# A Clustering-Based Optimization Algorithm in Zero-Skew Routings

Masato Edahiro
Department of Computer Science, Princeton University
Princeton, NJ 08544-2087, USA
C&C Systems Research Laboratories, NEC Corporation
Miyazaki, Miyamae-ku, Kawasaki 216, Japan

*Abstract*— **A zero-skew routing algorithm with clustering and improvement methods is proposed. This algorithm generates a zero-skew routing in $O(n \log n)$ time for $n$ pins, and it is proven that the order of the total wire length is best possible. Our algorithm achieves 20% reduction of the total wire length on benchmark data compared with the best known algorithm.**

## I. Introduction

Designing zero-skew routings with minimal total wire length is one of the most crucial issues in current and future performance-driven layouts. In synchronous circuits, in order to optimize the clock period, the clock skew needs to be minimized, while the shorter total wire length is required for lower power dissipation.

There have been many researches on routing algorithms to minimize the clock skew [1, 4, 8, 10, 11, 14, 16]. Jackson et al. [10] first proposed a routing method, which generates approximately equi-distant routings. This algorithm was improved to calculate exact equi-distant routings [3, 4, 11]. Although the clock skew in equi-distant routings is usually very small [3], equi-distant routings are not exact zero skew in general. A $RC$-network model was applied to these algorithms, so that exact zero skew routings were finally achieved [16].

Since the exact zero skew has been accomplished, one of unsolved problems is to minimize the total wire length. Two algorithms have been proposed for the total-wire-length minimization. One of the methods is based on a matching [3, 11], while the other algorithm uses a partitioning technique [1, 2]. These algorithms generate minimal total-wire-length routings especially when pins are regularly distributed on chips. In particular, the algorithm [1, 2] is the best known algorithm in the sense of the total wire length with $O(n \log^2 n)$ time complexity for $n$ pins 'under the most realistic case' [2].

In this paper, we first present a bottom-up constructing algorithm [NS] in which the nearest-neighbor selection is used $n-1$ times. We prove that the order of the total wire length in this algorithm is best possible. Moreover, experimental results show that this algorithm calculates 15% shorter routings than the algorithms [1, 2] on benchmark data [10, 16]. This implies that pins are irregularly distributed in actual layouts such as the benchmark data. However, this algorithm requires $O(n^2 \log n)$ time because the Delaunay triangulation [13] needs to be constructed $n-1$ times to find $n-1$ nearest neighbor pairs [1].

Next, we propose a clustering algorithm [CL], in which several pairs of nearest neighbors are selected on a Delaunay triangulation at the same time. We discuss that, in order *not* to increase the total wire length compared with Algorithm [NS], there is a limit for the number of the selected pairs from a triangulation. Our algorithm [CL] determines the number adaptively, so that it is proven that the order of the total wire length is still best possible. Furthermore, we also prove that the algorithm always selects $\Theta(|K|)$ pairs from a Delaunay triangulation for a set $K$, and that the time complexity is improved to $O(n \log n)$. Computational experiments show that the results generated by this clustering method are as good as (even a little better than) those by Algorithm [NS].

In addition, an improvement algorithm is described. For an initial zero-skew routing tree, an exhaustive search is performed for a small subtree rooted by each internal node. Since an optimum solution can be obtained for the small portion of the tree, the improvement carries out a shorter total-wire-length routing. With subtrees of constant size, the improvement takes linear time with respect to the number of pins. Experimental results show that the clustering-based algorithm achieves 20% reduction of the total-wire length compared with the best known algorithm [1, 2] on the benchmark data [10, 16].

## II. Zero-Skew Routing

Given a fan-out terminal $v_r$ and a set of $n$ fan-in terminals $S = \{v_1^s, v_2^s, \ldots, v_n^s\}$, a *clock tree* is defined by a Steiner tree [9] rooted by $v_r$ whose $n$ leaves are $S$ (Fig. 1). We call the fan-out terminal *root* and fan-in terminals *leaves*. A leaf set in the subtree rooted by a node $v$ is denoted by $S_v$. In this paper, clock trees are always binary, though nodes may degenerate and they do not look binary in some cases. Also, we call the nearest Steiner point to the root the *center* denoted by $v_c$.

We assume that the *load capacitance* $C(v_i^s)$ is given for each leaf $v_i^s$, which is usually the gate capacitance of transistors connecting with the leaf. Also, the *load capacitance* $C(v)$ for an internal node $v$ is defined by the total capacitance in $S_v$ that includes wire capacitance as well as gate capacitance.

