An Effective Cost-Skew Tradeoff Heuristic for VLSI Global Routing

[†]Andrew B. Kahng, [‡]Shreyas Thumathy, and [†]Mingyu Woo
[†]UC San Diego, La Jolla, CA, USA {abk,mwoo}@eng.ucsd.edu
[‡]Canyon Crest Academy, San Diego, CA, USA shreyasthumathy@gmail.com

***Abstract*—In global routing of relatively large nets, particularly clock subnets, obtaining a good *cost-skew tradeoff* has remained beyond the reach of efficient heuristics. This is in contrast to obtaining a good cost-radius tradeoff, which has been well-addressed by a series of methods. We propose a simple heuristic for optimizing the cost-skew tradeoff, based on a "multi-source" variation of the well-known Prim-Dijkstra (PD) construction [1]. Our experiments demonstrate the effectiveness of the multi-source PD heuristic compared to existing alternatives including H-tree [5], bounded-skew DME [10], and original Prim-Dijkstra constructions. We additionally develop characterization data for PD with respect to instance parameters to aid future research.**

I. Introduction

Interconnect tree constructions during physical synthesis, timing-driven placement and buffering/sizing are key elements of low-power, high-performance integrated circuit (IC) implementation. Of primary interest for both researchers and practitioners has been the tradeoff between tree *cost* (i.e., the sum of Steiner tree edge lengths, which reflects wirelength (WL)) and tree *radius* (i.e., the longest source-sink pathlength in the tree, which reflects delay and pathlength (PL)). Minimizing cost helps to reduce capacitance (power), coupling and routing congestion. Minimizing radius helps to improve design performance, especially during placement or global routing stages when detailed parasitics and delays are unavailable.

Several tree construction heuristics have focused on the *cost vs. radius* (i.e., WL vs. PL) tradeoff, as reviewed in Section 2 of [2]. Notably, the *Prim-Dijkstra* (PD) heuristic [1] has been used in industrial electronic design automation (EDA) tools for over 25 years. It has seen various improvements over time, including PD-II [2], SALT [8], and TreeNet [18] which applies deep learning to classify pointsets for well-parameterized application of PD-II or SALT.

However, *skew* – the maximum difference in source-to-sink pathlengths or path delays in a routing tree – is also a major consideration for interconnect tree design [17]. Clock subnets, which are determined in a post-placement clock tree synthesis (CTS) step, often have fanouts of 30-50, and skew in these subnets harms both chip performance and design schedule. Also, in trial routing of high-fanout nets before initial buffer insertion and sizing (e.g., during initial global placement), minimizing both skew and cost will improve the routability and timing seen at subsequent flow steps.

The pointset in Figure 1 illustrates why a traditional cost-radius tradeoff heuristic (in this example, Prim-Dijkstra with α parameter = 0.3) will often produce a poor solution in terms of skew. We see that any direct connection from the root to a nearby sink will induce skew at least on the order of the radius (i.e., half the diameter) of the pointset. A low-skew

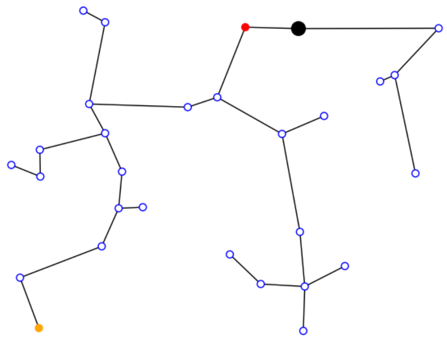


Fig. 1: Prim-Dijkstra (PD) tree with $\alpha = 0.3$. The root is highlighted as the large, solid black circle. The vertices with the minimum and maximum pathlengths (from the root) are respectively highlighted in red and orange.

tree should avoid such edges, but cost-radius tradeoff methods such as Prim-Dijkstra or Steiner arborescences [20] are prone to creating them. Intuitively, to reduce skew in the output of a traditional cost-radius tradeoff heuristic, it will help to initialize the tree with long initial edges that "teleport" the construction far away from the root. This intuition is seen in the H-tree construction [5] and in optimal cost-vs.-skew tree solutions produced by flow integer-linear programs [12] [13].

In this work, we seek an efficient heuristic that produces routing trees with a good cost-skew tradeoff. We furthermore seek to retain the simplicity and efficiency of the state-of-the-art cost-radius tradeoff – namely, the PD construction – while steering it to a good cost-skew tradeoff. Informally, our work explores a *"PD-Clock Problem"*: Given a pointset in the Manhattan plane with an identified source, apply an existing PD implementation to produce a tree with good cost-skew tradeoff.

Our main contribution is a new *multi-source PD* (MSPD) heuristic that produces improved cost-skew tradeoffs while still leveraging an existing efficient PD implementation. MSPD finds a starting configuration for the PD algorithm and the ensuing "Steinerization" step,[1] such that we obtain a Steiner tree with good cost-skew tradeoff. Here, we note that finding high-quality routing trees is understood to be worth a moderate computational expense, especially since poor wirelength

[1]The PD algorithm trades off between Prim's algorithm and Dijkstra's algorithm via the parameter α. ($\alpha = 0$ produces a minimum spanning tree; $\alpha = 1$ produces a shortest-paths tree; and $\alpha = 0.3$ is the usual default in commercial EDA implementations.) As such, PD produces a *spanning tree* over the given pointset. Works in the literature then apply methods such as edge-overlapping (Ho et al. [15]) or detour-aware Steinerization [2] to post-process the PD tree into a "Steinerized" (Steiner) tree.

or skew can lead to expensive downstream design efforts (difficulty in timing closure, or loops back to earlier flow steps). For example, the PD-II method runs at least 11 runs (values of its α parameter) and returns the best result, while SALT similarly uses 20 runs (values of its ϵ parameter). With this in mind, our goal is to identify small sets of candidate starting configurations for PD, such that we can find good cost-skew tradeoffs within reasonable runtimes. (In Section VI, we validate the quality of multi-source PD's cost-skew tradeoff using high-volume ground truth data.)

An additional contribution of our work is the first (to our knowledge) empirical characterization of PD skew vs. cost properties. As described in Section III below, we develop lookup tables to estimate PD skew and cost based on net bounding box aspect ratio, number of terminals, source location relative to the net bounding box center, and the PD algorithm's α parameter. These data highlight values for α that tend to achieve good cost-skew tradeoffs, and that differ from current defaults seen in open-source usage of PD [22].

Outline of the Paper. In the following, Section II establishes notation and terminology, and presents the PD-Clock problem. Section III presents our characterization studies of the cost-skew tradeoff that is inherent in the PD algorithm; this fills a gap that we ascribe to past focus on cost-radius tradeoffs. Section IV presents the multi-source PD construction. Sections V and VI summarize our experimental studies, and we conclude the paper in Section VII.

II. Definitions and Problem Statement

Table I lists notation that is used throughout the paper.

TABLE I: Notation.

