

The Associative-Skew Clock Routing Problem^{*†}

Yu Chen, Andrew B. Kahng, Gang Qu, and Alexander Zelikovsky[‡]

UCLA Department of Computer Science, Los Angeles, CA 90095-1596

[‡]Department of Computer Science, Georgia State University, Atlanta, GA 30303
{yuchen,abk,gangqu}@cs.ucla.edu, alexz@cs.gsu.edu

Abstract

We introduce the *associative skew* clock routing problem, which seeks a clock routing tree such that zero skew is preserved only within identified *groups* of sinks. The associative skew problem is easier to address within current EDA frameworks than useful-skew (skew-scheduling) approaches, and defines an interesting tradeoff between the traditional zero-skew clock routing problem (one sink group) and the Steiner minimum tree problem (n sink groups). We present a set of heuristic building blocks, including an efficient and optimal method of merging two zero-skew trees such that zero skew is preserved within the sink sets of each tree. Finally, we list a number of open issues for research and practical application.

1 Introduction

The design of the clock distribution network is one of the most critical tasks in high-performance, deep-submicron VLSI system design. A clock routing instance consists of (i) a set of sink register locations $S = \{s_1, s_2, \dots, s_n\} \subset \mathcal{R}^2$, and optionally (ii) a binary-tree *connection topology* G rooted at source s_0 , with n leaves corresponding to the sinks in S . The generic *clock routing problem* seeks to define and embed the topology into the Manhattan plane – i.e., construct a *clock tree* T that maps each internal node $v \in G$ to a location $l(v)$ in the Manhattan plane.

Among the constraints on the clock routing solution, *skew* is the most fundamental. When the clock tree is rooted at the source, any edge between a parent node p and its child v may be identified with the child node, i.e., we denote this edge as e_v . If $t(u, v)$ denotes the signal delay between nodes u and v , then the *skew* of clock tree T is given by

$$\begin{aligned} \text{skew}(T) &= \max_{s_i, s_j \in S} |t(s_0, s_i) - t(s_0, s_j)| \\ &= \max_{s_i \in S} \{t(s_0, s_i)\} - \min_{s_i \in S} \{t(s_0, s_i)\} \end{aligned}$$

The *cost* of the edge e_v is simply its wirelength, denoted $|e_v|$; this is at least as large as the Manhattan distance between the endpoints of the edge, i.e., $|e_v| \geq d(l(p), l(v))$. The cost of T is the total wirelength of the edges in T . Three directions in the recent clock routing literature are relevant to our present work.

- The *zero-skew tree* (ZST) literature, which seeks a minimum-cost clock tree having zero skew, saw rapid growth during the early 1990's [15, 13]. Notably, the *Deferred-Merge Embedding* (DME) algorithm [1, 3, 8] embeds internal nodes of a topology G via (i) bottom-up construction of a *tree of merging segments*, or *merging tree*, representing loci of possible placements of internal nodes in the ZST; and (ii) top-down

determination of exact locations for the internal nodes of G . DME achieves minimum wirelength and tree radius for any given input topology. Related works study topology constructions that lead to low-cost solutions when DME is applied; the most successful variant is Greedy-DME [9].

- More recently, it has been noted that “exact zero skew” comes at the price of increased wiring area and higher power dissipation, even as circuits still operate correctly within some non-zero skew bound. Hence, the *bounded-skew tree* (BST) problem was addressed in [14, 7, 16, 6]. The BST problem provides a continuous tradeoff between two classic routing problems – the *zero-skew tree* (ZST) problem for skew bound $B = 0$, and the *rectilinear Steiner minimum tree* (RSMT) problem for $B = \infty$.
- Finally, Friedman and coauthors have pointed out that the classic “zero-skew” formulation for clock routing is *over-constrained* – in that there are constraints between all pairs of sinks. Skew constraints actually exist only between pairs of *sequentially adjacent* registers, i.e., pairs of registers connected by purely combinational paths of logic and interconnect. Indeed, “clock skew between nonsequentially connected registers, from an analysis viewpoint, has no effect on the performance and reliability of the synchronous system and is essentially meaningless” [13]. However, no clean and/or optimal techniques (à la DME) have emerged for topology design and route embedding in this less constrained regime.¹

2 The Associative Skew Problem

Despite the observation in [13] and the availability of such works as [17], zero- and bounded-skew formulations dominate current EDA tools, i.e., relative sink delays are constrained for *all* pairs of sink registers. Adoption of useful-skew or skew-scheduling approaches will likely require some time, along with non-trivial methodology changes. The purpose of this paper is to highlight a new clock tree design formulation that offers interesting algorithmic challenges, as well as a more evolutionary path from current tools and methodology.