Then, a *zero-skew routing* for the given root and leaves is defined by a clock tree in which all delay time from the

[1] Another $O(n^2 \log n)$ algorithm was recently proposed in [12], which is equivalent to the minimum-diameter selection in [5]. Comparison between the selection strategies was discussed in [5].

**30th ACM/IEEE Design Automation Conference®**

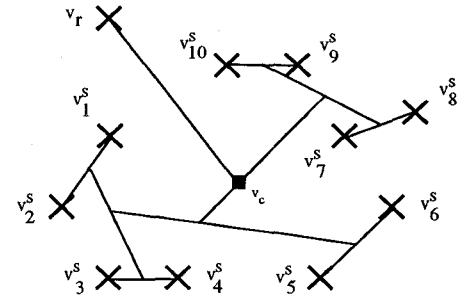



Figure 1: Clock Tree with a root $v_r$ and leaves $S = \{v^s_1, v^s_2, \ldots, v^s_{10}\}$.

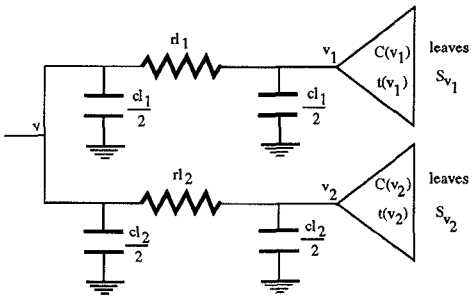



Figure 2: $\pi$-Model for Zero-Skew Merge at $v$.

root to all leaves is equal. From this definition, it is clear that, for any node in the zero-skew routing, all delay from the node $v$ to leaves in $S_v$ should be equal. We call this delay the *delay time* $t(v)$ *for* $v$. For leaves $v^s_i$, $t(v^s_i) = 0$.

An exact zero-skew routing can be constructed in a bottom-up fashion by repeatedly calculating the position of an internal node $v$ from the positions of children $v_1$ and $v_2$ of $v$ using the following equations derived from $\pi$-model (Fig. 2):

$$\begin{aligned} t(v) &= rl_1\left(\frac{cl_1}{2} + C(v_1)\right) + t(v_1) \\ &= rl_2\left(\frac{cl_2}{2} + C(v_2)\right) + t(v_2), \\ C(v) &= C(v_1) + C(v_2) + c(l_1 + l_2), \end{aligned}$$

where $l_1$ ($l_2$) is wire length from $v$ to $v_1$ ($v_2$), and $r$ and $c$ are resistance and capacitance for an unit-length wire [16]. In this paper, all distance is measured in the Manhattan distance.

This operation to determine the location of a node $v$ is called the *zero-skew merge*. At a zero-skew merge, it is clear that $l_1 + l_2 \geq l$, where $l$ is the distance between $v_1$ and $v_2$. Therefore, in order to minimize the total wire length, it is desired to find $v$ such that $l_1 + l_2 = l$. If there is no such a $v$, zero-skew routing algorithms should use a *detour*. Although the number of detours depends on algorithms and the input data, detours hardly appear in actual layouts on 'good' algorithms.

Note that, in Manhattan distance, a set of feasible points for $v$ satisfying the above equations forms a diagonal segment in general [7]. This segment is called the *segment for* $v$. The segment for $v$ can be calculated in constant time even if children are also expressed by diagonal segments [1, 2, 4, 7].

## III. Nearest-Neighbor Selection Algorithm [NS]

In this section, we briefly describe our nearest-neighbor selection algorithm, which is based on a bottom-up construction algorithm presented in [4, 5]. Then, we show that the order of the total wire length of zero-skew routings generated by Algorithm [NS] is best possible.

### A. Algorithm [NS]

Let $K$ be a set of points or diagonal segments. Initially, $K = S$. Algorithm [NS] has two phases, *Find_Center* and *Embedding*. In *Find_Center*, segments for all internal nodes are calculated in a bottom-up fashion, and then, in *Embedding*, the best position of each node is determined in a top-down fashion.

Algorithm *Find_Center* takes the nearest neighbor pair $v_1$ and $v_2$ from $K$, calculates a segment for $v$ from $v_1$ and $v_2$ using the zero-skew merge, and put the segment into $K$. After $n-1$ operations, $K$ has only one element that is the segment for the center $v_c$.