Notation	Meaning
V	A signal net, i.e., set of terminals v_i in the Manhattan plane with v_0 being the root (source) and all others being sinks.
G	$G = (V, E)$, the complete (distance-) weighted underlying routing graph induced by V.
T	$T = (V, E')$, a spanning tree in G where $E' \subseteq E$ and $\|E'\| = \|V\| - 1$.
T_P	The minimum spanning tree of G generated by Prim's algorithm (PD with $\alpha = 0$).
T_D	The shortest paths tree of G generated by Dijkstra's algorithm (PD with $\alpha = 1$).
$m_{i,j}$	The Manhattan distance between vertices $v_i, v_j \in V$.
$e_{i,j}$	The edge between $v_i, v_j \in V$ with weight $m_{i,j}$.
$path_{i,j}$	The set of edges on the unique path in T from $v_i \in V$ to $v_j \in V$.
L_T	The wirelength of tree T.
L'_T	The normalized wirelength of T, relative to L_{T_P}.
P_i	The pathlength from v_0 to $v_i \in V$.
S_T	The skew of tree T, i.e., absolute difference between minimum and maximum source-sink pathlengths.
S'_T	The normalized skew of T, relative to S_{T_D}.
ED_i	Elmore delay from the root to sink v_i.
$S_{T_{ED}}$	Elmore delay skew of the tree T.
α	Cost vs. radius control parameter ($0 \leq \alpha \leq 1$).
AR	The aspect ratio (max/min sidelength ratio) of a pointset's bounding box.
p	The *centrality* of the source location of a pointset, expressed as normalized distance from the center of the bounding box.
N	The number of terminals in a pointset, also equal to $\|V\|$.
T_x	A generated PD tree with index x.

In this section, we define key notation and terminology, along with the cost-skew tradeoff of a given routing tree. We then state the PD-Clock problem that this work addresses.

Basic Definitions. A *signal net* $V = \{v_0, v_1, ..., v_k\}$ consists of *terminals* (points) v_i in the Manhattan plane, with v_0 being the *root* (*source*) and all other terminals being *sinks*. We use N to refer to $|V|$, and "net" and "pointset" are used interchangeably. The underlying *routing graph* $G = (V, E)$ is the complete edge-weighted graph induced by V, where the edge $e_{i,j} \in E$ between $v_i, v_j \in V$ has *weight* equal to the Manhattan distance $m_{i,j}$ between the two terminals.

A *routing tree* $T = (V, E')$ is a connected, edge-weighted subgraph of G that spans V and has $|E'| = |V| - 1$. We use L_T to denote the total *wirelength* of T, i.e., the sum of the weights of all edges $e_{i,j} \in E'$ in T. We define P_i, the *pathlength* from the root to $v_i \in V$, as the sum of edge weights on the unique path from v_0 to v_i in T. The *skew* of T, denoted S_T, is the difference between the maximum and minimum root-to-sink pathlengths in T.

The PD algorithm greedily builds a spanning tree T, iteratively adding terminal v_i and edge $e_{i,j}$ into the growing tree that minimizes $(\alpha * P_j) + m_{i,j}$ over all $v_i \in V - T$, $v_j \in T$. Section III studies trees T that are produced by running PD with pointset V and $0 \leq \alpha \leq 1$ as inputs.

Cost-Skew Tradeoff. We will capture the *cost-skew tradeoff* of a given tree as the sum of a *normalized wirelength* of T, denoted L'_T, and a *normalized skew* of T, denoted S'_T. In this paper, we study the cost-skew tradeoff defined as $L'_T + S'_T$; our methods can apply to other combinations of L'_T and S'_T.

Let T_P be the minimum spanning tree (MST) of G, which is produced by running PD with $\alpha = 0$ on V (i.e., equivalent to Prim's MST algorithm). Because T_P is the tree with the smallest possible WL cost, we define the normalized wirelength of T as

$$L'_T = \frac{L_T}{L_{T_P}} \tag{1}$$

Let T_D be the shortest paths tree (SPT) of G, which is produced by running PD with $\alpha = 1$ on V (i.e., equivalent to Dijkstra's SPT algorithm). We define the normalized skew S'_T of a given tree T based on the concept of *shallowness* or *radius* of a rooted tree [17], i.e., the ratio between the maximum pathlength in T and the maximum pathlength in T_D. Because skew and pathlength are closely related in the context of the PD-Clock problem that we study, and because finding the minimum-skew tree is *NP-hard* [7], we define S'_T to be the ratio between S_T and S_{T_D},

$$S'_T = \frac{S_T}{S_{T_D}} \tag{2}$$

The PD-Clock Problem. Given a pointset V, the PD-Clock problem aims to find a small subset of terminals (in this work, between one and three terminals) to *initially* connect to the root v_0, such that the PD construction can be *continued* to minimize the cost-skew tradeoff $L'_T + S'_T$ while maintaining a low computational expense.

III. Cost-Skew Characterization of Prim-Dijkstra

In this section, we characterize the cost-skew tradeoff inherent in the PD heuristic. We use the improved PD-II version, as open-sourced [14] by the authors of [2]. Our characterization effort has produced lookup tables that can be used to quickly estimate the cost and skew of PD when run on a given pointset.[2] Such a table can be used, e.g., to guide hierarchical

[2]Finding a net bounding box and its aspect ratio requires time linear in N. The table lookup is constant-time.

clustering and buffer placement in CTS. Source codes for PD and for the MSPD heuristic that we describe below, along with lookup tables for this section's results, are available in the Multi-Source-Prim-Dijkstra GitHub repository [3].

A. Design of Experiments

Our characterization sweeps the pointset size, the pointset bounding box aspect ratio, the position of the source relative to the bounding box center, and the value of α. As mentioned earlier, relevant fanouts (N) in the CTS context can be as large as $\sim$50. Works such as [9] [6] [16] have studied the dependence of tree construction outcomes on both N and the aspect ratio (AR, i.e., height divided by width) of the bounding box of the given pointset. In this work, since we are concerned with skew, another relevant parameter is the *centrality* of the root location relative to the pointset bounding box.

Generating a pointset with prescribed bounding box aspect ratio. To generate a random pointset with a prescribed bounding box aspect ratio, we use code provided by authors of [16]. Note that the bounding box of a given pointset (when the points are in general position, with no two co-linear) is determined by two, three, or four points. The pointsets we use follow the probability distributions detailed in Section 2.2 of [16], namely,

$$P(N,2) = \binom{N}{2}\left(\frac{2}{N}\right)^2\left(\frac{1}{N-1}\right)^2 \tag{3}$$

is the probability that the bounding box is determined by 2 terminals;

$$P(N,4) = \binom{N}{4}\left(\frac{4!}{N^2(N-1)^2}\right) \tag{4}$$

is the probability that it is determined by 4 terminals; and

$$P(N,3) = 1 - P(N,4) - P(N,2) \tag{5}$$

is the probability that it is determined by 3 terminals.

Centrality of the root (source). We measure the *centrality* of a point with respect to the pointset bounding box using a parameter $0 \leq p \leq 1$, where $p = 0$ means that the point is at the bounding box center, and $p = 1$ means that the point is on the edge of the bounding box.

If the pointset bounding box has width w, height h, and center point (a, b), then a point has *centrality* p if it lies on the boundary of the rectangle defined by the lines $x = a \pm pw/2$ and $y = b \pm ph/2$.