The main point established by [13] is that in creating the clock topology over synchronizing elements (latches or flip-flops) of a design, the greatest care must be taken for those elements whose datapaths are connected directly together. For example, if flip-flops in a shift register are not clustered together in the same leaf cluster of the (buffer) topology, hold time violations are exceptionally

¹The observation in [13] is not new, e.g., the LP- and graph-based methods of [12] [19] [18] use skew optimizations (“useful skew” [21]) to improve system clock frequency. Kourtev and Friedman [17] use integer programming in a method for simultaneous skew scheduling and topology design. In general, there has not yet been a satisfactory methodology to address the difficult cyclic precedence constraints among (i) component placement, (ii) skew scheduling and retiming optimizations, (iii) clock topology (buffer hierarchy) design, (iv) ECO placement for buffer insertion (and related performance optimizations), and (v) route embedding.

^{*} In memory of Mr. Patrick Catapano, Jr. of Motorola Corporation, and his contributions to modern chip implementation methodology.

[†] Research at UCLA was supported by a grant from Cadence Design Systems, Inc., and by the MARCO/DARPA Gigascale Silicon Research Center. Professor Zelikovsky was supported by a GSU Research Initiation Grant.

likely.²

The term “associative skew” is due to the late Patrick Catapano, Jr., who suggested modifying the buffer clustering phase of existing clock tree synthesis and place-and-route methodologies to ensure that “closely-related” registers be driven by the same leaf buffer, guaranteeing low skew. Here, “closely-related” might mean, e.g., that no more than some small number k of combinational logic levels separates the two registers. Catapano’s suggestion promotes small skew between sequentially adjacent registers, even though it ignores concepts such as non-zero (useful) skew, skew scheduling, etc. In this way, setup and hold violations may be avoided while largely remaining within the current methodology.

Depending on the topology of the graph of sequential adjacency (i.e., over the set of registers), as well as the results of static timing (which determine global skew bounds), the design may turn out to have clusters of registers with few or no skew constraints between clusters, but very tight skew bounds within clusters. The skew bounds within clusters can be addressed, e.g., by existing zero-skew and bounded-skew constructions; the looser global skew constraints between clusters can be addressed with a *post-processing* approach (e.g., by insertion of delay elements). Such a perspective suggests the following problem formulation:

The Associative Skew Tree (AST) Problem. Given a sink set S that is partitioned into subsets S_1, S_2, \dots, S_k , such that the S_i ’s are disjoint and their union is S , construct a minimum-cost clock routing tree over the sinks of S , such that there is zero skew within each sink subset S_i , $i = 1, 2, \dots, k$. (There is no skew constraint between two sinks that are in different subsets.)

Note that the AST problem defines an interesting tradeoff between the traditional zero-skew clock routing problem ($k = 1$ sink group) and the Steiner minimum tree problem ($k = n$ sink groups). This tradeoff is qualitatively different from the bounded-skew tradeoff.

3 Heuristics for the Associative Skew Problem

In this section, we describe and compare several heuristics for the AST problem. The first corresponds to available current practice, while the remaining four are our new approaches.

3.1 Heuristic H0: k -Greedy-DME

An obvious baseline heuristic is to run Greedy-DME [9] over all sinks at once, essentially ignoring the fact that the skews are less constrained than in the classic ZST problem. Below, we will refer to this baseline heuristic as GDME. Nearly as simple, but potentially more effective, is to run Greedy-DME on each of the k sink groups S_1, S_2, \dots, S_k separately, then join the roots of the k ZSTs with a heuristic rectilinear Steiner minimum tree (RSMT), e.g., returned by the Iterated 1-Steiner method [15]. This “ k -Greedy-DME” heuristic, which we call $H0$, can save a logarithmic factor in tree cost versus the previous approach since a ZST can be logarithmically more expensive than the RSMT. Figure 1 shows how the optimal ZST cost over k points (tree roots) evenly spaced on the unit line segment grows as $\Theta(\log k)$ (cf. analysis techniques of [5]), while the optimal RSMT cost remains constant. The use of heuristic $H0$ is motivated when the groups of sinks are well-separated (e.g., have disjoint convex hulls); this would tend to be the case if the physical layout hierarchy reflects the functional hierarchy.