Next, Algorithm *Embedding* determines the best position for each node in the reverse order of Algorithm *Find_Center*. First, the position of the center $v_c$ is determined on the segment for $v_c$ by routing from the root $v_r$ in the shortest way. Once the position of a node is determined, we can easily calculate the positions of its children so as to satisfy the zero-skew merge equations. Therefore, all positions of nodes are calculated in a top-down fashion starting from the position of $v_c$.

The algorithm is stated as follows:

**Algorithm** *Find_Center*([*NS*])

Step 1: $K := S$.

Step 2: If $|K| = 1$, stop. (The element in $K$ is the segment for the center $v_c$.) Otherwise, choose the nearest pair of points (or diagonal segments) $v_1$ and $v_2$ from $K$.

Step 3: (Zero-Skew Merge) Calculate the segment for $v$ from $v_1$ and $v_2$ using the zero-skew merge. Delete $v_1$ and $v_2$ from $K$, and add $v$ to $K$. Go to Step 2.

**Algorithm** *Embedding*

Step 1: Determine the center $v_c$ on the segment for $v_c$ by selecting the nearest point to the root $v_r$. Route from $v_r$ to $v_c$.

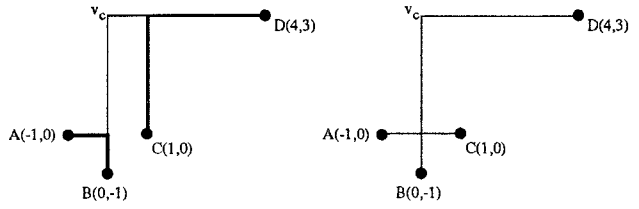
Step 2: *local_embedding*($v_c$)

**procedure** *local_embedding*($v$)

Step 1: If $v$ has no child, return.

Step 2: Let $v_1$ and $v_2$ be the children of $v$. For $i = \{1, 2\}$, determine a point $v_i$ on the segment for $v_i$ so as to satisfy the zero-skew merge equations. Route from $v$ to $v_i$.

Step 3: *local_embedding*($v_1$), *local_embedding*($v_2$).




(a) Matching-based ($L = 12$) (b) Nearest-neighbor ($L = 10$)

Figure 3: Matching-Based vs. Nearest-Neighbor Selection Algorithm ($L$ = total wire length).

### B. Properties

The distance between the nearest neighbor pair in Step 2 of Algorithm *Find_Center* is bounded in the next property. Proofs are shown in [6].

**Property 1** *Suppose that a set $K$ of $|K|$ ($> 1$) points or segments has diameter $D$. Then, the distance $d^*$ between the nearest neighbor pair in $K$ satisfies*

$$d^* \leq \frac{D}{\sqrt{|K|-1}}.$$

Then, the total wire length for Algorithm [NS] is bounded using Property 1.

**Lemma 1** *If there is no detour, the total wire length for Algorithm [NS] is $O(\sqrt{n}D)$, where $D$ is diameter of $S$.*

Next, we show that this bound is best possible and the same order as the total edge length for the minimum Steiner tree [9] for $S$. It is clear that, for a leaf set, the total wire length for the zero-skew routing is not shorter than that for the minimum Steiner tree. Therefore, we have only to show that the total edge length for the minimum Steiner tree is $\Omega(\sqrt{n}D)$.

Consider $S^+$ in which $n$ points lie on $\sqrt{n} \times \sqrt{n}$ grid whose interval is $\frac{D}{\sqrt{n}}$. It is easy to see that the total edge length of the minimum Steiner tree for $S^+$ is $\Omega(\sqrt{n}D)$. Note that it was proven that this lower bound is tight even in average cases for the Euclidean distance [15].

**Lemma 2** *The order of the total wire length of a zero-skew routing generated by Algorithm [NS] is best possible.*

The matching-based algorithm has the same bound [3]. When pins are regularly distributed, this algorithm generates a good results due to the perfect matching. However, in real layouts like benchmark data, the perfect matching could cause long wires. In Fig. 3, the routing by the matching method is 20% longer than that by [NS]. In this figure, a zero-skew routing is approximated by an equidiatant routing.

Next, the time complexity is evaluated. In Algorithm *Find_Center*, Steps 2 and 3 are repeated $n - 1$ times. Step 2 requires $O(n^2)$ time in a naive implementation. This time complexity can be reduced to $O(n \log n)$ by a generalized Delaunay triangualtion [13] which always has an edge connecting the nearest-neighbor pair. This leads to the next lemma.

