A Clustering-Based Optimization Algorithm in Zero-Skew Routings

Masato Edahiro [1]

Research Report CS-TR-416-93
March 1993

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 and 15%-50% improvement of the delay time on benchmark data compared with the best known algorithm.

[1]Department of Computer Science, Princeton University and C&C Systems Research Laboratories, NEC Corporation

1 Introduction

Designing zero-skew routings with minimal delay time is one of the most crucial issues in current and future performance-driven layouts. In synchronous circuits, the clock skew t_{skew} limits the clock period t_c together with data path delay t_d and other constant factors t_{const}. The following equation formulates the clock period:

$$t_c = t_{skew} + t_d + t_{const}.$$

Therefore, in order to optimize the clock period, the clock skew needs to be minimized. In addition, in designing high-clock rate circuits, the dynamic response of MOS transistors ought to be cared about, which can be formulated in a linear combination of the total wire length and the propagation delay time [5].

There have been many researches on minimizing the clock skew [1, 4, 6, 7, 10, 11, 13, 15, 16]. As for routing algorithms, 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 delay time. 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 needs to be constructed $n-1$ times to find $n-1$ nearest neighbor pairs.

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 $O(|K|)$ 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.

Also, we remark that our clustering-based algorithm can work with any distance function as an objective function instead of the nearest neighbor for the total-wire-length optimization. In particular, it is worth while to see that a delay-time estimation function in [5] can be an objective function for our algorithm to minimize the delay time.

Experimental results show that the clustering-based algorithm achieves 20% reduction of the total-wire length and 15%-50% improvement of the delay time compared with the best known algorithm [1, 2] on the benchmark data [10, 16].

2 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 [8] rooted by v_r whose n leaves are S (Fig. 1). We call the fan-out terminal *root* and fan-in terminals *leaves*. A set of leaves in the subtree rooted by a node v is called *leaves connecting to* v and 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 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_i^s, $t(v_i^s) = 0$.

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

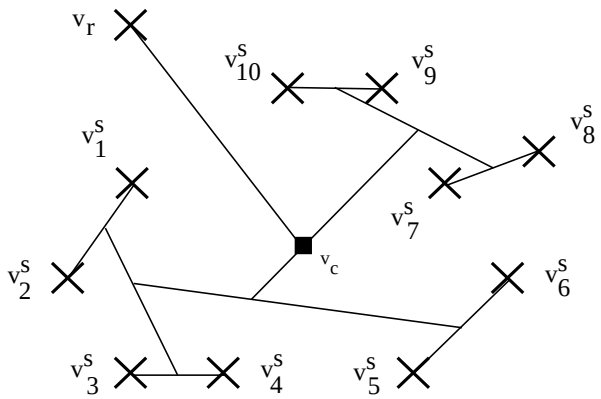


Figure 1: Clock Tree with a root v_r and leaves $S = \{v_1^s, v_2^s, \ldots, v_{10}^s\}$.

equations derived from π-model [5, 16] (Fig. 2)[2]:

$$\begin{aligned} t(v) &= \frac{rl_1}{w_1}\left(\frac{cl_1w_1}{2} + C(v_1)\right) + t(v_1) = \frac{rl_2}{w_2}\left(\frac{cl_2w_2}{2} + C(v_2)\right) + t(v_2), \\ C(v) &= C(v_1) + C(v_2) + c(l_1w_1 + l_2w_2), \end{aligned}$$

where l_1 (l_2) and w_1 (w_2) are length and width of the wire from v to v_1 (v_2), and r and c are wire resistance and capacitance for an unit length and width wire.

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 delay time, 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 the 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. 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].

3 Nearest-Neighbor Selection Algorithm [NS]

In this section, we present an algorithm with nearest-neighbor selections. In addition, we show that the order of the total wire length of zero-skew routings generated by Algorithm [NS] is best possible. In proofs, we use the minimum Steiner trees and the minimum spanning trees, which are

[2] In those formulae, we use a generalized model in which wire widths are variable [5]