²Since in a shift register there is no logic stage between the output of one register and the input of the next register, it is easy to have a hold violation unless the clock skew is very small. (Hold time violations can also be addressed with useful skew, but as noted above, the concept of useful skew is foreign to current place-and-route tools.)

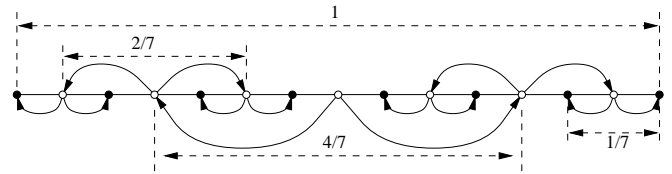


Figure 1: For $k = 2^i$ points (tree roots) positioned evenly on the unit segment, the cost of the rectilinear Steiner minimum tree is 1 while the cost of the ZST is $2^{i-1} \cdot \frac{1}{2^{i-1}} + \dots + 1 \cdot \frac{2^{i-1}}{2^{i-1}} = i \cdot \frac{2^{i-1}}{2^{i-1}} \geq \frac{i}{2} = \Theta(\log k)$.

3.2 Heuristic H1: Optimal Branching

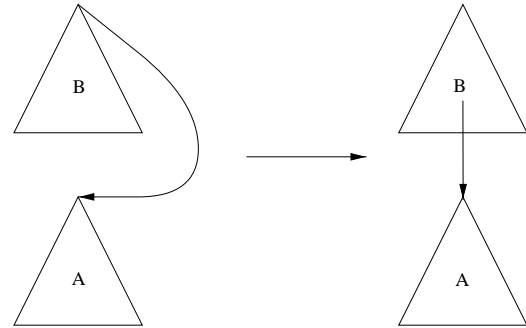


Figure 2: The optimal branching allows a tree root to join into a non-root portion of the target tree; this can result in considerable savings versus simply constructing the RSMT over tree roots.

We next observe that given two ZSTs, tree A over sink subset $S(A)$, and tree B over sink subset $S(B)$, we can combine the trees by *joining* the root of one tree *into* the other tree (not necessarily at the root). The tail of the connecting edge is the root of A , and the head is *inside* the tree B . To achieve this, we need only find the closest point of approach of the root of A to a point of B . Note that this kind of joining can save the root-to-sink cost in the target tree T_B (see Figure 2).

Our heuristic $H1$ starts with k ZSTs $\{Z_1, \dots, Z_k\}$ computed by Greedy-DME over the k sink groups. For each ordered pair of trees (Z_i, Z_j) we find $cost(Z_i, Z_j)$, the cost of the optimal joining of (the root of) Z_j into Z_i . These joining costs may be represented by a complete directed graph G with nodes v_1, \dots, v_k , in which the cost of any edge (v_i, v_j) of G is equal to $cost(Z_i, Z_j)$. Then, finding the optimal joining of all k ZSTs is equivalent to finding the optimal *branching* in G , i.e., the minimum-cost subgraph of G containing paths from a single node to all others. The optimal branching problem was first addressed by Tarjan and by Camerini et al. in the late 1970s [20] [2]; an optimal solution can be found in time $O(k^2)$ using Edmonds’ algorithm as implemented in [11].

3.3 Heuristic H2: Optimal Slice Merging

Even the optimal joining of one tree’s root into another tree can waste a great deal of wiring. We say that the *merging* of tree A into tree B entails connecting each node of a *slice* in A – i.e., a subset of nodes of A that contains exactly one node on each root-to-leaf path (see Figure 5) – into some *joining point* in B . The resulting tree will have paths from the root of B to all sinks of both trees. As shown in

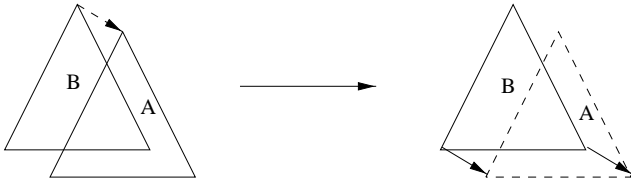


Figure 3: The optimal merging solution is to connect pairs of leaves from the two zero-skew trees. This results in considerable savings versus the optimal branching.

