementation of Prim's algorithm is usually used in practice because the degree $n$ are small for most nets [5].

We can also achieve accurate estimation by constructing rectilinear Steiner minimal tree (RSMT) using either optimal algorithms [6, 7] or near-optimal heuristics [8, 9]. But these algorithms are computationally too expensive to use in practice. Recently, Chen et al. [10] presented a very efficient RSMT heuristic called Refined Single Trunk Tree (RST-T). This technique provides very good wirelength estimation for low-degree nets (exact up to degree 5) but not for high-degree nets. For example, the average error is 12.46% for nets with degree 16. Although most nets in a typical circuit have a low degree, there are still a significant proportion of high-degree nets. Moreover, those high-degree nets usually account for a significant proportion of the total wirelength. For the 18 IBM circuits in the ISPD98 benchmark suite, 2.19% of all nets have a degree $\geq 16$ and those nets account for 8.32% of the total wirelength. Hence, the RST-T technique will have limited accuracy in practice.

In this paper, we present a lookup table based routing estimation technique called FLUTE. We show 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 possible routing topologies can be written as a small number of linear combinations of distances between adjacent pins. We call each linear combination a potentially optimal wirelength vector (POWV). We store the few POWVs for each group into a table. To find the optimal 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. This idea works well for low degree nets. For high-degree nets, we proposed a net breaking technique to reduce the net size until the table can be used. We show experimentally that for FLUTE, RMST, and HPWL, the average wirelength error are 0.72%, 4.23%, and -8.71%, respectively, and the normalized runtime are 1, 1.24, and 0.16, respectively.

The remainder of the paper is organized as follows. In Section 2, we present the lookup table idea for wirelength estimation of low-degree nets. In Section 3, we describe the algorithm to generate the POWVs. In Section 4, we derive a very efficient technique to evaluate all the POWVs when estimating wirelength for a given net. In Section 5, we present the net breaking technique for high-degree nets. In Section 6, we show the experimental results.

## 2 Lookup Table for Low-Degree Nets

We define a *net* of degree $n$ to be a set of $n$ pins such that the coordinates of pin $i$ is $(x_i, y_i)$ for $1 \leq i \leq n$. Without loss of generality, assume $x_1 \leq x_2 \leq \cdots \leq x_n$. Let the *vertical sequence* $s_1 s_2 \ldots s_n$ be the list of indexes of all pins sorted in ascending order according to the $y$-coordinate.[2] For example, for the net in Figure 1, its vertical sequence is 3142.

In this paper, we only consider routing along the Hanan grid[3] as Hanan [11] pointed out that an optimal RSMT can always be constructed based on the Hanan grid. 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 *horizontal edge length* as $h_i = x_{i+1} - x_i$ and

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

[2] Ties can be broken arbitrarily.

[3] Given a net, the Hanan grid is formed by drawing a horizontal line and a vertical line through each pin.

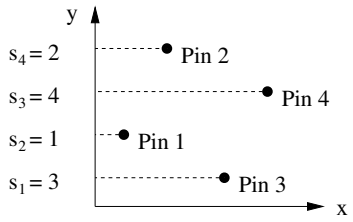



Figure 1: An illustration of the vertical sequence of a net.

*vertical edge length* as $v_i = y_{s_{i+1}} - y_{s_i}$ for $1 \leq i \leq n$. These definitions are illustrated in Figure 2.

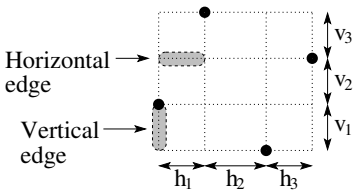



Figure 2: An illustration of horizontal and vertical edge lengths.

**Observation 1** *The wirelength of any routing solution on the Hanan grid can always be written as a linear combination of edge lengths such that all coefficients are positive integers.*

For example, for the net in Figure 1, the wirelength of the three possible routing solutions shown in Figure 3 (a), (b), and (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 routings in Figure 3 (a), (b), and (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.