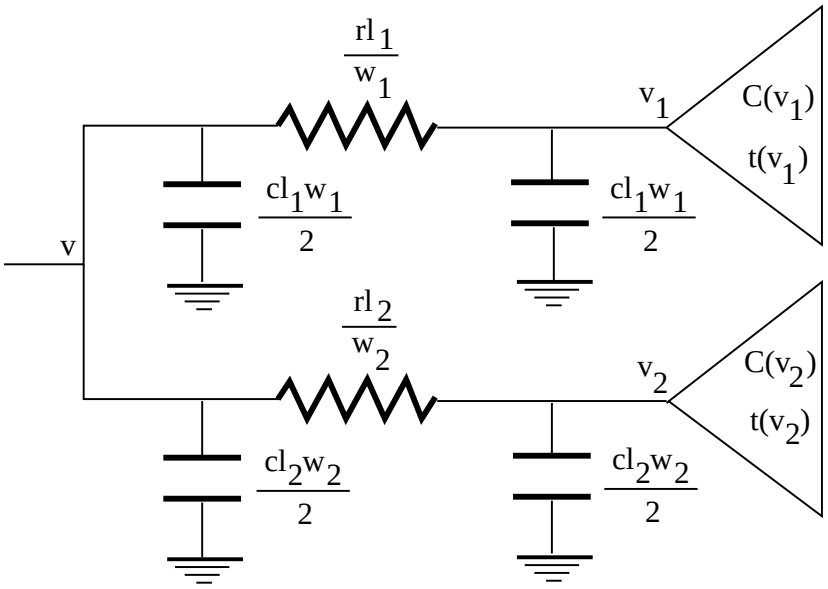


Figure 2: π-Model for Zero-Skew Merge at v.

defined on the complete graph on S whose edge $e = (v_i^s, v_j^s)$ has a weight $w(e)$ that is the distance between v_i^s and v_j^s measured by the Manhattan distance.

3.1 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 then put the segment into K. After repeating these operations $n-1$ times, 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. Since, after determining the position of a node, we can easily calculate the positions of its children so as to satisfy the zero-skew merge equations, 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 by selecting the nearest point to the root v_r on the segment for v_c. 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, 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: $local_embedding(v_1)$, $local_embedding(v_2)$.

3.2 Properties

The distance between the nearest neighbor pair in Step 2 of Algorithm $Find_Center$ is bounded in the next property. We assume that the distance is measured by the Manhattan distance.

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

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

Proof: Consider a set of $|K|$ circles of diameter d^* centered by each point or one of the endpoints of each segment. Since the distance between any pair in K is not less than d^*, these circles do not intersect each other.

In addition, it is clear that these circles are contained by a circle of diameter $D + d^*$. Therefore, sum of area of $|K|$ circles cannot exceed the area of the circle of diameter $D + d^*$, i.e.,

$$|K|\frac{(d^*)^2}{2} \leq \frac{(D + d^*)^2}{2}.$$

It follows that

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

■

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

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

Proof: When points (or segments) v_1 and v_2 are merged in Step 3 of $Find_Center$, the total wire length increases by $l_1 + l_2$, which is the distance between v_1 and v_2 if there is no detour. Thus, from Property 1, $l_1 + l_2 \leq \frac{D}{\sqrt{|K|}-1}$. It follows that the total wire length L is

$$\begin{aligned} L &\leq \sum_{|K|=n}^{2} \frac{D}{\sqrt{|K|} - 1} \\ &\leq \sum_{|K|=n}^{2} \frac{D}{\sqrt{|K|}} \frac{\sqrt{2}}{\sqrt{2} - 1} \\ &\leq (2 + \sqrt{2})D \int_1^n \frac{1}{\sqrt{x}} dx \\ &= O(\sqrt{n}D). \end{aligned}$$

■

Next, we show that this bound is tight and the same order as the total edge length for the minimum Steiner tree for S.

The following collorary is clear from Lemma 1.

Collorary 1 *The total edge length for the minimum Steiner tree for S is $O(\sqrt{n}D)$.*

Since the Steiner ratio[3] is 2/3 in the Manhattan distance [9], the following collorary is true.

Collorary 2 *The total edge length for the minimum spanning tree for S is $O(\sqrt{n}D)$.*

Consider S^+ in which n points lie on $\sqrt{n} \times \sqrt{n}$ grid whose interval is $\frac{D}{\sqrt{n}}$. It is clear that the total edge length of a spanning tree for S^+ is $\Omega(\sqrt{n}D)$[4]. Therefore,

Lemma 2 *The total edge length for the minimum spanning tree for S is $\Theta(\sqrt{n}D)$.*

Using Hwang's theorem again, the total edge length for the minimum Steiner tree for S^+ is $\Omega(\sqrt{n}D)$. Thus,

Collorary 3 *The total edge length for the minimum Steiner tree for S is $\Theta(\sqrt{n}D)$.*

Consequently, since the total wire length for a zero-skew routing cannot be less than that in the minimum Steiner tree, the following lemma is proven.