For a given pointset, we say that the point with centrality p is the closest point, by Manhattan distance, to the boundary of the rectangle defined by the lines $x = a \pm pw/2$ and $y = b \pm ph/2$.

Pointset generation. For each combination of pointset size $N = \{20, 30, 40, 50\}$, bounding box aspect ratio $AR = \{1, 1.2, 1.4, 1.6, 1.8, 2, 2.5, 3, 4, 5\}$, and centrality of the root $p = \{0, 0.2, 0.4, ..., 1\}$, 2500 pointsets are randomly generated, yielding a total of 600,000 distinct pointsets. (Generated pointsets are scaled such that a pointset with prescribed bounding box AR has bounding box [0...10,000] × [0...AR * 10,000].) Then, each of the 600,000 pointsets (i.e., nets) is input to PD and PD-II with various α values, $\alpha = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.42, 0.44, ..., 0.7, 0.8, 0.9, 1.0\}$. Last, for each generated PD-II output tree, the WL and skew are respectively normalized to L_{T_P} and S_{T_D} of that net.

B. Characterization Results

We now describe empirical relationships observed in PD-II trees between the cost-skew tradeoff and the AR, N and p parameters.

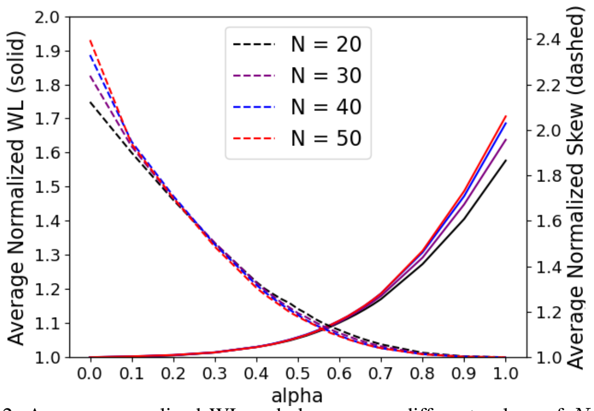


Fig. 2: Average normalized WL and skew across different values of N, for fixed $p = 0$ and $AR = 1$.

Figure 2 shows traces of average normalized WL (solid lines) and skew (dashed lines) for $N = \{20, 30, 40, 50\}$, with fixed $AR = 1$ and $p = 0$, over the range of possible α values. We observe a small increase of both L' and S' as N increases. The average normalized skew increase with larger N is expected, since generally the largest PL increases and/or the minimum PL decreases with larger pointsets. The "sweet spot" at $\alpha \approx 0.55$ is noteworthy: (i) it is different from the folklore $\alpha = 0.3$ that is commonly used with the PD heuristic, and is currently seen in the clock subnet tree generation of OpenROAD [22]; and (ii) it shows surprising stability across all our data.

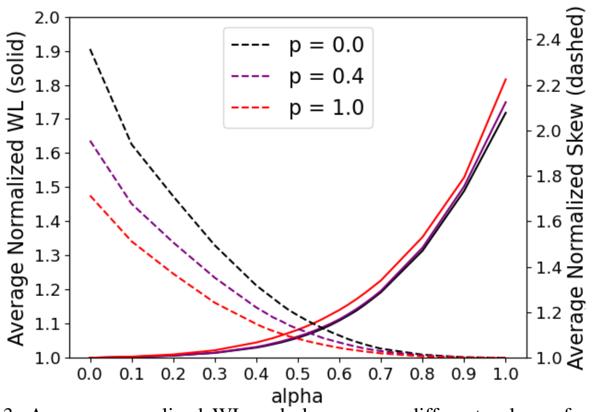


Fig. 3: Average normalized WL and skew across different values of p, for fixed $AR = 1.4$ and $N = 50$.

Figure 3 shows how centrality p of the source location affects the cost-skew tradeoff, with fixed $AR = 1.4$ and $N = 50$. As p increases, average normalized WL L' (solid lines) also increases, but average normalized skew S' (dashed lines) decreases significantly. With large p, the source is near the edge of the pointset bounding box, and in some sense not much can be done to improve spanning tree skew over the Dijkstra ($\alpha = 1$) outcome. On the other hand, MST cost

($\alpha = 0$) of a pointset remains the same regardless of source location centrality (p). We believe this explains the average drop-off of S' and the small increase of L' seen for $p = 1.0$ as opposed to $p = 0.0$.

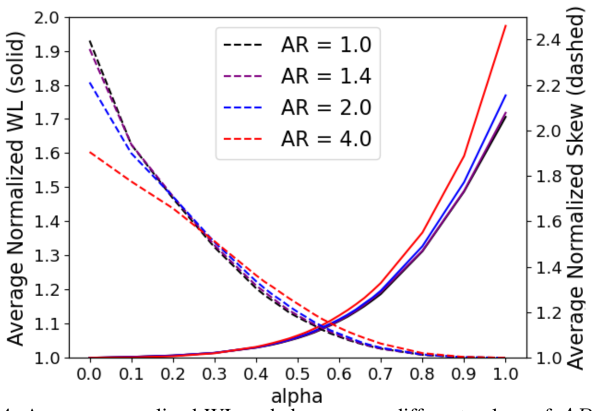


Fig. 4: Average normalized WL and skew across different values of AR, for fixed $p = 0$ and $N = 50$.

Finally, Figure 4 shows outcomes for various pointset bounding box aspect ratios AR, for fixed $N = 50$ and $p = 0$. As AR increases, the average normalized L' (solid lines) increases, which we see as consistent with previous studies of RSMT cost (see Figure 11 of [6]). The average normalized S' (dashed lines) tends to decrease with increasing AR. There is also less variation in S' over the entire range of α with smaller values of AR.[3]

C. Lookup Tables

In the *LUTs* folder of the GitHub repository [3], we provide 4-dimensional lookup tables which give mean and variance of L'_T and S'_T for combinations of N, AR, p, and α parameters. To achieve this, we randomly generate 2500 nets for each combination of $N = \{20, 30, 40, 50\} \times AR = \{1, 1.2, ..., 2, 2.5, 3, 4, 5\} \times p = \{0, 0.2, ..., 0.8, 1\}$. For each net, we run PD-II with the 23 values of $\alpha = \{0, 0.1, 0.2, ..., 0.4, 0.42, 0.44, ..., 0.68, 0.7, 0.8, 0.9, 1\}$, i.e., for each ($N$, AR, p) triple we collect a total of 2500 * 23 = 57,500 PD-II trees. We then populate our lookup table with the average and variance of the normalized L'_T and S'_T values for each combination of N, AR, p, and α.

IV. The Multi-Source PD Heuristic

We now describe *multi-source Prim-Dijkstra*, which heuristically addresses the goal of efficiently finding good cost-skew tradeoff solutions based on an existing PD implementation. We are unable to formally quantify or bound the suboptimality of our heuristic solutions (the problem is difficult [13] [12]), but experiments in Section V below assess heuristic solutions against a large set of ground-truth data.