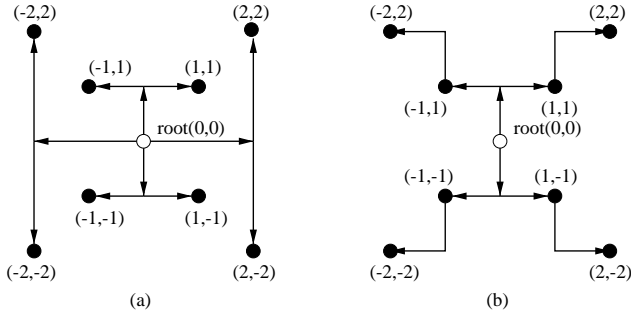


Figure 4: An 8-point example consisting of the corners of two squares centered at the origin. Points in the first group have coordinates $(-1,-1)$, $(-1,1)$, $(1,1)$ and $(1,-1)$. Points in the second group have coordinates $(-2,-2)$, $(-2,2)$, $(2,2)$ and $(2,-2)$. The optimal merging solution (b) with wirelength 14 saves 28% versus the Greedy DME solution with the same offsets (a) with the wirelength 18.

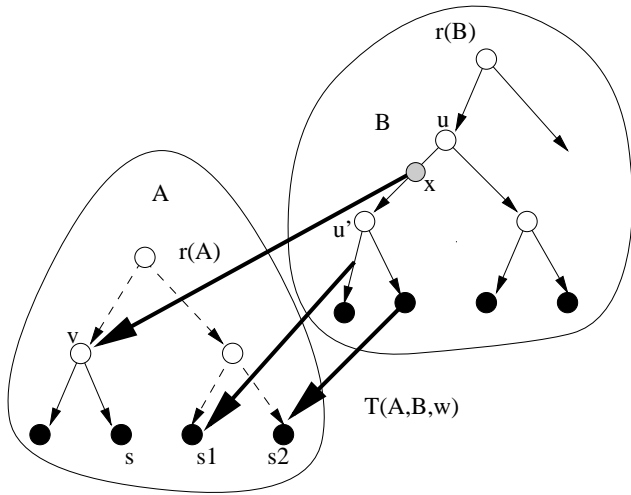


Figure 5: A tree A is joined into a tree B via thick edges. The set of endpoints of these edges forms a slice $\{v, s_1, s_2\}$. The dashed edges of the tree A form the set $\text{above}(\{v, s_1, s_2\})$; they are not part of the merged tree $T(A, B, w)$ and represent an upper bound on wirelength savings. The path from the root of $T(A, B, w)$ to the sink s of the tree A consists of the edge $(r(B), u)$, a part (u, x) of the edge (u, u') , a connecting edge (x, v) and the edge (v, s) of the tree A . The delay along this path is $t(B) + w$.

Optimal Slice Merging Algorithm
Input: two trees A and B , and the prescribed delay offset w
Output: min-wirelength merged AST solution $T(A, B, w)$
(1) For each vertex v in tree A Calculate the adjusted cost $adc(v)$ Traverse edges of tree B in <i>postorder</i> Find $x = x(v)$ on the edge e of B closest to v s.t. delay along the $v - \text{root}(B)$ -path equals $t(B) + w$; Calculate the merging cost $c(v) = v, x(v) $ and $gain(v)$ as $gain(v) = adc(v) - c(v)$;
(2) Traverse the tree A in <i>postorder</i> : if node v is a sink then <i>potential gain</i> $pg(v) = gain(v)$ else $pg(v) = \max\{gain(v), pg(\text{leftchild}) + pg(\text{rightchild})\}$
(3) Traverse the tree A in <i>preorder</i> : if $gain(v) = pg(v)$ for vertex v then put vertex v in <i>Slice</i> else traverse v

Figure 6: The Optimal Slice Merging Algorithm.

