PatLabor: Pareto Optimization of Timing-Driven Routing Trees

Zhiyang Chen[1], Hailong Yao[2][3], Xia Yin[1]
[1]Tsinghua University, [2]University of Science and Technology Beijing,
[3]Key Laboratory of Advanced Materials and Devices for Post-Moore Chips, Ministry of Education of China

***Abstract*—Wirelength is the fundamental metric for VLSI routing. With the advancement of new technologies, wire delay has also become a significant factor for timing performances. It is thus necessary to consider both wirelength and delay in routing tree construction, i.e., timing-driven routing trees. Prior methods propose various heuristics to balance wirelength and delay with a tunable parameter, which cannot compute the full Pareto frontier. In this work, we propose PatLabor, a practical method for timing-driven routing. PatLabor directly optimizes the Pareto set, which obtains tighter Pareto curves than prior methods and does not require parameter tuning. PatLabor obtains all Pareto-optimal solutions on small-degree nets up to 9 pins and is theoretically guaranteed by provable time complexity and approximation bounds. Experimental results verify our theoretical findings and show that PatLabor obtains tighter Pareto curves than state-of-the-art methods on ICCAD-15 benchmarks. For example, PatLabor obtains up to 58.5% more Pareto-optimal solutions than prior methods for degree-9 nets.**

I. Introduction

Timing has become a critical performance metric for modern VLSI design. As Markov [1] points out, with the advancement of new technologies, wire delay has become a significant factor in performance issues. It is hence necessary to consider both wirelength and delay in the routing stage [2]. Therefore, we study the problem of *timing-driven routing trees*: Given a degree-n net with one pin as the source and other $n-1$ pins as sinks, the objective is to construct a rectilinear Steiner tree topology for these pins that minimize both the total wirelength and the path lengths from the source to sinks. However, these two objectives may be contradictory to each other. Instead of finding a best-of-both-world solution, we aim to find a set of solutions, targeting the Pareto frontier of these two objectives. We study and propose algorithmic methods for bicriterion Pareto optimization of timing-driven routing trees (see Figure 1).

Why Pareto optimization? Computing a Pareto set of solutions for bicriterion problems provides more decision options and better characterizes the relationship between objectives. Recent work on global routing [3] shows that selecting net topologies from a candidate solution set may improve the performance of global routers. Pareto optimization for timing-driven routing trees can be naturally applied to timing optimization in global routing.

A. Our Contributions

Our main contributions are listed as follows:

- We study the size of Pareto frontiers for timing-driven routing trees. We prove that routing instances have only a polynomial size of the Pareto frontier in expectation using smoothed analysis. This provides theoretical foundations for Pareto optimization of timing-driven routing trees. (Section III)
- We propose algorithms for Pareto optimization of timing-driven routing, including an exponential-time exact algorithm and a polynomial-time approximation algorithm. We prove time complexity and approximation bounds. Although these algorithms cannot be directly applied in practice, they serve as important building blocks for our practical methods. (Section IV)

Corresponding author: Hailong Yao. This work was supported by the Key Program of National Natural Science Foundation of China (No. 62034005).

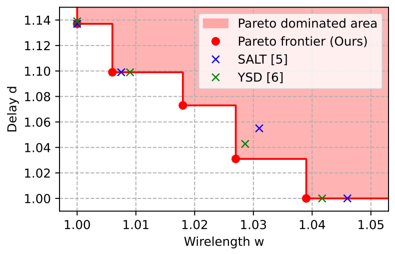


Fig. 1. An example of Pareto optimization from ICCAD-15 benchmarks. Our method obtains the full Pareto frontier while prior methods cannot.

- Based on the theoretical results, we propose **PatLabor**, a practical method for Pareto optimization of timing-driven routing trees. For small-degree nets ($n \leq 9$), we construct lookup tables to compute the Pareto frontier optimally and efficiently. For large-degree nets, we propose a local search heuristic to approximate the Pareto frontier. (Section V)
- Experimental results on ICCAD-15 benchmarks and randomly generated instances illustrate the effectiveness of PatLabor. We first empirically verify our theoretical results in Section III. Then, we present results of constructing lookup tables for small-degree nets and show that our construction method is about $441\times$ faster than FLUTE [4]. Finally, we empirically show that PatLabor obtains tighter Pareto curves than previous timing-driven routing algorithms [5] [6]. For example, PatLabor obtains up to 58.5% more solutions on the Pareto frontier than prior methods for degree-9 nets. (Section VI)

B. Comparisons with Related Work

Rectilinear Steiner minimum trees (RSMT). The most basic routing tree problem is to minimize the total wirelength. Many methods [7] [8] [4] [9] have been proposed to solve RSMT, among which the most popular method is FLUTE [4]. It is worth noting that our idea of using lookup tables for small-degree nets is motivated by FLUTE. FLUTE uses an exhaustive search on boundary compaction to generate lookup tables. However, we propose a new dynamic-program-based method to generate lookup tables, which is more efficient than FLUTE.