Recall the above intuition that skew can be reduced by "teleporting" the tree construction far away from the root. We have studied the benefit of introducing a small number of root-to-terminal edges into the start of the PD execution. Intuitively, this can decrease skew by increasing the shortest root-to-terminal pathlength, and/or decreasing the longest root-to-terminal pathlength. However, we must be mindful that the PD algorithm adds an additional edge incident to the root if it minimizes the expression $(\alpha * P_j) + m_{i,j}$, and PD-II incrementally changes the tree topology if it can improve $\alpha * \sum_{v_i \in V} Q_i + (1 - \alpha) * L_T$, where Q_i represents the detour cost $P_i - m_{0,i}$ for a vertex [2]. Thus, unless explicitly prevented, short root-terminal connections beyond the initial "seeded" connections could be added that spoil the tree skew.

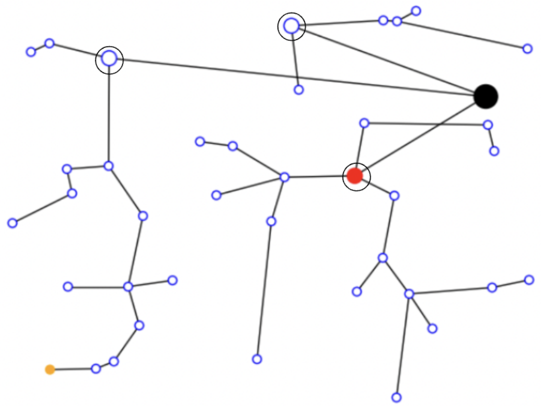


Fig. 5: An example tree produced by MSPD for a pointset with $N = 40$. The root is shown in solid black. The three points used as sources are indicated by enlarged double-circles. In the resulting spanning tree, the sinks with minimum and maximum pathlength distances from the root are respectively highlighted in red and orange.

Our *multi-source PD* heuristic forces PD to not allow any incident edges to the root, other than edges inserted at the initialization of the algorithm. In the multi-source approach, the chosen set of sources are included in the initial heap. This simple change allows PD to construct disjoint trees from these sources, with these trees connected to each other via the initialized edges from the root. Intuitively, this preserves skew benefits of "teleporting" away from the root, while preventing short edges incident to the root. Figure 5 illustrates the output of the multi-source variant.

Figure 6 shows cost-skew Pareto frontiers of original PD and multi-source PD for four pointsets with $N = 40$. The PD traces are from 11 PD-II runs with $\alpha = \{0, 0.1, ..., 1\}$. The multi-source traces are derived from runs of PD-II with all combinations of 1, 2 and 3 terminals as seeds (a total of 9920 combinations), crossed with 11 α values, as detailed in Section V below. While relative performance is instance-dependent, the original PD is clearly dominated by the multi-source variant.

Algorithm 1 formally presents the multi-source PD algorithm, which functions in nearly the same way as the PD algorithm described in [1], [2].

- Lines 1-8 initialize an empty tree and a queue with the sources, then set the keys of the root and sources to be zero. Any terminal that is not the root or a source has its key and PL set to the Manhattan distance between that terminal and the root.
- Line 9 sets the pathlength and key of the root to be zero, but does not add the root to Q so as to prevent additional incident edges to the root.

[3]Note that it is not sensible to compare the normalized *skew* values observed over various parameters of p, AR. This is because unlike normalized cost, the normalized skew is highly dependent on root location.

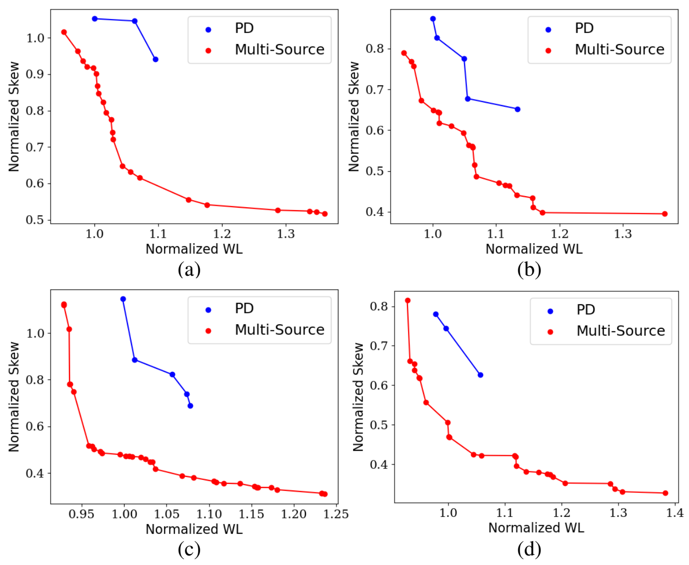


Fig. 6: Cost-skew Pareto frontiers of PD (blue) and multi-source (red) heuristics. (a)∼(d) are from four pointsets with $N = 45$.

ALGORITHM 1: Multi-source PD

```
Input: Underlying Routing Graph $G = (V, E)$, Source Set $S$, $\alpha$
Output: Spanning tree $T_{out} = (V, E_{out})$ with $E_{out} \subseteq E$ and
$e_{0,source} \in E_{out}$ s.t. $source \in S$
 1: Initialize $Q, T_{out}, \leftarrow \emptyset$
 2: for all $v \in V$ do
 3:     $PL_v, key_v \leftarrow m_{0,v}$
 4: end for
 5: for all $s \in S$ do
 6:     $keys_s \leftarrow 0$
 7:     add $s$ to $Q$
 8: end for
 9: $PL_{root}, key_{root} \leftarrow 0$
10: while size $Q > 0$ do
11:     $v \leftarrow$ vertex in $Q$ with min $key_v$
12:     Remove $v$ from $Q$
13:     for $j = 0...|V|$ do
14:         $u \leftarrow j^{th}$ closest neighbor to $v$
15:         $d \leftarrow m_{u,v} + PL_v * \alpha$
16:         if $key_u \geq d$ then
17:             $par_u \leftarrow v$, $key_u \leftarrow d$
18:             $PL_u \leftarrow m_{u,v} + PL_v$
19:             Add $u$ to $Q$ if not already added
20:         end if
21:     end for
22: end while
23: for $v \in V$ do
24:     Insert $e_{par_v,v}$ into $T_{out}$
25: end for
26: Steinerize $T_{out}$ without removing $sources \in S$
    or adding incident edges to the root.
27: return $T_{out}$
```

- Lines 10 and 22 delimit a **while** loop which iterates through each element in the queue, removing the vertex with the minimum key (call this vertex v).
- Lines 11-12 remove the vertex $v \in Q$ with the minimum key.
- Lines 13-21 change u's parent to v if the change decreases u's key. Then, u's key and pathlength are both updated, and Q is modified to contain u.
- Lines 23-27 add terminals and edges into the growing tree, Steinerize the tree, and return the final tree.

Like the original PD heuristic, Algorithm 1 follows the template of Dijkstra's algorithm and has the same asymptotic runtime of $\mathcal{O}(N^2)$.

V. Empirical Studies of Multi-Source PD

We now present empirical studies of MSPD, where up to three sources are used. We first describe data generation, followed by two metrics of suboptimality and two intuitions obtained from study of multi-source trees with good cost-skew properties. We then describe a Multi-Source Selection (MSS) heuristic that incorporates these intuitions.