Figure 3, the optimal merging solution can save as much as the total cost of tree A when compared with the optimal branching solution. Figure 4 shows that the optimal merging solution can save as much as 28% in tree cost versus the GDME algorithm, even with the same offset between groups.

For the optimal merging to be efficiently applied, we must specify in advance the pairs of trees to be merged, along with a *delay offset* for the two trees being merged – i.e., the difference in source-sink delays (in the merged tree) to sinks of A and sinks of B . Our heuristic $H2$ obtains pairs of trees to be merged, and their order of merging, from the optimal branching solution.³ For each edge (Z_i, Z_j) of the optimal branching of G , heuristic $H2$ applies optimal merging of tree Z_j into tree Z_i . Here, the choice of *delay offset* is critical to success. When merging a tree over sink set A into a tree over sink set B , we compute for each sink in A its distance to the closest sink in B . Let x denote the maximum such distance over all sinks in A . Further, without loss of generality let $t(A) > t(B)$ respectively denote the source-sink delays in trees A and B . Then, the delay offset that we apply in optimal merging is $\max(x, t(A) - t(B) - x)$; this offset is added to the sinks in B .

In the remainder of this subsection, we formally describe the merging of two ZSTs, and the problem of finding an optimal merging. We then show how to optimally merge two ZSTs with prescribed skews.

Given two zero-skew trees A and B with sinks $S(A)$ and $S(B)$, respectively, we wish to optimally merge A into B , i.e., construct the minimum-wirelength tree $T = T(A, B, w)$ such that

- (i) $B \subseteq T$, the root of B is the root of T and the sinks of T is $S(T) = S(A) \cup S(B)$;
- (ii) each path from the root of T to any sink from $S(A)$ consists of edges of B (possibly including a part of a B -edge), a new edge from a node of B to a node of A and edges of A (see Figure 5);
- (iii) Each sink from $S(A)$ has the same delay $t(B) + w$ i.e. the same *delay offset* w with each sink from $S(B)$.

We solve this problem optimally via the following two steps. For each node v in A , we first find the closest point $x = x(v)$ on edges of the tree B , such that the delay along the path

³For $k > 2$, the order of merging is according to a topological ordering of the optimal branching solution. Always, the smaller tree is merged into the larger tree (where size corresponds to the number of sinks in the tree).

$(r(B), \dots, x, v, \dots, s)$ equals $t(B) + w$. Then we find the optimal slice in the tree A , denoted Opt , which minimizes the total length of the tree $T(A, B, w)$. The first step can be done easily by examining all $2|S(B)|$ edges of the tree B for each node v in A . The second step is more involved.

The optimal slice Opt should maximize the *gain* in wirelength, $gain(Opt)$, which is equal to $above(Opt)$ (the total length of edges connecting the elements of Opt to the root $r(A)$ of A , see Figure 5), minus the length of edges connecting each $v \in Opt$ to the corresponding $x(v)$ in B . To enable efficient calculation of gain, we must associate with each node $v \in A$ a value $gain(v)$ such that the gain of any slice will be equal to the sum of gains of its nodes. The gain of v is the difference of two terms: the *adjusted cost* of the v -to-root path in A , minus the cost $|v, x(v)|$ to the corresponding node $x(v)$ in B . In other words, the adjusted cost $adc(v)$ is defined as

$$adc(v) = \sum_{i=0}^{k-1} \frac{|v_i, v_{i+1}|}{2^i} \quad (1)$$

where $v = v_0, v_1, \dots, v_k = r(A)$ is the v -to-root path in A . Note that $|v, x(v)|$ may take on the value $+\infty$ if $x(v)$ does not exist.

Now we will prove the following property of the adjusted cost.

$$\sum_{v \in Slice} adc(v) = above(Slice) \quad (2)$$

We will show that the sum will contain the cost of any edge e above $Slice$. If the head v of e is in $Slice$, then $|e|$ will be counted exactly once in $adc(v)$. By induction assume that $|e_1|$ and $|e_2|$ are counted in the sum once, where e_1 and e_2 are the edges heading to the children of v . The definition (1) implies that exactly the half of $|e|$ is counted in $adc(v)$'s for descendants of the head of e_1 and the other half is counted in $adc(v)$'s for descendants of the head of e_2 . Thus $|e|$ is counted exactly once in the LHS of (2).