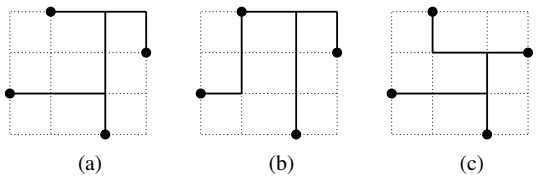



Figure 3: Three possible routings for the net in Figure 1.

In order to find the optimal wirelength for a given net, we can enumerate all possible wirelength vectors. Note that although the number of possible routing solutions is huge, the number of possible wirelength vectors is much less. More importantly, we observe that we only need to consider a few wirelength vectors which have the potential to produce the optimal wirelength. Most vectors are redundant because they have a larger or equal value than 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 *potentially optimal wirelength vector* (POWV). This observation is summarized below.

**Observation 2** *For every low-degree net, there is only a few potentially optimal wirelength vectors (POWVs).*

For example, for all degree-3 nets, the only optimal wirelength vector is $(1, 1, 1, 1)$, which corresponds to the HPWL. For the net in Figure 1, the only two POWVs 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 routing solutions corresponding to these two wirelength vectors are given in Figure 4. Some statistics on the number of POWVs will be given later in Table 1.

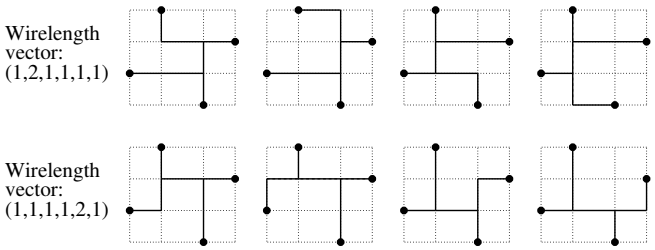



Figure 4: The potentially optimal routing solutions for the net in Figure 1.

If we can pre-compute all the POWVs and store them in a lookup table, the optimal wirelength will be easy to find. However, the number of different nets is infinite as the pin coordinates can take infinite different values. To handle this problem, we try to group together nets which can share the same set of POWVs. To see which nets can be grouped together, we first introduce the following definition. Two routing solutions 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 illustrated in Figure 5.

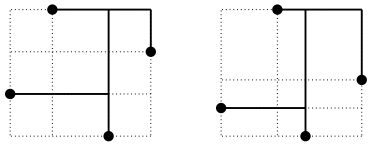


Figure 5: Topologically equivalent routing solutions for two different nets.

**Observation 3** *The wirelengths of topologically equivalent routing solutions can be expressed by the same wirelength vector.*

For example, the wirelength of the two solutions in Figure 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.

**Lemma 1** *If two nets have the same vertical sequence, then every routing solution of one net is topologically equivalent to a routing solution 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 vertical sequence, the pins of the two nets are in the same locations in the Hanan grid. So every routing solution of one net will also be a routing solution of the other. □

Based on Observation 3 and Lemma 1, nets with the same vertical sequence can be grouped together to share the set of POWVs. Since the vertical sequence of a degree-$n$ net is a permutation of $12 \ldots n$, there should be $n!$ groups. This observation is summarized below.

**Observation 4** *The set of all degree-$n$ nets can be divided into $n!$ groups according to the vertical sequence such that all nets in each group share the same set of POWVs.*

Our wirelength estimation technique pre-computes a lookup table to store the set of POWVs associated with each group for low degree nets. To compute the optimal wirelength for a given net, we can find out the vertical 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 entries of the vector with $h_i$'s and $v_i$'s. The minimum value over all vectors will give the optimal wirelength.

In fact, our approach can provide other information in addition to the wirelength estimation. The table contains more detailed information on the utilization of different regions of the Hanan grid. That information can be used for better congestion estimation. Moreover, for each POWV in each group, one or more of the routing topologies can be stored. Then routing solutions can also be generated.