Pointset and Ground-Truth Data Generation. We generated 500 pointsets of size 10, 15, 25, and 30, 100 pointsets of size 40, and 5 pointsets of size 45 and 50. All of these pointsets were generated uniformly at random, with all points having integer coordinates within a [0...1000] $\times$ [0...1000] bounding square.

Each pointset of size N is input to the modified PD algorithm. Let $R = 1 + \binom{N-1}{1} + \binom{N-1}{2} + \binom{N-1}{3}$. R is the total number of 0-, 1-, 2- and 3-source combinations that can be passed into our PD algorithm. (When $N = 30$ (resp. 40, 50), $R = 4090$ (resp. 9920, 19650).) For all R combinations, 11 values of $\alpha = \{0.0, 0.1, ..., 1.0\}$ are used to test the effect of α on our modified PD. Thus, a total of $R' = R * 11$ combinations of source choices and α values are fed to the multi-source PD algorithm.

$J\%$ (Rank) and $K\%$ (Suboptimality) Metrics. We introduce two metrics to determine how the solution quality of tree T, i.e., $L'_T + S'_T$, compares to that of *all* (i.e., $R' - 1$) other trees that can be generated using the multi-source approach. Please note that while these metrics (and, even, our choice of normalization for tree skew) may at first seem "arbitrary", we believe that the conclusions we draw are robust and accurately reflect what we find with other reasonable alternatives.

Our $J\%$ metric reflects the *ranking* of T. Consider all R' trees that are generated by the multi-source PD variant. We say that a given tree T has $J\%$ suboptimality with respect to ranking if, after sorting all R' trees in non-decreasing order of their solution quality, T is within the first $J\%$ of all trees.

Our $K\%$ metric reflects the *suboptimality* of T. For a given net, let T_b be the tree with minimum $L'_T + S'_T$ among all R' trees that we evaluate. Then, a given tree T_a has $K\%$ suboptimality with respect to solution quality, where $K = 100 * (L'_{T_a} + S'_{T_a} - (L'_{T_b} + S'_{T_b}))/(L'_{T_b} + S'_{T_b})$.

Intuition: Center-Most Point. From studies of high-quality multi-source trees, we observe that there is often a source that is "central" to the set of terminals. This tends to be true regardless of the root location. Intuitively, choosing a central point as a source is reasonable, since this would tend to prevent any terminal from having too long a PL from the root. And, reducing maximum PL by choosing a central point as a source would in turn reduce skew. This intuition is borne out in our experimental studies of the Multi-Source Selection heuristic (Algorithm 2) below, which adds a *center-most point* to its set of candidate sources.[4] See Table IV in Section VI.

Intuition: Reaching Clustered Points. Multi-source PD trees with good cost-skew balance also tend to use sources that are well placed to reach other points with minimal wire cost. Intuitively, this means that if there is a relatively dense cluster

[4]For a given pointset, we say that the *center-most point* is the point that is closest, by Manhattan distance, to the center of the pointset's bounding box.

of points, one of these points is promising as a source; having a direct edge from the root will limit the PL to other points in the cluster, which helps to control skew. Based on this intuition, our Multi-Source Selection heuristic (Algorithm 2) preferentially uses sources that have small sum of Manhattan distances to their closest neighbors.

Multi-Source Selection Heuristic. Given the above intuitions, we have implemented the following heuristic (Algorithm 2) for multi-source selection in a given pointset. The MSS heuristic has two input parameters κ and λ.

ALGORITHM 2: Multi-Source Selection

```
   Input: $V$, $\kappa$, $\lambda$
   Output: Spanning tree $T_{out} = (V, E_{out})$ with $E_{out} \subseteq E$
 1: $S \leftarrow V$
 2: for $v \in V$ do
 3:     $kw_v \leftarrow$ sum of Manhattan distances to the
        $\kappa$ closest neighbors of $v$
 4: end for
 5: Sort vertices in $S$ by $kw_v$
 6: while $|S| > \lambda$ do
 7:     $v_1, v_2 \leftarrow$ vertices in $|S|$ that are closest to each other
        by Manhattan distance
 8:     Remove $v_{1 \leq i \leq 2}$ such that $kw_{v_i} \geq kw_{v_{j \neq i}}$
 9: end while
10: Add the center-most terminal to $|S|$
11: Initialize $T_{best} \leftarrow \emptyset$
12: for all 0-, 1-, 2- and 3- combinations of terminals $C$ do
13:     for $\alpha = 0, 0.1, ..., 0.9, 1.0$ do
14:         $T$ = Run multi-source PD with $V$, $C$, and $\alpha$
15:         if $T$ has better solution quality than $T_{best}$ then
16:             $T_{best} \leftarrow T$
17:         end if
18:     end for
19: end for
```

- Lines 1-4 populate a set of vertices, S, and sort S by kw_v, which is the sum of Manhattan distances to the κ closest neighbors, of each vertex v.
- Lines 5-8 remove vertices in S until $|S|$ has size λ, by (i) finding the closest pair of vertices in $|S|$, then (ii) removing the vertex with lower kw_i value in that pair.
- Line 9 adds the center-most point with respect to the bounding box, based on Manhattan distance, to $|S|$.
- Lines 10-18 construct and return the best tree by generating combinations of 1 to 3 vertices in S to be the multi-sources, and crossing these combinations with α values in the range [0, 1].

The parameter λ allows us to control computational expense, by upper-bounding the number of calls to multi-source PD by 11 * $\mathcal{O}(\lambda^3)$. Larger values of λ will generally improve the quality of the best MSPD tree found. The heuristic takes $\mathcal{O}(N^2 \log N + \lambda^3)$ time to create a candidate set of sources and construct combinations of sources and α values. We achieve this runtime by pre-computing closest pairs of vertices, and traversing this array of pairs until we come across a pair that contains two vertices in S ($\mathcal{O}(N^2 \log N)$ runtime). We also pre-compute the $N-1$ nearest neighbors to each vertex ($\mathcal{O}(N^2 \log N)$ runtime). When taking into account the number of times the MSPD algorithm is called, the total runtime of our heuristic is $\mathcal{O}(N^2(\log N + \lambda^3))$.

VI. Experimental Results and Discussion

We now present experimental studies of the Multi-Source Selection heuristic (Algorithm 2). Our main experiment runs Multi-Source Selection with $|V| = \{10, 15, 25, 30, 35, 40, 45, 50\} \times \kappa = \{1, 2, 3, 4\} \times \lambda = \{1, 2, 3, 4\}$. For each tree T returned by Multi-Source Selection, we compare the $L'_T + S'_T$ value, using both the $K\%$ suboptimality and $J\%$ rank metrics, to the best result obtained from running MSPD with *all* possible 1-, 2-, and 3-source combinations $\times$ $\alpha = \{0, 0.1, ..., 0.9, 1\}$. We do this using (i) PDRev [24], which is comprised of (PD+HVW+DAS), and (ii) STT, which is comprised of (stt+DAS), where stt is the implementation of [4]. We call DAS on the stt output because it significantly improves wirelength with a minimal computational expense.