Rectilinear Steiner minimum arborescence (RSMA). The RSMA problem [10] [11] [12] aims to find the shortest path Steiner tree for the net with minimum total wirelength. Córdova and Lee (CL) [11] propose a 2-approximation algorithm, which is the most popular for RSMA construction in practice.

Note that RSMT and RSMA are NP-hard [13] [14]. Both RSMT and RSMA are single-objective problems, which are insufficient for timing-driven routing since it has two objectives.

Timing-driven routing trees. Alpert et al. [2] propose Prim-Dijkstra (PD-II), an algorithm blending Prim and Dijkstra to balance the total wirelength and the maximum path length. Chen and Young [5] propose SALT, a method based on a classic shallow-light tree algorithm [15], together with post-processing heuristics. The most recent work is due to Yang, Sun, and Ding (YSD) [6], which combines

TABLE I
THE LIST OF NOTATIONS.

Notation	Implication
$\tilde{O}(f(n))$	$f(n) \cdot (\log n)^{O(1)}$
$O^*(f(n))$	$f(n) \cdot n^{O(1)}$
n	The number of pins
$\|\cdot\|_1$	The l_1 norm (rectilinear distance)
$P = \{p_1, \ldots, p_n\}$	A set of pins with each $p_i = (x_i, y_i)$
(r, P)	A problem instance with source r and pins P
T	A rectilinear Steiner routing tree
$w(T)$	The wirelength of T
$d(T)$	The delay of T
$s(T) = (w(T), d(T))$	The optimization objectives of T
$s_1 \preceq s_2$	Pareto dominance
$S = \{s_i(T_i)\}_i$	A set of solution objectives
$l_1, \ldots, l_{2n-2}$	The lengths of Hanan grids (see Figure 3)

a neural network for small-degree nets, and a divide-and-conquer framework for large-degree nets. More older work [16] [17] [18] are omitted due to the page limit. Prior methods introduce parameters to balance wirelength and delay, which is different from our Pareto optimization method.

We outline some superior features of our method:

The first Pareto optimization method. To the best of our knowledge, this is the first work that directly optimizes the Pareto curve of timing-driven routing. Previous methods introduce parameters to balance the two objectives and generate only one solution for a single parameter value. Tuning parameters is usually painful for heuristic algorithms. Li et al. [19] use a complicated neural network to predict the parameters for PD-II and SALT, which brings heavy training and inference overhead. Our method directly outputs a set of solutions on the Pareto curve, and users can choose any solution they need, without spending time on parameter tuning.

Optimality on small-degree nets. Our method ensures Pareto optimality for nets with $n \leq 9$ pins. Note that none of the prior methods [2] [5] [6] provide optimality guarantees even for nets with $n = 5$ pins. Our method applies an exact dynamic programming algorithm to produce lookup tables containing all potentially optimal tree topologies for small-degree nets, and is hence able to produce all solutions on the Pareto frontier.

Tighter Pareto curves on large-degree nets. For nets with $n > 9$ pins, our method theoretically has the potential to obtain tighter Pareto curves. In Section IV, we prove an approximation bound showing that our method can approximate *every* solution on the Pareto frontier. This is a stronger guarantee compared with SALT, which uses a parameter to control the tradeoff of the approximation ratios. Moreover, our method avoids the issue caused by the weighted sum method in YSD, which only obtains convex Pareto curves.

II. Preliminaries

Notations. Following the convention of algorithm analysis, we use $\tilde{O}(f(n)) = f(n) \cdot (\log n)^{O(1)}$ to suppress logarithmic factors, and use $O^*(f(n)) = f(n) \cdot n^{O(1)}$ to suppress polynomial factors for simplicity of notations.