# 3 Generation of Wirelength Vectors

In this Section, we discuss the generation of the sets of POWVs. For each small net degree and for each group (i.e., vertical sequence), we need to generate all possible routing topologies, find the corresponding wirelength vectors, and prune away the redundant ones. The set of remaining vectors are the POWVs for the group. A trivial approach to generate all possible routing topologies is to enumerate all possible combinations of using and not using each edge in the Hanan grid and check if the resulting sub-graph 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 any one of the four boundaries, i.e., shifting all pins on a boundary to the grid line adjacent to that boundary. The set of routing topologies of the original problem can be generated by expanding the routing topologies of the reduced grid back to the original grid. Figure 6 uses the compaction of left boundary as an example to illustrate the idea.

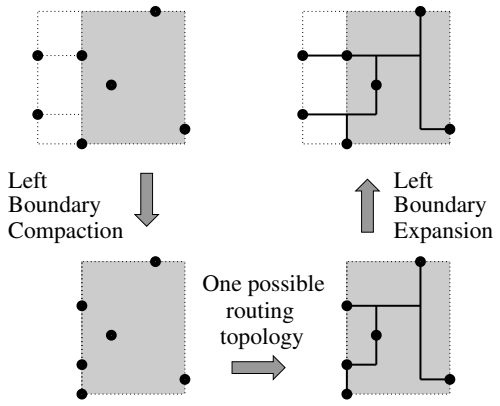



Figure 6: An illustration of left boundary compaction.

We can route a net by performing boundary compaction and expansion recursively. By compacting the four boundaries in different order, a set of different routing topologies can be generated. Since we are performing the routing in a restricted way, it is possible that some routing topologies and hence some wirelength vectors are missed. The following lemma suggests that nothing will be missed if a boundary with only one pin is compacted.

**Lemma 2** *Given a grid G with some pins at grid nodes. Let G' be a reduced grid from G by compacting a boundary with only one pin. Every POWV of G can be obtained by adding an entry of value 1 corresponding to the compacted edge to some POWV of G'.*

**Proof:** Assume without loss of generality that the left boundary with one pin P is compacted. So the first entry in POWVs of G corresponds to the compacted edge. We show that any POWV $V$ of G must be in the form $(1, V')$ where $V'$ is a POWV of G'. Consider any routing topology associated with $V$. If there are multiple branches from P to other pins (as in Figure 7(a)), another routing topology with a single branch can be constructed (as in Figure 7(b)). The POWV of this topology 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, there should only be a single branch from P, which implies the first entry of $V$ should be 1. Moreover, if the branch does not go horizontally from P (as shown in Figure 7(c)), it can be "flipped" (as in Figure 7(d)) to obtain a topology 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'. □

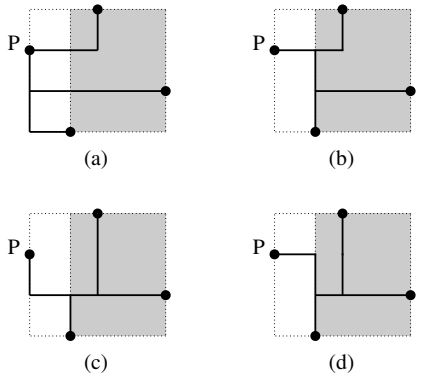



Figure 7: Illustration for proof of Lemma 2.

Our algorithm is given in Figure 8. Instead of first enumerating all routing topologies and then producing the wirelength vectors, we generate wirelength vectors directly, which is much easier and much more efficient. We also incorporate the pruning of redundant wirelength vectors into the algorithm to prune as early as possible, which further improves the efficiency of the algorithm.

**Algorithm Gen-WVs(G)**
Input: G is a grid with some pins at grid nodes
Output: A set of WVs for G
**begin**
1. If G is simple enough,
2. generate and return the set of POWVs for G
3. else if any boundary *b* contains only one pin,
4. return Expand-*b*(Gen-WVs(Compact-*b*(G)))
else
5. S={Some extra WVs for routing topologies
not considered by boundary compaction}
6. return Prune(S ∪ Expand-left(Gen-WVs(Compact-left(G)))
∪ Expand-right(Gen-WVs(Compact-right(G)))
∪ Expand-top(Gen-WVs(Compact-top(G)))
∪ Expand-bot(Gen-WVs(Compact-bot(G))))
**end**

Figure 8: The wirelength vector generation algorithm.