**Lemma 3** *The time complexity of Algorithm [NS] is $O(n^2 \log n)$.*

As we have seen above, although Algorithm [NS] has good properties, the time complexity is more expensive than the best known algorithm because Algorithm [NS] reconstructs the Delaunay triangulation every time the nearest neighbor pair needs to be found.

## IV. Clustering-Based Algorithm [CL]

In this section, we propose a clustering-based algorithm, in which several nearest-neighbor pairs are selected from a Delaunay triangulation at the same time, so that the time complexity is reduced without increasing the total wire length.

In Algorithm [NS], we reconstruct the Delaunay triangulation every time a pair $(v_1, v_2)$ is merged into a node $v$ (i.e., the segment for a node $v$ is calculated from points (or segments) for nodes $v_1$ and $v_2$). Now, consider the next merge. In many cases especially where $|K|$ is large, the next nearest-neighbor pair $(v_1', v_2')$ does not include $v$. We say for this situation that the next pair is *independent* from the previous pair. In this independent case, since the previous Delaunay triangulation also has an edge connecting $(v_1', v_2')$, we do not have to reconstruct the Delaunay triangulation.

From this observation, it is expected that merging several independent pairs on a Delaunay triangulation will not increase the total wire length, and that this improvement may reduce the time complexity. However, as we saw in Fig. 3, the total wire length will increase if most nodes are merged in a triangulation. Thus, there is a limit number for zero-skew merges in a triangulation in order *not* to increase the total wire length compared with Algorithm [NS]. In our clustering-based algorithm, the number of zero-skew merges for a Delaunay triangulation is adaptively varied. We will prove that the time complexity is reduced without increasing the order of the total wire length.

We first construct a graph called the nearest-neighbor graph, which maintains the nearest-neighbor point (or segment) to each point (or segment) in $K$. In zero-skew merges, $|K|/k$ nearest-neighbor pairs are taken from the graph in the non-decreasing order of distance of pairs, where $k$ ($> 1$) is a constant. Typically, $2 \leq k \leq 4$. For each pair, if either of the points (or segments) has already been merged, we consider the pair is *not* independent from the previous pairs, and the pair is discarded.

There are two significant characteristics in this clustering-based algorithm: 1) since the nearest-neighbor graph is used, pairs of points (or segments) apart from each other are not taken into account; 2) since only independent pairs are merged, several nearest neighbor pairs are selected without increasing the total wire length nor reconstructing the Delaunay triangulation. Therefore, it is expected that the total wire length will not increase. We will prove that the order of the total wire length is still best possible for Algorithm [CL]. In addition, experimental results show that Algorithm [CL] generates as good results as Algorithm [NS].

Furthermore, it will also be proven that the number of merges in a Delaunay triangulation is $\Theta(|K|)$ in any case, though the number strongly depends on the distribution

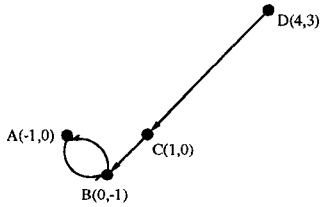



Figure 4: Nearest-Neighbor Graph.

of points or segments. From this property, we prove that the time complexity is $O(n \log n)$.

Now, we state our algorithm. First, we define the nearest-neighbor graph $G(K, E)$ on a set $K$ of points or diagonal segments.

*A. Nearest Neighbor Graph*

Let $K$ be a set of points or diagonal segments. The *nearest-neighbor graph* $G(K, E)$ for $K$ is defined by the weighted directed graph such that

**i)** Each node $v \in K$ has exact one out-going edge, so that $|E| = |K|$,

**ii)** $(v_i, v_j) \in E$ for $v_i, v_j \in K \Rightarrow v_j$ is the nearest to $v_i$ (if there is a tie, one of the tie edges is arbitrarily selected),

**iii)** Each edge $e = (v_i, v_j) \in E$ has a weight $w(e)$ that is the distance between $v_i$ and $v_j$.

From the definition, $G(K, E)$ is not unique when a node has more than one nearest neighbors. However, it is easy to see that a Delaunay triangulation for $K$ should contain one of the nearest-neighbor graphs for $K$ as a subgraph. The example of the nearest-neighbor graph for the point set in Fig. 3 is depicted in Fig. 4.

Let the sequence $\{e_0, e_1, \ldots, e_{|K|-1}\}$ be the edges in $E$ sorted by their weights in non-decreasing order. Then, the following property characterizes the weight of $e_i$.