We follow classic notations for multi-objective optimization. We use $s = (s_1, s_2)$ to denote two objectives of a solution. Since timing-driven routing is a minimization problem, given two solutions $s \neq s'$, we say s *dominates* s' (formally, $s \preceq s'$) if $s_1 \leq s'_1$ and $s_2 \leq s'_2$. The *Pareto curve* (or *Pareto set*) is a set of solutions $\{s_1, \ldots, s_k\}$ such that $s_i \not\preceq s_j$ for any $i \neq j$. The *Pareto frontier* is the optimal Pareto curve. Solutions on the Pareto frontier are *Pareto-optimal.*

Problem formulation. Following previous work [2] [6], we define the *timing-driven routing* problem. We are given a set of points $P =$

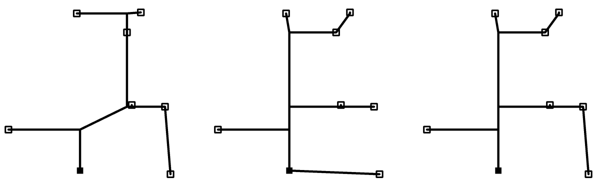


Fig. 2. An example of timing-driven routing trees from ICCAD-15 benchmarks with three Pareto-optimal solutions. Solid squares are source pins. (a) Minimizing total wirelength ($w = 23882, d = 13397$); (b) Minimizing sink delay ($w = 26084, d = 11538$); (c) Balancing wirelength and delay ($w = 24879, d = 11682$).

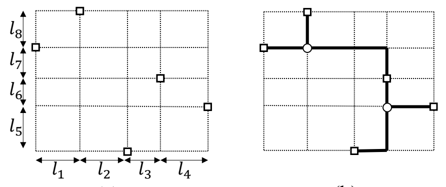


Fig. 3. (a) The Hanan grid; (b) A routing tree constructed on the grid. Squares represent pins and circles represent Steiner points.

$\{p_1, \ldots, p_n\}$ from the metric space $(\mathbb{R}^2, \|\cdot\|_1)$, where each point $p_i = (x_i, y_i)$ denotes a pin from the net to be routed. We use $r = p_1 = (x_1, y_1)$ to denote the *source* of the net and the remaining $n-1$ pins are sinks. The target is to construct a rectilinear Steiner tree $T = (V, E)$ for P rooted at r, considering both the sum of edge lengths of T (wirelength $w(T)$) and the maximum path length from source r to any sink $p_i \in P\, (i \geq 2)$ (delay $d(T)$). Previous work minimizes a weighted sum of $w(T)$ and $d(T)$. We instead aim to compute the Pareto frontier of $(w(T), d(T))$ (an example of three Pareto-optimal solutions is shown in Figure 2).

It is folklore that an optimal RSMT can be constructed on the Hanan grid [20]. We point out that the same holds for the Pareto-optimal timing-driven routing trees. We use $l_1, \ldots, l_{n-1}$ to denote lengths of horizontal grid edges, and $l_n, \ldots, l_{2n-2}$ to denote lengths of vertical edges on the Hanan grid (see Figure 3). The notations are summarized in Table I.

III. Analysis of Pareto Frontiers

In this section, we analyze the Pareto frontier of timing-driven routing trees. It is known that a multi-objective problem usually has an exponential size of the Pareto frontier [21], which makes solving multi-objective optimization intractable (even if $\mathsf{P} = \mathsf{NP}$).

Theorem 1: *There exists a timing-driven routing instance* (r, P) *such that the number of solutions on the Pareto frontier is* $2^{\Omega(n)}$.

Proof Sketch: The construction of the instance is illustrated in Figure 4. The instance consists of m "S-shape" gadgets, with 11 pins each. The m gadgets are placed in a diagonal pattern, with a source pin at the lower-left corner and another pin at the upper-right corner. In the k-th gadget, we set $x = 2^{k-2}$ and $y = 2^{k-1} + 2^{k-3}$. In total, we have $n = 11m + 2$ pins. It can be verified that the number of Pareto-optimal solutions is at least $2^m = 2^{(n-2)/11} = 2^{\Omega(n)}$. □

However, such bad instances never appear in practice. In the following, we show that the size of the Pareto frontier for timing-driven routing in real-world testcases is only polynomial. Based on this result, we present efficient algorithms for computing Pareto frontiers of timing-driven routing in the next section.