Study 1: $K\%$ and $J\%$ Suboptimalities. Tables II (for PDRev) and III (for STT) each show the $K\%$ solution quality suboptimality metric and the $J\%$ rank suboptimality metric on the left and right, respectively. For PDRev runs, we randomly generate nets according to the following experimental design:[5]

- $N = \{10, 15, 25, 30\} \times T = 1000$
- $N = \{40, 45, 50\} \times T = 500$

For STT runs, we randomly generate nets according to the following experimental design:

- $N = \{10, 15, 25, 30, 40, 45, 50\} \times T = 5000$

TABLE II: $K\%$ (left) and $J\%$ (right) suboptimalities for best heuristic-generated trees compared to the best of all R' MSPD PDRev trees.

$\|V\|$	κ	λ			
		1	2	3	4
10	1	5.83%	4.71%	3.63%	2.77%
	2	5.69%	4.06%	2.95%	2.08%
	4	5.72%	3.86%	2.81%	1.98%
	8	5.06%	3.81%	2.77%	1.95%
15	1	6.16%	5.12%	4.24%	3.90%
	2	6.19%	5.09%	4.02%	3.68%
	4	6.09%	5.03%	4.05%	3.13%
	8	6.07%	4.85%	3.94%	3.07%
25	1	5.78%	5.45%	4.68%	4.54%
	2	5.69%	5.40%	4.70%	4.46%
	4	6.23%	4.83%	4.57%	3.85%
	8	5.70%	4.78%	4.00%	3.82%
30	1	6.14%	5.41%	5.19%	4.43%
	2	6.21%	5.40%	5.04%	4.29%
	4	6.09%	5.23%	4.97%	4.20%
	8	6.09%	5.23%	4.49%	4.19%
40	1	6.00%	5.52%	5.06%	4.64%
	2	6.06%	5.50%	5.13%	4.76%
	4	5.91%	5.32%	4.87%	4.47%
	8	5.86%	5.28%	4.74%	4.25%
45	1	5.83%	5.36%	4.96%	4.60%
	2	5.83%	5.39%	4.99%	4.55%
	4	5.70%	5.29%	4.94%	4.38%
	8	5.77%	5.27%	4.82%	4.37%
50	1	6.51%	5.93%	5.55%	5.20%
	2	6.40%	6.00%	5.53%	5.14%
	4	6.42%	5.84%	5.43%	4.97%
	8	6.42%	5.86%	5.39%	4.95%

$\|V\|$	κ	λ			
		1	2	3	4
10	1	3.45%	2.44%	1.71%	1.28%
	2	3.22%	2.18%	1.43%	1.03%
	4	3.32%	1.97%	1.36%	0.95%
	8	3.18%	1.97%	1.37%	1.01%
15	1	2.45%	1.77%	1.35%	1.00%
	2	2.37%	1.69%	1.13%	0.87%
	4	2.27%	1.52%	1.08%	0.77%
	8	2.23%	1.43%	1.07%	0.78%
25	1	1.25%	0.90%	0.71%	0.58%
	2	1.21%	0.88%	0.69%	0.54%
	4	1.23%	0.78%	0.59%	0.44%
	8	1.17%	0.73%	0.52%	0.38%
30	1	1.09%	0.80%	0.61%	0.46%
	2	1.12%	0.79%	0.55%	0.42%
	4	1.01%	0.71%	0.51%	0.37%
	8	1.05%	0.69%	0.49%	0.37%
40	1	0.83%	0.57%	0.42%	0.32%
	2	0.85%	0.52%	0.44%	0.33%
	4	0.72%	0.50%	0.40%	0.30%
	8	0.71%	0.52%	0.38%	0.24%
45	1	0.73%	0.51%	0.39%	0.30%
	2	0.65%	0.52%	0.39%	0.27%
	4	0.60%	0.48%	0.37%	0.23%
	8	0.63%	0.43%	0.33%	0.24%
50	1	0.73%	0.48%	0.37%	0.31%
	2	0.65%	0.50%	0.37%	0.30%
	4	0.66%	0.45%	0.34%	0.28%
	8	0.68%	0.46%	0.34%	0.26%

As expected, for fixed $|V|$ and κ, both $K\%$ and $J\%$ values decrease as λ increases. Also, in our experience, increasing κ

[5]The PDRev implementation described in [2] uses HVW to initially Steinerize the output of PD. The implementation in [24] uses the subroutine generate_permutations, which has runtime exponential in the tree depth, to find edge overlaps. Consequently, we decrease T for nets with 40 or more terminals. The runtimes in Table V are due to larger tree depths seen with larger α values.

TABLE III: $K\%$ (left) and $J\%$ (right) suboptimalities for best heuristic-generated trees compared to best of all R' MSPD STT trees.

$\lvert V\rvert$	κ	λ			
		1	2	3	4
10	1	6.11%	4.87%	3.82%	2.99%
	2	6.07%	4.60%	3.40%	2.58%
	4	5.97%	4.22%	3.09%	2.25%
	8	5.71%	4.10%	3.05%	2.26%
25	1	6.93%	5.97%	5.04%	4.28%
	2	6.86%	5.73%	4.78%	3.97%
	4	6.82%	5.48%	4.43%	3.62%
	8	6.65%	5.19%	4.11%	3.30%
25	1	7.03%	6.34%	5.71%	5.14%
	2	6.98%	6.25%	5.56%	5.00%
	4	6.95%	6.15%	5.35%	4.78%
	8	6.86%	5.96%	5.08%	4.46%
30	1	6.95%	6.37%	5.86%	5.40%
	2	6.92%	6.30%	5.71%	5.22%
	4	6.89%	6.19%	5.58%	5.04%
	8	6.85%	6.08%	5.33%	4.79%
40	1	6.75%	6.32%	5.88%	5.49%
	2	6.72%	6.26%	5.80%	5.38%
	4	6.70%	6.18%	5.66%	5.21%
	8	6.63%	6.04%	5.44%	4.97%
45	1	6.62%	6.25%	5.85%	5.47%
	2	6.61%	6.22%	5.76%	5.40%
	4	6.57%	6.12%	5.68%	5.25%
	8	6.57%	6.05%	5.53%	5.09%
50	1	6.64%	6.26%	5.88%	5.55%
	2	6.63%	6.25%	5.84%	5.44%
	4	6.60%	6.15%	5.69%	5.29%
	8	6.57%	6.07%	5.55%	5.13%