**Property 2** *For* $0 \leq \forall i < |K| - 1$,

$$w(e_i) \leq \frac{D}{\sqrt{|K| - i - 1}},$$

*where $D$ is diameter of $K$.*

*B. Algorithm [CL]*

Now, we state the clustering-based algorithm [CL]. Although this algorithm also has two phases (*Find_Center* and *Embedding*), we omit *Embedding* phase because it is exactly the same as that in Algorithm [NS].

In *Find_Center* phase, we use a parameter $k > 1$. Also, we use a function $s(a, b, c)$ where $a \leq c$ defined by

$$s(a, b, c) = \begin{cases} a; & a \geq b, \\ b; & a < b < c, \\ c; & b \geq c. \end{cases}$$

**Algorithm** *Find_Center*([*CL*])

Step 1: $K := S$.

Step 2: If $|K| = 1$, stop. (The element in $K$ is the segment for the center $v_c$.) Otherwise, construct the nearest-neighbor graph $G(K, E)$ on $K$, and sort edges in $E$ by their weights in the non-decreasing order.

Step 3: Repeat Step 3.1 $s(1, |K|/k, |K| - 1)$ times. Then, go to Step 2.

Step 3.1: Take the smallest weight edge $(v_1, v_2)$ from $E$, and delete the edge from $E$. If neither $v_1$ nor $v_2$ has already been deleted from $K$, do Step 3.2.

Step 3.2: (Zero-Skew Merge) Calculate the segment for $v$ from $v_1$ and $v_2$ using the zero-skew merge. Delete $v_1$ and $v_2$ from $K$, and add $v$ to $K$.

*C. Properties*

In this section, we show some properties for Algorithm [CL]. Again, proofs are shown in [6]. First, we evaluate the total wire length. The next lemma is proven from Property 2.

**Lemma 4** *If there is no detour, the total wire length for Algorithm [CL] is $O(\sqrt{n}D)$, where $D$ is diameter of $S$.*

**Lemma 5** *The order of the total wire length of a zero-skew routing generated by Algorithm [CL] is the best possible.*

Next, we calculate the time complexity of Algorithm [CL]. It is easy to see that the following property is satisfied for nearest-neighbor graphs in Algorithm [CL].

**Property 3** *For a pair $(v_1, v_2)$ merged in Step 3.2, at most $c_e$ edges are in-coming to either $v_1$ or $v_2$ in $G(K, E)$, where $c_e$ is a constant.*

This property shows that, when $p$ pairs are merged in Step 3.2, at most $(c_e - 1)p$ edges will be ignored in Step 3.1. This observation leads to the next lemma:

**Lemma 6** *For a set $K$, at least* $\max(1, |K|/(c_e k))$ *pairs are merged in Step 3.2 of Algorithm Find_Center.*

Clearly, Step 2 requires $O(|K| \log |K|)$ time, and Step 3 can be performed in $O(|K|)$. From Lemma 6, at the $i$-th iteration, $|K|$ is at most $\left(1 - \frac{1}{c_e k}\right)^{i-1} n$. Consequently, the complexity for our algorithm is evaluated.

**Lemma 7** *The time complexity of Algorithm [CL] is $O(n \log n)$.*

## V. Improvement Algorithm

In this section, we present an improvement algorithm for a zero-skew routing previously designed.

Let $m$ be a positive integer. Typically, $4 \leq m \leq 6$. Also, for an internal node $v$, let a tree $T_v$ be a subtree, which has at most $2m - 1$ nodes and is rooted by $v$, on the initial zero-skew routing. $N_v$ denotes a set of 'leaves' of $T_v$. (Note that $N_v$ may be internal nodes on the initial zero-skew routing.) Then, the improvement algorithm for

Table 1: Total Wire Length for four zero-skew routing algorithms, [1], [NS], [CL], [CL+I6].

| | #pins | [1] | [NS] | [CL] | [CL+I6] | ([1]) |
|---|---|---|---|---|---|---|
| prim1 | 269 | 177928 | 138787 | 132980 | 129185 | (0.73) |
| prim2 | 603 | 392348 | 366646 | 334107 | 303994 | (0.77) |
| r1 | 267 | 1582498 | 1485399 | 1421307 | 1253347 | (0.79) |
| r2 | 598 | 3060399 | 2828584 | 2627494 | 2483754 | (0.81) |
| r3 | 862 | 3953785 | 3645674 | 3550494 | 3193801 | (0.81) |
| r4 | 1903 | 8039353 | 7705858 | 6794605 | 6499660 | (0.81) |
| r5 | 3101 | 12129536 | 10402178 | 10195581 | 9723726 | (0.80) |

$N_v$ uses an exhaustive search, and calculates an optimum sequence of zero-skew merge pairs such that the sequence generates the minimum total-wire-length zero-skew routing $T'_v$ whose leaves are $N_v$, without changing positions, $C(\cdot)$'s, and $t(\cdot)$'s for $N_v$. The root of $T'_v$ is the new location of $v$.