In Step 1–2, we directly generate the POWVs when G consists of

| 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 | 15.803 | 34 |

Table 1: Number of POWVs in a group for nets of a given degree.

a single (horizontal or vertical) grid line or is a $2 \times 2$ grid. Step 3–4 is based on Lemma 2. Since one recursive call is made instead of four and this case occurs frequently for low degree nets, the runtime of the algorithm can be dramatically reduced. Step 5 is to include some extra wirelength vectors missed by boundary compaction to ensure that POWVs are generated. We prove in the following theorem that Step 5 is not needed for nets with degree 6 or less.

**Theorem 1** *The algorithm Gen-WVs() enumerates all POWVs for nets with degree 6 or less, even if no extra wirelength vector is included in Step 5.*

**Proof sketch:** We can always apply Lemma 2 to reduce the grid until all boundaries contain at least two pins. If the net has only six pins or less, it implies at least two of the pins are at the grid corners. Some detailed case analysis can show that all POWVs will be generated by the algorithm. □

If we ignore Step 5 for nets with degree 7 or more, the wirelength is not always optimal. We notice that for those non-optimal degree-7 nets, there is always a stage during recursive compaction such that all 7 pins are on the boundary of the grid. In that case, the optimal wirelength may be produced by performing the routing along the grid boundary. The boundary compaction technique may (or may not) miss such a topology and the associated wirelength vector. This problem also occurs for nets with degree 8 or more. Hence, in Step 5 of algorithm Gen-WVs(), if there are $n$ ($\geq 7$) pins on boundaries and no pin inside, $n$ wirelength vectors will be included in S. Each vector corresponds to a ring topology that surrounds the grid with part of the ring between one of the $n$ pairs of adjacent pins removed. Although we do not have a formal proof that after introducing the near-ring topology, the set of WVs generated includes all POWVs for 7-pin nets, we have experimentally verified that the FLUTE wirelength is always exact for 3 million randomly generated 7-pin nets. So it is very likely that FLUTE is also exact for degree 7. At least, it can be considered to be exact in practice. The near-ring topology in Step 5 also helps to reduce the estimation error for nets with degree 8 or more but is not enough to ensure that the WVs generated are POWVs. In the following, for the sake of simplicity, we will refer to the WVs generated by Gen-WVs() as POWVs, although we lack a formal proof for degree 7 and we know it is not the case for degree 8 or more.

The number of POWVs generated by the algorithm Gen-WVs() is listed in Table 1. Note that the numbers in this table may not be accurate for $n = 7$ (although very unlikely) and $n = 8$.

# 4 Minimum Wirelength Computation

To compute the minimum wirelength of a given net, we need to consider the corresponding set of POWVs. A straightforward approach is to evaluate the POWVs independently. For each POWV $(\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n)$, we compute the expression $WL = \sum_{i=1}^{n} \alpha_i h_i + \sum_{i=1}^{n} \beta_i v_i$. 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 length should be used at least once. So it is better to evaluate the expression as $WL = HPWL + \sum_{i=1}^{n} (\alpha_i - 1) h_i + \sum_{i=1}^{n} (\beta_i - 1) v_i$. Then we have $2n$ less terms to add.

| Degree $n$ | Average # of ADD/SUB | | | |
|---|---|---|---|---|
| | per group | | per POWV | |
| | 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 | 14.222 | 10.333 | 3.208 | 2.331 |
| 7 | 39.651 | 20.025 | 4.999 | 2.525 |
| 8 | 114.687 | 41.521 | 7.257 | 2.627 |

Table 2: Average number of addition/subtraction required.

However, we observe that most POWVs shared by a group of nets are very similar to one another. Many are differed 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 dependency among the POWVs, the evaluation of all POWVs for a net can be made more efficient than the independent approach.