We now define $gain(v)$ as $gain(v) = adc(v) - |v, x(v)|$. Then, equation (2) yields that the gain of a slice is equal to the sum of the gains of its elements. After finding the gain of each node of A , we can find the slice with the maximum total gain using the obvious property: the optimal slice in a subtree rooted at v is either $\{v\}$ itself or the union of the optimal slices in two subtrees rooted in the left and the right child of v . We achieve a linear-time implementation by first traversing the tree A in *postorder*⁴, marking the nodes v for which the optimal slice in the subtree rooted at v is $\{v\}$ itself. We then traverse the tree in *preorder*⁵ stopping traversal at the marked nodes.

Note that we can further improve the optimal slice merging by choosing nodes *inside* edges of A as heads of connecting edges similarly to the method how we choose tails inside edges of B .

3.4 Heuristic H3: DME-Merging

As described in [1, 3, 8], the DME algorithm constructs an optimal ZST for a given topology. Our heuristic $H3$ extracts the topology of the optimal slice merging solution ($H2$). We then run the DME algorithm on this topology, using the offsets between sink subsets that were computed by $H2$. The result of $H3$ should be at least as good as that of $H2$, because it gives the optimum embedding of the $H2$ topology. However, as we see in the next section, differences on our testbed between $H2$ and $H3$ are surprisingly small.

4 Computational Experience

We have implemented the four heuristics $H0$, $H1$, $H2$ and $H3$ using C++ in a Unix environment. Portions of our software are derived

⁴I.e., visiting the left child, the right child and then the parent

⁵I.e., visiting the parent, the left child, and then the right child

from the DME implementations of Ken Boese. Our experiments compare the performance (based on total tree wirelength) of these heuristics on synthetic data.⁶ We additionally assess implementations of Greedy-DME (GDME); the GDME1 implementation runs GDME with the sink offsets used by $H2$, while the GDME2 implementation runs GDME with all sink offsets = 0. Runtimes are reported for $H2$ (**Time1**); $H3$ runtimes are essentially the same) and for GDME1 (**Time2**); GDME2 runtimes are essentially the same), measured in CPU seconds on a 300MHz Sun Ultra-10 workstation with 512MB RAM.

The first type of data consists of points in k unit squares ($k = 2, 4$) whose successive origins are displaced from each other by $(shift, shift)$ ($shift = 0, 0.1, 0.25$). In each of these squares we generate n random points ($n = 125, 250$ (and 500 for $k = 2$)); each such group of points simulates the subset of sinks $S_i, i = 1, \dots, k$. The results in Table 1 show that although heuristics $H2$ and $H3$ are better than $H0$ and $H1$, the minimum wirelength is achieved by DME algorithms with greedy topology, i.e., $GDME1$ using offsets from $H2$, and $GDME2$ which is simple GDME without offsets. We conclude that our AST-specific heuristics will not win wirelength if the groups overlap and have similar distances (minimum possible sink delays) to the root.

The second type of data consists of points in sets of concentric squares Q_1, Q_2, \dots, Q_k . The side length of Q_i is equal to either $2i$ or $4i + 2, i = 1, \dots, k$. We generate $2n$ random points in the innermost square, and n random points in each of the "rings" $Q_2 - Q_1, \dots, Q_k - Q_{k-1}$. Again, each such group of points simulates the subset of sinks $S_i, i = 1, \dots, k$. The results in Table 2 show that $H2$ and $H3$ clearly gain over the GDME algorithms that greedily define topologies. Since the runtimes of $H2$ and $H3$ are relatively small in comparison with those of GDME variants, a reasonable metaheuristic may be to simply run both $H2$ and $GDME2$, then return the better result.