Lemma 3 *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][5].

Next, the time complexity of this algorithm is evaluated.

Lemma 4 *The time complexity of Algorithm [NS] is $O(n^2 \log n)$.*

Proof: In Algorithm *Find_Center*, Steps 2 and 3 are repeated $n-1$ times. Step 2 requires $O(n^2)$ time in a naive implementation, which can be reduced to $O(n \log n)$ by a generalized Voronoi diagram [12] because the Delaunay triangulation, the dual of the Voronoi diagram, always has an edge connecting the nearest-neighbor pair.

Since Step 3 can be performed in constant time, the time complexity for Algorithm *Find_Center* is $O(n^2 \log n)$. It is easy to see that Algorithm *Embedding* requires $O(n)$.

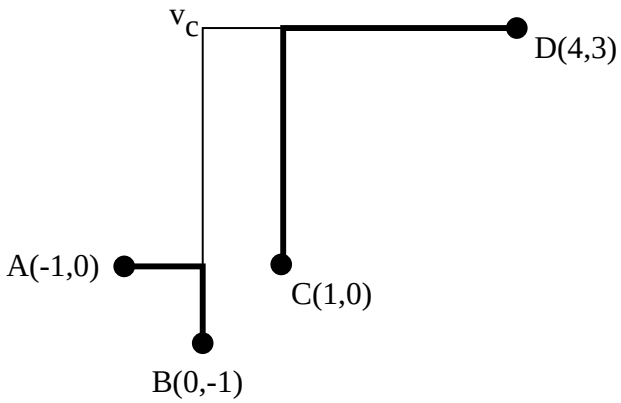
Consequently, the time complexity of Algorithm [NS] is $O(n^2 \log n)$. ∎

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.

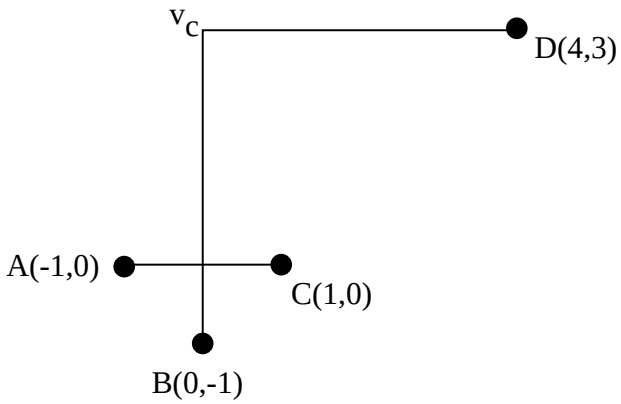
[3] The Steiner ratio is the lower bound of the ratio of the total edge length of the minimum Steiner tree to that of the minimum spanning tree on the same point set.

[4] The average case bound for the total edge length of a spanning tree is discussed in remarks.

[5] In this figure, a zero-skew routing is approximated by an equi-diatant routing.



(a) Routing by the matching-based algorithm (length = 12)
(Bold lines show the minimum weight perfect matching.)



(b) Routing by the nearest-neighbor selection algorithm [NS] (length = 10)

Figure 3: Matching-Based Algorithm vs. Nearest-Neighbor Selection Algorithm for leaves $\{A, B, C, D\}$.

4 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 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 $O(|K|)$ in any case, though the number strongly depends on the distribution 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.

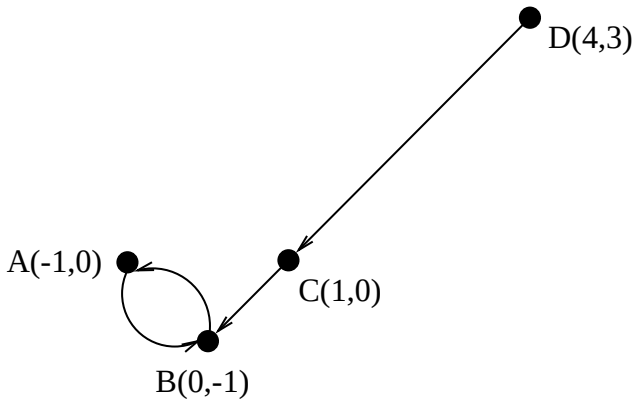


Figure 4: Nearest-Neighbor Graph for the example in Fig. 3.

4.1 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. Therefore, the nearest-neighbor graph can be constructed from the Voronoi diagram for K. 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 the diameter of* K.