$\lvert V\rvert$	κ	λ			
		1	2	3	4
10	1	2.83%	2.02%	1.47%	1.08%
	2	2.78%	1.85%	1.26%	0.90%
	4	2.76%	1.68%	1.14%	0.80%
	8	2.67%	1.75%	1.22%	0.86%
15	1	2.02%	1.51%	1.12%	0.83%
	2	1.97%	1.36%	0.97%	0.70%
	4	1.97%	1.28%	0.88%	0.62%
	8	1.91%	1.20%	0.81%	0.57%
25	1	1.25%	0.97%	0.75%	0.57%
	2	1.23%	0.93%	0.68%	0.52%
	4	1.21%	0.89%	0.63%	0.47%
	8	1.19%	0.85%	0.58%	0.43%
30	1	1.07%	0.84%	0.66%	0.54%
	2	1.06%	0.82%	0.62%	0.48%
	4	1.04%	0.77%	0.58%	0.44%
	8	1.04%	0.74%	0.53%	0.39%
40	1	0.84%	0.69%	0.55%	0.44%
	2	0.82%	0.66%	0.52%	0.41%
	4	0.81%	0.63%	0.47%	0.36%
	8	0.80%	0.59%	0.43%	0.33%
45	1	0.73%	0.61%	0.48%	0.38%
	2	0.72%	0.58%	0.45%	0.36%
	4	0.72%	0.56%	0.43%	0.33%
	8	0.71%	0.53%	0.39%	0.29%
50	1	0.69%	0.57%	0.47%	0.39%
	2	0.68%	0.56%	0.44%	0.35%
	4	0.67%	0.53%	0.41%	0.32%
	8	0.66%	0.51%	0.38%	0.29%

for fixed $|V|$ and λ will generally increase solution quality. For fixed κ and λ, as $|V|$ increases so does the $K\%$ suboptimality. However, the opposite is seen for the $J\%$ rank metric: as $|V|$ increases for a fixed λ and κ, our heuristic tree rank tends to improve (i.e., the percentages shown in Tables II and III decrease). This is intuitively reasonable, as the number of multi-source trees we consider grows as $|V|^3$. If our heuristic generates very high-quality trees, their numerical rank might be similar across various values of $|V|$, but the $J\%$ metric would decrease since the denominator (i.e., number of trees that can be generated) grows rapidly with $|V|$.

Study 2: Benefit from the Center-Most Point. We have also studied the benefit derived from including the the center-most point as a candidate source (Line 9 of Algorithm 2). Table IV analyzes data from our main experiment that corresponds to $\kappa = 2$ and $\lambda = 4$. A given entry in a table is computed as a ratio: the denominator is the number of multi-source trees generated by the heuristic that are within $K\%$ of the best possible solution quality, and the numerator is the number of these trees that use the center-most point as a source. In other words, the table shows the frequency with which a heuristic-generated tree within $K\% = \{1, 2, ..., 5\}$ suboptimality of the best possible result has the center-most point as a source. We see that high-quality generated trees are quite likely to use the center-most point as a source. Further, as $K\%$ increases, the frequency generally decreases. This is anticipated since as suboptimality threshold $K\%$ increases, so does the number of generated trees that will fall within that threshold.

Additional Data: Runtime and Elmore Delay Skew. Table V details the runtime (in seconds, on a 1.70GHz Xeon E5-2650L v4 server) taken by our MSS heuristic for multiple $|V|$ with a fixed $\lambda = 2$ and $\kappa = 4$. We use the same OpenMP

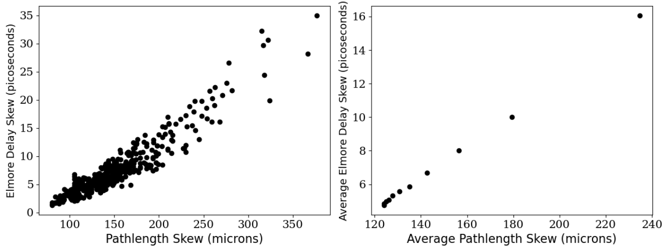
TABLE IV: Relative incidence of the center-most terminal as a source in heuristic PDRev multi-source PD trees that are within $K\%$ suboptimality with respect to the best possible result (minimum $W'_T + S'_T$).

$\lvert V\rvert$	$K\%$				
	1	2	3	4	5
10	0.82	0.80	0.78	0.77	0.76
15	0.87	0.84	0.82	0.80	0.79
25	0.94	0.91	0.87	0.87	0.85
30	0.95	0.93	0.89	0.88	0.85
40	0.92	0.89	0.88	0.88	0.87
45	0.98	0.94	0.89	0.87	0.86
50	0.88	0.88	0.87	0.87	0.87

TABLE V: Runtime (in seconds) of the MSS heuristic with an upper bound of $11 * (1 + \binom{\lambda+1}{1} + \binom{\lambda+1}{2} + \binom{\lambda+1}{3}) = 11 * (1 + 3 + 3 + 1) = 88$ calls to the Multi-Source PD algorithm (averaged over 100 distinct nets).

N	25	30	40	50
Average Time using PDRev (s)	18.09	28.88	56.89	117.04
Average Time using STT (s)	0.06	0.09	0.19	0.34

settings as in [2] for PDRev runs. The large PDRev runtimes reflect the "HVW" implementation noted in Footnote 5.



(a) (b)

Fig. 7: (a) S_T versus $S_{T_{ED}}$ for 550 MSPD trees ($N = 40$). (b) Average S_T and $S_{T_{ED}}$ for 550 MSPD trees at 11 α values ($N = 40$).

We have also confirmed fidelity of linear delay (pathlength) skew S_T to Elmore delay skew $S_{T_{ED}}$. Figure 7 shows data obtained using the NanGate 45nm open-source design enablement [22], with pointsets randomly generated within a 100μm $\times$ 100μm region.[6]

Figure 7(a) shows correlation of Elmore delay and pathlength skews, for 550 PD-generated trees obtained from 50 randomly generated pointsets $N = 40$, crossed with 11 α values ($\{0.0, 0.1, ..., 0.9, 1\}$). Figure 7(b) shows averages of Elmore delay and pathlength skews for the 550 trees.

Study 3: MSPD vs. BST-DME vs. H-Tree. Finally, we have studied MSPD performance relative to two classical constructions that are conscious of the cost-skew tradeoff: Bounded-Skew DME (BST-DME) [10] and H-Tree [5]. We obtain code for BST-DME at the GitHub repository [23]. For our implementation of H-Tree, we recursively construct H's up to k levels, where a Steiner point is added to a quadrant, relative to the current center if there are 2+ sinks in that quadrant. We use the Steiner points generated at the k^{th} level as sources for MSPD.

Figure 8 shows cost-skew Pareto frontiers of L'_T and S'_T for four random pointsets of $N = 45$. MSPD and BST-DME

[6]In this technology, a CLKBUF_X1 instance has estimated driver on-resistance (r_d) of 7.19 kOhm and sink input pin capacitance $7.79 * 10^{-1}$ fF. Interconnect per-micron resistance is 2.18 Ohm, and per-micron capacitance is $9.45 * 10^{-2}$ fF. We calculate Elmore delay at a sink using $ED_i = r_d C_0 + \sum_{e_j \in path(v_0, v_i)} r_{e_j} (\frac{c_{e_j}}{2} + C_j)$ [17], where r_{e_j} and c_{e_j} are respectively the resistance and capacitance of the edge between v_j and its parent, and C_j is the total capacitance of the subtree rooted at v_j.

offer solutions that optimize both criteria, whereas H-Tree generation at $k = 2$ levels produces solutions that heavily optimize skew while increasing cost. We see that MSPD has merits compared to previous methods, and warrants further study.