n	k	shift	H0	H1	H2	H3	GDME1	GDME 2	Time1	Time2
125	2	0	28.90	28.81	24.05	23.86	25.03	19.8	1.8	2.5
250	2	0	40.13	40.09	34.59	34.35	35.80	28.19	9.9	22.0
500	2	0	56.69	56.65	48.92	48.80	52.48	40.61	69.5	189.8
125	2	0.1	28.91	28.85	24.59	24.57	25.58	20.91	1.8	2.8
250	2	0.1	39.64	39.48	35.88	35.83	37.87	30.28	10.1	22.5
500	2	0.1	56.59	56.39	52.75	52.71	55.96	43.15	71.1	184.3
125	2	0.25	28.92	28.32	26.50	26.49	28.35	23.96	1.8	2.6
250	2	0.25	39.76	39.25	38.15	38.13	40.59	33.05	10.2	20.6
500	2	0.25	57.06	56.61	55.11	55.11	57.47	47.95	71.4	198.1
125	4	0	59.85	59.80	45.26	45.12	47.75	28.03	7.0	24.6
250	4	0	80.57	80.52	62.38	62.27	67.85	40.97	34.7	183.1
500	4	0	113.0	112.98	92.69	92.54	98.37	57.84	213.3	1505.1
125	4	0.1	59.55	59.01	48.99	48.78	56.50	34.36	6.9	23.1
250	4	0.1	79.8	79.23	69.93	69.82	48.86	77.45	33.8	202.4
500	4	0.1	114.58	114.06	103.03	102.89	110.26	69.5	214.1	1576.3
125	4	0.25	59.47	58.83	54.44	54.15	58.83	43.30	6.7	25.4
250	4	0.25	83.18	81.74	78.19	78.07	83.19	60.46	35.9	196.2
500	4	0.25	128.20	127.45	110.13	110.02	116.49	86.46	213.5	1602.5

Table 1: Experimental data for k pointsets generated in shifted square regions. Time1 refers to $H2$ runtime; Time2 refers to $GDME1$ runtime.

5 Conclusions and Ongoing Work

We have introduced the *associative skew tree* (AST) clock routing problem, which seeks a clock tree such that zero skew is preserved

⁶All trees have exact zero pathlength skew within each of the k sink groups. Use of Elmore-DME or higher-order delay models is straightforward, as is accounting for buffer insertion to ensure that no driven capacitive load exceeds a given limit. Our current belief is that such variant testbeds will not qualitatively affect our observations.

n	k	H0	H1	H2	H3	GDME1	GDME 2	Time1	Time2
125	2	48.04	47.63	45.75	45.73	49.26	48.36	2.0	2.8
250	2	84.81	84.72	82.22	81.95	84.88	86.61	2.8	23.6
500	2	123.15	123.12	119.07	119.02	123.28	124.25	11.4	191.2
125	4	156.77	154.31	123.11	123.02	147.67	142.31	6.4	23.9
250	4	204.24	202.19	168.46	168.30	195.34	192.52	42.6	207.1
500	4	270.81	268.23	236.38	236.29	247.61	259.61	214.9	1558.4
125	2	125.63	124.49	121.79	121.75	126.04	125.89	1.9	3.1
250	2	169.46	169.01	165.30	165.24	167.24	169.55	9.8	26.3
500	2	239.87	239.45	235.89	235.88	240.65	240.43	68.1	214.9
125	4	549.23	548.52	489.82	489.74	540.80	548.71	7.1	20.0
250	4	748.75	748.29	686.35	686.27	741.23	749.81	36.0	163.4
500	4	1018.76	1018.19	955.31	955.21	998.08	1019.47	211.1	1300.6

Table 2: Experimental data for k pointsets generated in concentric square and “ring” regions. Time1 refers to H2 runtime; Time2 refers to GDME1 runtime.

only within identified *groups* of sinks. AST gives a new trade-off between the traditional ZST problem (one sink group) and the RSMT problem (n sink groups). We present a set of four heuristics, including two that are based on an efficient *optimal merging* of two zero-skew trees, such that zero skew is preserved within each of the two sink groups. Our computational experience with randomly generated sink sets and the linear delay model suggest that the optimal branching heuristic can outperform Greedy-DME when the sink groups are spatially separated. On the other hand, when the sink groups are intermingled (recall that this is the motivating domain for our optimal-merging algorithm, and the heuristics $H2$ and $H3$), Greedy-DME performs surprisingly well. Since Greedy-DME is oblivious to the relaxation inherent in the AST, the key open issue is to find a heuristic that consistently outperforms Greedy-DME for the domain with intermingled sink groups. Another open issue is how to perform *height-limited* joining of multiple trees (either branchings or slice-mergings) so as to satisfy insertion delay constraints on the clock distribution. Generalizing the approach to include hierarchical buffering, higher-order delay models, and non-zero skew within each given sink group is another direction for future work. Finally, applying the formulation to real design data within a modern flow – where global skew constraints much be enforced, and where methodology often demands a “skew number” returned to the designer – will help point out refinements to the current formulation.