Proof: Let a set $K_i \subseteq K$ $(0 \leq i \leq |K|)$ be

$$\begin{aligned} K_0 &= \emptyset, \\ K_{i+1} &= K_i \cup \{v_i\}, \text{ s.t. } e_i = (v_i, v_i'), \text{ for } 0 \leq i \leq |K| - 1. \end{aligned}$$

Also, $\bar{K_i} = K - K_i$.

It is clear that, for $0 \leq \forall i < |K| - 1$, the distance between any pair of points or segments in $\bar{K_i}$ is not less than $w(e_i)$. Since $|\bar{K_i}| = |K| - i$, from Property 1,

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

■

4.2 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 for 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.

4.3 Properties

First, we evaluate the total wire length.

Lemma 5 *If there is no detour, the total wire length for Algorithm [CL] is $O(\sqrt{n}D)$, where D is the diameter of S.*

Proof: Since only $s(1, |K|/k, |K|-1)$ edges are selected in Step 3 from a nearest-neighbor graph, by Property 2, the length of each selected edge e is bounded by

$$\begin{aligned} w(e) \leq w(e_{s(1,|K|/k,|K|-1)-1}) &\leq \frac{D}{\sqrt{s(2,(1-\frac{1}{k})\,|K|+1,|K|)}-1} \\ &= \frac{D}{\sqrt{s'(|K|)}-1}, \end{aligned}$$

where $s'(x) = s(2, (1-\frac{1}{k})\,x+1, x)$.

Let p_i $(i = 1, \ldots, q)$ be the number of pairs merged in Step 3.2 at the i-th iteration of Steps 2-3. Clearly, $\sum_{j=1}^{q} p_j = n-1$, and, at the i-th iteration, $|K| = n - \sum_{j<i} p_j$. Thus, the total edge length is evaluated as follows:

$$\begin{aligned} L &\leq \sum_{i=1}^{q} \frac{p_i D}{\sqrt{s'(n-\sum_{j<i} p_j)}-1} \\ &\leq \sum_{|K|=n}^{2} \frac{D}{\sqrt{s'(|K|)}-1} \\ &= \sum_{|K|=n}^{2} \frac{D}{\sqrt{s(2,(1-\frac{1}{k})\,|K|+1,|K|)}-1} \\ &\leq \frac{2+\sqrt{2}}{\sqrt{1-1/k}} D \int_1^n \frac{1}{\sqrt{x}} dx \\ &= O(\sqrt{n}D) \end{aligned}$$

■

Lemma 6 *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 follows the next lemma:

Lemma 7 *For a set K, at least $\max(1, |K|/(c_e k))$ pairs are merged in Step 3.2 of Algorithm Find_Center.*

Now, we evaluate the complexity.

Lemma 8 *The time complexity of Algorithm [CL] is $O(n \log n)$.*

Proof: Clearly, Step 2 requires $O(|K| \log |K|)$ time, and Step 3 can be performed in $O(|K|)$. From Lemma 7, at the i-th iteration, $|K|$ is at most $\left(1 - \frac{1}{c_e k}\right)^{i-1} n$. Thus, the time complexity $T(n)$ is

$$\begin{aligned} T(n) &\leq \sum_{i=0}^{\infty} \left[c' \left(1 - \frac{1}{c_e k}\right)^i n \log \left(\left(1 - \frac{1}{c_e k}\right)^i n \right) \right] \quad (c' \text{ is a constant.}) \\ &\leq c' n \log n \sum_{i=0}^{\infty} \left(1 - \frac{1}{c_e k}\right)^i \\ &= c_e c' k n \log n \\ &= O(n \log n). \end{aligned}$$

■

5 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 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

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)

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.

6 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].

7 Remarks

7.1 Average case bound for the total wire length

In the Euclidean distance, the worst case bound $O(\sqrt{n}D)$ for the total wire length of the minimum spanning trees is also the average case bound when points are uniformly distributed [14]. We will study the average case bound for the Manhattan distance and the approximation ratio of Algorithm [NS] and [CL] to the optimum solutions in the future work.

7.2 Delay Time Optimization

Although we have mainly discussed the total-wire-length minimization using the distance of points as an objective function to be minimized, it is easy to see that our clustering-based algorithm

also works on any objective function that is a distance function on the plane.