The problem of determining the best dependency among POWVs for a given group can be transformed into a minimum spanning tree problem. Consider a group associated with a set of $k$ POWVs. We construct a complete graph with $k+1$ nodes. $k$ of these nodes correspond to the $k$ 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 the 1-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 a minimum spanning tree 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 minimum spanning tree gives the number of addition/subtraction required to compute all $k$ POWVs.

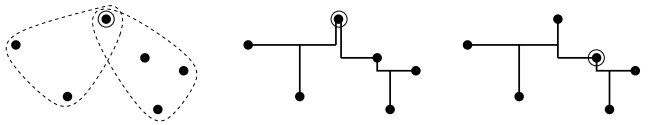
The average number of addition/subtraction required for the independent approach and the MST-based approach are listed in Table 2. Columns two and three give the average number per group, which is proportional to the average runtime to evaluate a net. It is clear that the MST-based approach can significantly 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, while for the MST-based approach, the number of entries to be add/subtract increases slowly with net degree.

# 5 Algorithm for High-Degree Nets

For high-degree nets, the CPU time to generate the POWVs will be significant and the memory requirement for the table will be huge. So 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 $D$. Nets with degree higher than $D$ are

broken into several sub-nets with degree ranging from 2 to $D$ to which the table lookup estimation can be applied. To break a net, we first select a breaking direction (either horizontal or vertical) and a pin. We then separate the other pins into two sub-nets according to the pin coordinate and the breaking direction. The selected pin is included in both sub-nets as well. For example, in Figure 9(a), pin 3 is selected to break a 7-pin net horizontally. Then two sub-nets are routed independently. If the degree of a sub-net is still greater than $D$, we can apply this idea recursively.



(a) (b) (c)

Figure 9: An illustration of net breaking.

However, Figure 9(b) demonstrates that there may be some redundant wires in the resulting routing solution. Note that for the same example, if we break it horizontally at pin 4 as shown in Figure 9(c), the solution will be optimal. So we need to try both directions and selecting different pins. If we really make recursive calls to evaluate each of these possibilities, the runtime will be significantly increased. In our implementation, we use the HPWL to predict the wirelength of each possibilities. Then only one pin in each direction is selected to really break the net. The better of the two wirelengths will be returned.

# 6 Experimental Results

We compare the following five techniques: the exact RSMT software GeoSteiner 3.1 [12], the near-optimal Batched Iterated 1-Steiner (BI1S) heuristic for RSMT [8], HPWL, an efficient $O(n^2)$ implementation of Prim's algorithm for RMST [5], and FLUTE with D=7. HPWL and FLUTE are implemented by us in C. All the other three programs are downloaded from the GSRC Bookshelf [13]. The 18 IBM circuits in the ISPD98 benchmark suite are used. Some information of the benchmark circuits are given in Table 3. There are totally 1.57 million nets. The placement is generated by FastPlace [14]. All experiments are carried out on a 750 MHz Sun Sparc-2 machine.

The wirelength estimation comparison is shown in Table 4. GeoSteiner provides the exact wirelength for all nets. As the table shows, FLUTE can consistently produce accurate wirelength estimation for all circuits. The average error over all nets is only 0.72%. RMST produces acceptable accuracy but is far less accurate than FLUTE. The average error of RMST over all nets is 4.23%. HPWL underestimates the wirelength significantly. The average error of HPWL over all nets is -8.71%. BI1S is the most accurate.

The breakdown of the wirelength estimation for nets with different degree is shown in Table 5. 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 high degree nets and the contribution of high degree nets to the wirelength is very significant. For example, nets with degree 8 or higher account for 11.91% of all nets and contribute 35.27% of total wirelength. Columns 4 to 7 report the percentage error in wirelength. As the table shows, all four techniques have more error for nets with higher degree. In particular, the HPWL underestimates the wirelength significantly for