References

- [1] K. D. Boese and A. B. Kahng, “Zero-skew clock routing trees with minimum wirelength,” *Proc. IEEE Intl. Conf. on ASIC*, pp. 1.1.1 – 1.1.5, 1992.
- [2] P. M. Camerini, L. Fratta and F. Maffioli, “A note on finding optimum branchings”, *Networks* 9 (1979), pp. 309-312.
- [3] T.-H. Chao, Y.-C. Hsu, and J.-M. Ho, “Zero skew clock net routing”, *Proc. ACM/IEEE Design Automation Conf.*, pp. 518–523, 1992.
- [4] T.-H. Chao, Y. C. Hsu, J. M. Ho, K. D. Boese, and A. B. Kahng, “Zero skew clock routing with minimum wirelength”, *IEEE Trans. Circuits and Systems*, 39(11):799–814, November 1992.
- [5] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai and A. Tomkins, “Minimizing wirelength in zero and bounded skew clock trees”, *Proc. ACM/SIAM Symp. on Discrete Algorithms*, Baltimore, 1999, pp. 177-184.
- [6] J. Cong, A. B. Kahng, C. K. Koh and C.-W. A. Tsao, “Bounded-Skew Clock and Steiner Routing”, *ACM Trans. on Design Automation of Electronic Systems* 3(3) (1998), pp. 341-388.
- [7] J. Cong and C.-K. Koh, “Minimum-cost bounded-skew clock routing”, *Proc. IEEE Intl. Symp. Circuits and Systems*, volume 1, pp. 215–218, April 1995.
- [8] M. Edahiro, “Minimum skew and minimum path length routing in vlsi layout design. *NEC Research and Development*, 32(4):569–575, 1991.
- [9] M. Edahiro, “A clustering-based optimization algorithm in zero-skew routings”, *Proc. ACM/IEEE Design Automation Conf.*, pp. 612–616, June 1993.

- [10] M. Edahiro, “Delay Minimization for Zero-Skew Routing” *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 563–566, November, 1993.
- [11] M. Fischetti and P. Toth, “An efficient algorithm for the min-sum arborescence problem on complete digraphs”, *ORSA J. Computing* 5(4) (1993), pp. 426-434.
- [12] J. P. Fishburn, “Clock Skew Optimization” *IEEE Trans. Computers*, 39 (7): 945-951, july, 1990.
- [13] E. G. Friedman, editor. *Clock Distribution networks in VLSI Circuits and Systems: A Selected Reprint Volume*. IEEE Press, 1995.
- [14] J. H. Huang, A. B. Kahng, and C.-W. A. Tsao, “On the bounded-skew clock and steiner routing problems”, *Proc. ACM/IEEE Design Automation Conf.*, pp. 508–513, 1995.
- [15] A. B. Kahng and G. Robins. *On Optimal Interconnections for VLSI*, Kluwer Academic Publishers, 1995.
- [16] A. B. Kahng and C.-W. A. Tsao, “More Practical Bounded-Skew Clock Routing”, *Proc. ACM/IEEE Design Automation Conference*, Anaheim, 1997, pp. 594-599.
- [17] I. S. Kourtev and E. G. Friedman, “Topological Synthesis of Clock Trees with Non-Zero Clock Skew”, *Proc. ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Dec. 1997, pp.158-163.
- [18] K. A. Sakallah, T. N. Mudge and O. A. Olukotun, “checkTc and minTc: Timing verification and optimal clocking of synchronous digital circuits”, *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1990, pp. 552-555.
- [19] T. G. Szymanski, “Coputing Optimal Clock Schedules”, *Proc. ACM/IEEE Design Automation Conf.*, 1992, pp. 399-404.
- [20] R. E. Tarjan, “Finding Optimal Branchings”, *Networks* 7 (1977), pp. 25-35.
- [21] J. G. Xi and W. W.-M. Dai, “Useful-Skew Clock Routing with Gate Sizing for Low Power Design”, *Proc. ACM/IEEE Design Automation Conf.*, pp. 383–388, 1996.