This improvement for a node is applied to all nodes in a zero-skew routing, and the improvement for all internal nodes is iterated until no improvement is made. The time complexity for an improvement on a node is a function only of $m$, though the function is exponential. Therefore, since $m$ is constant, the improvement for all nodes requires $O(n)$ time. The number of iterations of the improvement is experimentally less than 20.

## VI. Experimental Results

We show experimental results for four algorithms on benchmark data prim1-prim2 [10] and r1-r5 [16]. The first algorithm [1] is the best known for the total wire length. The other three algorithms are [NS], [CL], and [CL+I6] that applies the improvement with $m = 6$ to [CL]. For [CL] and [CL+I6], we tested 9 values for the parameter $k$ ($k = 2 + 0.25i$ $(0 \leq i \leq 8)$), and only the best results are shown in Table 1. It is observed that Algorithm [CL] is as good as (even a little better than) Algorithm [NS], and that [CL+I6] achieves 20% reduction of the total wire length compared with [1].

## VII. Conclusion

A clustering-based algorithm has been proposed for zero-skew routings. We first showed that a nearest-neighbor selection algorithm [NS] generates shorter total-wire-length routings than previous algorithms. The clustering method reduces the time complexity of Algorithm [NS] without increasing the total wire length. An improvement algorithm helps to obtain better solutions using a localized exhaustive search. Our algorithm accomplished 20% reduction of the total wire length on benchmark data.


## Acknowledgment

The author would like to thank A. Kahng and K. Boese of UCLA and R. S. Tsay of ArcSys Inc. for providing benchmark data.


# References


[1] T. H. Chao, Y. C. Hsu, and J. M. Ho, "Zero skew clock net routing," *Proc. of the 29th Design Automation Conference*, 1992, pp.518-523.

[2] T. H. Chao, Y. C. Hsu, J. M. Ho, K. D. Boese, and A. B. Kahng, "Zero skew clock routing with minimum wirelength," *IEEE Transactions on Circuits and Systems*, in press.

[3] J. C. Cong, A. B. Kahng, and G. Robins, "Matching-based methods for high-performance clock routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, in press.

[4] M. Edahiro, "Minimum skew and minimum path length routing," *NEC Res. & Develop.*, Vol. 32 (1991), No. 4, pp.569-575.

[5] M. Edahiro and T. Yoshimura, "Minimum path-length equidistant routing," *Proc. of 1992 IEEE Asia-Pacific Conference on Circuits and Systems*, 1992, pp.41-46.

[6] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," *Technical Report CS-TR-416-93*, Department of Computer Science, Princeton University, 1993.

[7] M. Edahiro, "Equi-spreading tree in Manhattan distance," unpublished.

[8] E. G. Friedman and S. Powell, "Design and analysis of a hierarchical clock distribution system for synchronous standard cell/macrocell VLSI," *IEEE Journal of Solid-State Circuits*, Vol. SC-21 (1986), No.2, pp.240-246.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, New York, 1979.

[10] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, "Clock routing for high-performance ICs," *Proc. of the 27th Design Automation Conference*, 1990, pp.573-579.

[11] A. Kahng, J. Cong, and G. Robins, "High-performance clock routing based on recursive geometric matching," *Proc. of the 28th Design Automation Conference*, 1991, pp.322-327.

[12] Y. M. Li and M. A. Jabri, "A zero-skew clock routing scheme for VLSI circuits," *Proc. of 1992 International Conference on Computer-Aided Design*, 1992, pp.458-463.

[13] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction.* Springer-Verlag, New York, New York, 1985.

[14] P. Ramanathan and K. G. Shin, "A clock distribution scheme for non-symmetric VLSI circuits," *Proc. of 1989 International Conference on Computer-Aided Design*, 1989, pp.398-401.

[15] J. M. Steele, "Growth rates of Euclidean minimal spanning trees with power weighted edges," *Annals of Probability*, Vol. 16 (1988), pp.1767-1787.

[16] R. S. Tsay, "Exact zero skew," *Proc. of the 1991 International Conference on Computer-Aided Design*, 1991, pp.336-339.