| Circuit | # of nets | Ave. degree |
|---|---|---|
| ibm01 | 14111 | 3.58 |
| ibm02 | 19584 | 4.15 |
| ibm03 | 27401 | 3.41 |
| ibm04 | 31970 | 3.31 |
| ibm05 | 28446 | 4.44 |
| ibm06 | 34826 | 3.68 |
| ibm07 | 48117 | 3.65 |
| ibm08 | 50513 | 4.06 |
| ibm09 | 60902 | 3.65 |
| ibm10 | 75196 | 3.96 |
| ibm11 | 81454 | 3.45 |
| ibm12 | 77240 | 4.11 |
| ibm13 | 99666 | 3.58 |
| ibm14 | 152772 | 3.58 |
| ibm15 | 186608 | 3.84 |
| ibm16 | 190048 | 4.10 |
| ibm17 | 189581 | 4.54 |
| ibm18 | 201920 | 4.06 |
| All | 1570355 | 3.92 |

Table 3: Benchmark information.

| | Wirelength error | | | |
|---|---|---|---|---|
| Circuit | BI1S | HPWL | RMST | FLUTE |
| ibm01 | 0.10% | -8.86% | 4.09% | 0.70% |
| ibm02 | 0.12% | -12.56% | 5.85% | 1.02% |
| ibm03 | 0.10% | -8.65% | 4.64% | 0.64% |
| ibm04 | 0.06% | -6.30% | 4.05% | 0.40% |
| ibm05 | 0.11% | -11.22% | 4.49% | 1.17% |
| ibm06 | 0.14% | -12.98% | 5.96% | 0.94% |
| ibm07 | 0.09% | -8.29% | 4.72% | 0.49% |
| ibm08 | 0.12% | -12.45% | 4.78% | 1.38% |
| ibm09 | 0.07% | -7.97% | 4.33% | 0.58% |
| ibm10 | 0.08% | -7.85% | 4.11% | 0.60% |
| ibm11 | 0.06% | -6.16% | 4.02% | 0.37% |
| ibm12 | 0.07% | -7.61% | 3.78% | 0.59% |
| ibm13 | 0.11% | -9.24% | 4.78% | 0.71% |
| ibm14 | 0.07% | -6.98% | 3.91% | 0.48% |
| ibm15 | 0.08% | -8.06% | 4.20% | 0.65% |
| ibm16 | 0.09% | -9.26% | 4.23% | 0.77% |
| ibm17 | 0.08% | -8.15% | 3.90% | 0.66% |
| ibm18 | 0.10% | -10.65% | 4.43% | 1.07% |
| All | 0.09% | -8.71% | 4.23% | 0.72% |

Table 4: Percentage error in wirelength estimation.

| Degree | Net breakdown | | Wirelength error | | | |
|---|---|---|---|---|---|---|
| | # | WL | BI1S | HPWL | RMST | FLUTE |
| 2 | 54.92% | 27.98% | 0.00% | 0.00% | 0.00% | 0.00% |
| 3 | 14.40% | 10.26% | 0.00% | 0.00% | 2.50% | 0.00% |
| 4 | 7.68% | 7.84% | 0.00% | -2.20% | 3.89% | 0.00% |
| 5 | 5.61% | 8.18% | 0.05% | -4.74% | 4.74% | 0.00% |
| 6 | 3.20% | 5.65% | 0.07% | -7.02% | 5.40% | 0.00% |
| 7 | 2.28% | 4.82% | 0.09% | -9.06% | 5.91% | 0.00% |
| 8-13 | 8.40% | 22.71% | 0.17% | -16.28% | 7.13% | 1.29% |
| $\geq$14 | 3.51% | 12.56% | 0.28% | -28.83% | 8.56% | 3.41% |

Table 5: Breakdown of the wirelength estimation according to degree for nets of all 18 circuits.