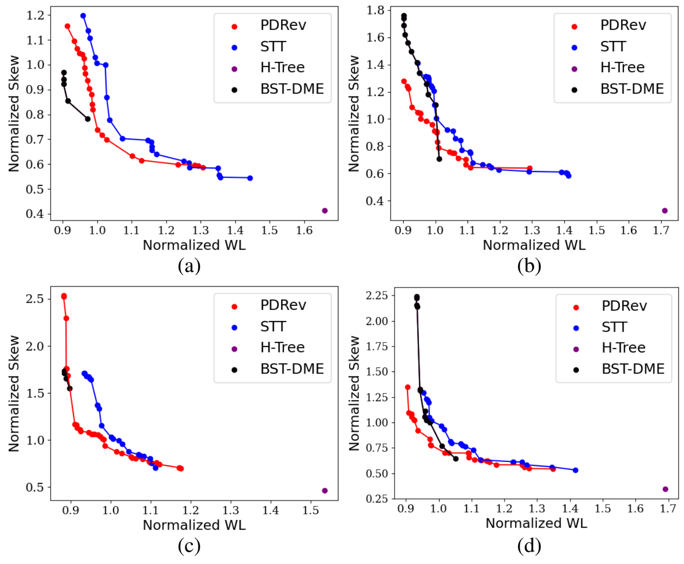


Fig. 8: Results for four pointsets (a), (b), (c) and (d) with $N = 45$ illustrate the constrast among MSPD PDRev (red), MSPD STT (blue), BST-DME (black), and H-Tree at $k = 2$ (purple).

VII. Conclusion

We have studied efficient and practical methods for achieving improved cost-skew tradeoffs in Steiner routing trees. In particular, our work has focused on a "PD-Clock Problem", which seeks to leverage well-established Prim-Dijkstra implementations for better cost-skew outcomes. We have characterized normalized cost vs. normalized skew achievable by PD, across number of terminals, bounding box aspect ratio, centrality of the root, and the PD α parameter. Our data shows a potential unrealized "sweet spot" for use of PD for better cost-skew tradeoff; also, our lookup tables for mean and variance of cost and skew metrics can be used in CTS clustering and buffer placement. To go beyond PD's capabilities, we also study *multi-source* PD variants, and propose a *multi-source selection* heuristic that achieves good empirical performance in reasonable runtimes. The *src* folder of [3] provides open-source implementations of PD, MSPD and MSS (for both PDRev and STT).

Our ongoing work pursues alternative tree metrics to capture the cost-skew tradeoff, as well as machine learning to infer multi-source combinations and α values that yield high-quality multi-source PD trees with improved $J\%$ and $K\%$ metrics. In support of this latter goal, we have set up a machine learning contest in collaboration with The OpenROAD Project [25]. The *contest* folder of [3] introduces the cost-skew tradeoff problem and the goal of identifying best multi-source combinations for a given MSPD instance. It also provides data corresponding to the R' MSPD trees for STT and PDRev described in Section VI above.

VIII. Acknowledgments

We thank Drs. Charles Alpert and Zhuo Li for several preliminary discussions of the PD-Clock problem. We would also like to thank Dr. Kwangsoo Han, Sriram Venkatesh, and Matt Liberty for providing pointers and useful code references for implementations of PDRev.

References

[1] C. J. Alpert, T. Hu, J. Huang, A. B. Kahng and D. Karger, "Prim-Dijkstra Tradeoffs For Improved Performance-driven Routing Tree Design", *IEEE Trans. on CAD* 14(7) (1995), pp. 890–896.
[2] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu and S. Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-Driven Routing Trees", *Proc. ISPD*, 2018, pp. 10–17.
[3] Multi-Source-Prim-Dijkstra [accessed February 15, 2023, commit: ef74adf]. https://github.com/TILOS-AI-Institute/Multi-Source-Prim-Dijkstra
[4] OpenROAD Steiner Tree Builder [accessed January 12, 2023, commit: 879683f]. https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/stt
[5] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, 1990.
[6] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-Based Placement", *IEEE Trans. on CAD* 18(9), (1999), pp. 1265-1278.
[7] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai and A. Tomkins, "Minimizing Wirelength in Zero and Bounded Skew Clock Trees", *Proc. SODA*, 1999, pp. 177-184.
[8] G. Chen, P. Tu and E. F. Young, "SALT: Provably Good Routing Topology by a Novel Steiner Shallow-Light Tree Algorithm", *IEEE Trans. on CAD* 39(6) (2020), pp. 1217-1230.
[9] C.-L. E. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling", *Proc. ICCAD*, 1994, pp. 690-695.
[10] J. Cong, A. B. Kahng, C. K. Koh and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing", *ACM TODAES* 3(3) (1998), pp. 341-388.
[11] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, 1 (1959), pp. 269-271.
[12] H. Fatemi, A. B. Kahng, M. Kim and J. Pineda de Gyvez, "Optimal Bounded-Skew Steiner Trees to Minimize Maximum k-Active Dynamic Power", *Proc. ACM/IEEE SLIP*, 2020, pp. 1-8.
[13] K. Han, A. B. Kahng, C. Moyes and A. Zelikovsky, "A Study of Optimal Cost-Skew Tradeoff and Remaining Suboptimality in Interconnect Tree Constructions", *Proc. ACM/IEEE SLIP*, 2018, pp. 1-8.
[14] K. Han, A. B. Kahng and S. Venkatesh, "UCSD Prim-Dijkstra Revisited". https://openroad.readthedocs.io/en/latest/main/src/stt/src/pdr/
[15] J. M. Ho, G. Vijayan and C. K. Wong, "New Algorithms for the Rectilinear Steiner Tree Problem", *IEEE Trans. on CAD* 9(2) (1990), pp. 185-193.
[16] A. B. Kahng, C. Moyes, S. Venkatesh and L. Wang "Wot the L: Analysis of Real versus Random Placed Nets, and Implications for Steiner Tree Heuristics", *Proc. ISPD*, 2018, pp. 2–9.
[17] A. B. Kahng and G. Robins, *On Optimal Interconnections for VLSI*, Kluwer International Publishers, 1995.
[18] W. Li, Y. Qu, G. Chen, Y. Ma and B. Yu, "TreeNet: Deep Point Cloud Embedding for Routing Tree Construction", *Proc. ASP-DAC*, 2021, pp. 635-640.
[19] R. C. Prim, "Shortest Connecting Networks and Some Generalizations", *Bell System Tech. J.*, 36 (1957), pp. 1389-1401.
[20] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, "The rectilinear steiner arborescence problem" *Algorithmica* 7(2) (1992), pp. 277-288.
[21] Q. Zhu and W. W. M. Dai, "High-speed Clock Network Sizing Optimization Based on Distributed RC and Lossy RLC Interconnect Models", *IEEE Trans. on CAD* 15(9) (1996), pp. 1106-1118.
[22] OpenROAD-flow-scripts [accessed July 30, 2022]. https://github.com/The-OpenROAD-flow/The-OpenROAD-flow-scripts
[23] Bounded-Skew DME v1.3 [accessed November 15, 2022]. https://github.com/abk-openroad/BST-DME
[24] Prim-Dijkstra Revisited [accessed February 12, 2023]. https://github.com/The-OpenROAD-Project/PD-Rev
[25] The OpenROAD Project Website. https://theopenroadproject.org