We proposed a delay-time estimation function in [5]. Since this is also a distance function, this delay-time estimation function can be used as an objective function to be minimized in our clustering-based algorithm. Table 2 shows the results, in which [DM+WW] is our clustering-based algorithm with the delay-minimization objective fuction + the wire width optimization [5]. It is observed that our algorithm with the delay minimization technique improves 15%-50% of the total delay time.

For the delay time optimization, we have not proven any optimality because of its complicated objective function. Fortunately, it is also observed in Table 2 that the total wire length for the delay minimization is within 10% of that for the total-wire-length minimization in Table 1. If the total wire length for the delay minimization is bounded as we have proven for the total-wire-length minimization, the optimality of our algorithm will be proven for the delay time minimization as well.

8 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 and 15%-50% improvement of the delay time on 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*, to appear.

[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*, to appear.

[4] M. Edahiro: Minimum Skew and Minimum Path Length Routing. *NEC Res. & Develop.*, Vol. 32 (1991), No. 4, pp.569-575.

Table 2: Total Wire Length [TWL], Propagation Delay Time [PD], Total Delay Time [TD], and clock skew [SKW] for the algorithms [1] and [DM+WW]. (SPICE (LEVEL3) was used in the simulation.)

	#pins		[1]		[DM+WW]		
			estimated	simulated	estimated	sim. (ratio to [1])	
prim1	269	TWL	177928		135563		
		PD	2.69ns	3.07ns	0.29ns	0.30ns	(0.10)
		TD	6.56ns	6.53ns	5.58ns	5.62ns	(0.86)
		SKW	0.00ns	0.00ns	0.00ns	0.00ns	
prim2	603	TWL	392348		332791		
		PD	11.24ns	14.15ns	0.77ns	0.79ns	(0.06)
		TD	16.99ns	17.27ns	12.70ns	12.54ns	(0.73)
		SKW	0.00ns	0.01ns	0.00ns	0.00ns	
r1	267	TWL	1582498		1276993		
		PD	1.61ns	2.01ns	0.65ns	0.73ns	(0.36)
		TD	2.49ns	2.66ns	1.88ns	2.01ns	(0.76)
		SKW	0.00ns	0.00ns	0.00ns	0.00ns	
r2	598	TWL	3060399		2549132		
		PD	4.52ns	5.53ns	1.44ns	1.60ns	(0.29)
		TD	5.61ns	5.99ns	4.02ns	4.10ns	(0.68)
		SKW	0.00ns	0.00ns	0.00ns	0.00ns	
r3	862	TWL	3953785		3474596		
		PD	6.43ns	7.78ns	1.37ns	1.48ns	(0.19)
		TD	7.71ns	8.23ns	5.41ns	5.43ns	(0.66)
		SKW	0.00ns	0.01ns	0.00ns	0.01ns	
r4	1903	TWL	8039353		6673776		
		PD	18.94ns	20.24ns	3.26ns	3.56ns	(0.18)
		TD	18.58ns	20.59ns	11.33ns	11.21ns	(0.54)
		SKW	0.00ns	0.05ns	0.00ns	0.01ns	
r5	3101	TWL	12129536		9952239		
		PD	35.32ns	35.56ns	5.32ns	5.83ns	(0.16)
		TD	31.48ns	35.88ns	17.60ns	17.30ns	(0.48)
		SKW	0.00ns	0.13ns	0.00ns	0.01ns	

[5] M. Edahiro: Delay Minimization for Zero-Skew Routing. *Technical Report CS-TR-415-93*, Department of Computer Science, Princeton University, 1993.

[6] J. P. Fishburn: Clock Skew Optimization. *IEEE Trans. on Computers*, Vol. 39 (1990), No. 7, pp.945-951.

[7] E. G. Friedman and S. Powell: Design and Analysis of a Hierarchical Clock Distribution System for Synchronous Standard Cell/Macrocell VLSI. *IEEE J. of Solid-State Circuits*, Vol. SC-21 (1986), No.2, pp.240-246.

[8] 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.

[9] F. K. Hwang: On Steiner Minimal Trees with Rectilinear Distance. *SIAM J. on Applied Mathematics*, Vol. 30 (1976), No. 1, pp.104-114.

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

[13] 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.

[14] J. M. Steele: Growth Rates of Euclidean Minimal Spanning Trees with Power Weighted Edges. *Annals of Probability*, Vol. 16 (1988), pp.1767-1787.

[15] M. Shoji: Elimination of Process-Dependent Clock Skew in CMOS VLSI. *IEEE J. of Solid-State Circuits*, Vol. SC-21 (1986), No. 5, pp.875-880.

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