| Circuit | Runtime (s) | | | | |
|---|---|---|---|---|---|
| | GeoS | BI1S | HPWL | RMST | FLUTE |
| ibm01 | 473.33 | 72.48 | 0.00 | 0.03 | 0.02 |
| ibm02 | 907.86 | 108.93 | 0.01 | 0.06 | 0.05 |
| ibm03 | 856.83 | 140.00 | 0.00 | 0.05 | 0.03 |
| ibm04 | 970.42 | 162.15 | 0.01 | 0.06 | 0.03 |
| ibm05 | 1194.88 | 146.23 | 0.01 | 0.08 | 0.07 |
| ibm06 | 1264.44 | 176.88 | 0.01 | 0.06 | 0.05 |
| ibm07 | 1683.45 | 244.50 | 0.01 | 0.09 | 0.07 |
| ibm08 | 1922.16 | 266.02 | 0.02 | 0.14 | 0.13 |
| ibm09 | 2039.19 | 308.61 | 0.01 | 0.12 | 0.08 |
| ibm10 | 3097.52 | 383.44 | 0.02 | 0.16 | 0.13 |
| ibm11 | 2778.43 | 411.29 | 0.02 | 0.14 | 0.09 |
| ibm12 | 3152.42 | 394.68 | 0.02 | 0.18 | 0.14 |
| ibm13 | 3364.12 | 504.71 | 0.02 | 0.19 | 0.13 |
| ibm14 | 5558.21 | 775.54 | 0.04 | 0.29 | 0.21 |
| ibm15 | 7053.64 | 949.99 | 0.05 | 0.40 | 0.32 |
| ibm16 | 7934.45 | 968.36 | 0.06 | 0.43 | 0.37 |
| ibm17 | 8629.35 | 973.79 | 0.07 | 0.50 | 0.45 |
| ibm18 | 7816.41 | 1036.36 | 0.07 | 0.49 | 0.43 |
| All | 21678 | 2866 | 0.16 | 1.24 | 1 |

Table 6: Runtime comparison. The overall runtimes in the last row are normalized with respect to FLUTE runtime.

high degree nets. Note that if we scale the wirelength by the net weights[4] in [2], we will significantly overestimate the wirelength. FLUTE is exact for nets up to degree 7, while BI1S is exact only up to degree 4. For higher degree nets, FLUTE is still very accurate. The net breaking step is the cause of the error of FLUTE. For higher net degree, the net breaking step will be applied more times. So the wirelength error will also be higher.

The runtime comparison is listed in Table 6. GeoSteiner and BI1S are both very expensive and not suitable for routing estimation in early design stages. On the other hand, HPWL, RMST and FLUTE are all very fast. In particular, RMST is 24% slower than FLUTE and HPWL is 6.22 times faster than FLUTE. The time to generate the table up to $D = 7$ is 50.5 seconds and it only needs to be done once.

We have also performed the experiments when $D$ is set to 8. The error in wirelength estimation over all 18 circuits is 0.59%. Only 0.064% of all degree-8 nets in 18 circuits are non-optimal. The wirelength estimation runtime is 1.25 times slower than that of $D = 7$ (i.e., comparable to RMST runtime). The time to generate the table up to $D = 8$ is 54.5 minutes.

[4] According to [2], the net weights for nets with degree 3 to 8 are 1, 1.08, 1.15, 1.22, 1.28, and 1.34, respectively.

# 7 Conclusion and Discussion

In this paper, we introduced a fast and accurate lookup table based wirelength estimation technique called FLUTE. The table stores the set of POWVs associated with each vertical sequence for low degree nets. We proposed an algorithm based on boundary compaction to generate the sets of POWVs. We designed a 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. The experimental results showed that FLUTE is significantly more accurate than RMST and HPWL. It produces optimal wirelength for all nets with degree 7 or less. It is also faster than RMST.

# References


[1] M. Sarrafzadeh and C. K. Wong. *An Introduction to VLSI Physical Design*. McGraw-Hill, 1996.

[2] Chih-Liang Eric Cheng. RISA: Accurate and efficient placement routability modeling. In *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, pages 690–695, 1994.

[3] L. J. Guibas and J. Stolfi. On computing all northeast nearest neighbors in the L1 metric. *Information Processing Letters*, 17:219–223, 1983.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

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

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

[7] 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.

[8] 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.

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

[10] 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.

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

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

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

[14] 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.