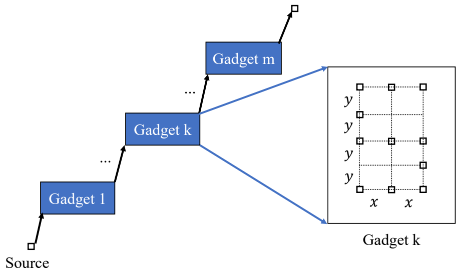


Fig. 4. The construction of an instance with an exponential number of solutions on the Pareto frontier in Theorem 1.

One way of modeling real-world instances is to use average-case analysis, assuming the pins are uniformly random points on the routing plane. However, this assumption is too strong. In practice, the problem instance cannot be seen as purely random instances on the entire routing plane. Alternatively, our idea is to study the Pareto frontier using *smoothed analysis*, which is a popular algorithmic framework to explain the practical behavior of many algorithms proposed by Spielman and Teng [22]. The philosophy is that in practice, there is always some small noise in the problem instance, and the performance of an algorithm seldom achieves the worst-case bound. In timing-driven routing, the positions of pins are generated by VLSI placement. A small perturbation on the positions of a pin does not impact placement-and-routing results greatly. The assumption is hence reasonable.

Smoothed analysis interpolates between worst-case analysis and average-case analysis. It is a relaxation of worst-case analysis, but with a weaker assumption than average-case analysis.

Definition 1 (Smoothed timing-driven routing instances): Without loss of generality, we assume the coordinates of pins in the routing instance $P = \{p_1, \ldots, p_n\}$ lie in $[0, 1]$. In a κ*-smoothed instance*, the coordinates x_i and y_i of pins are sampled independently from distributions with probability density at most κ on $[0, 1]$.

For example, consider a degree-2 net with x_1 uniformly sampled from $[0, 0.25]$, y_1 sampled from $[0.1, 0.35]$, x_2 sampled from $[0.6, 0.85]$ and y_2 sampled from $[0.65, 0.9]$. We say this net is a 4-smoothed instance. As $\kappa \to \infty$, the framework reduces to worst-case analysis. If $\kappa = 1$, the framework reduces to average-case analysis.

Theorem 2: *The expected number of solutions on the Pareto frontier for κ-smoothed timing-driven routing is $O(n^3\kappa)$.*

Proof Sketch: Due to the page limit, we only sketch the proof ideas. The main proof technique is to divide the range of objectives into small intervals. Since the edge weights are κ-smoothed, the probability density is at most κ. Therefore, the probability of a Pareto-optimal solution falling into an interval of length l is only $l \cdot \kappa$. The total length of all intervals is only $\text{poly}(n)$, and the expected number of optimal solutions is hence at most $\text{poly}(n) \cdot \kappa$. □

In practice, we assume the nets contain small randomness, and thus regard κ as a constant. Based on Theorem 2, it is possible to design efficient algorithms for timing-driven routing trees.

IV. Theoretical Algorithms for Timing-Driven Routing

In this section, we present two theoretical algorithms for timing-driven routing trees. The time complexities of both algorithms depend on the size of Pareto frontiers, and are thus efficient by Theorem 2. Although they cannot be directly applied in practice, their ideas serve as significant building blocks for our practical method in the next section.

A. An exponential-time exact algorithm

Dreyfus-Wagner (DW) [23] is a classic algorithm for minimum Steiner trees on graph metrics, which finds an exact solution in $O^*(3^n)$ time using a dynamic program. We can adapt this algorithm to compute the full Pareto frontier of timing-driven routing trees.

Our algorithm, namely **Pareto-DW**, is described as follows: Let $S_{v,Q}$ denote the Pareto frontier of routing trees with source v and sinks Q. Here, v is an arbitrary node on the Hanan grid, and $Q \subseteq P$ is a subset of pins. Each $S_{v,Q}$ is a Pareto set of solution objectives (w, d). Then, $S_{r,P}$ is our desired result. We compute $S_{v,Q}$ using dynamic programming,

$$S_{v,Q} = \text{Pareto} \begin{cases} \cup_u \{S_{u,Q} + \|u - v\|_1\}, \\ \cup_{Q_1 \subseteq Q} \{S_{v,Q_1} \oplus S_{v,Q \setminus Q_1}\}. \end{cases} \quad (1)$$

Note that $S_{v,Q}$ is a set of vectors. For any solution sets S, S' and any $x \in \mathbb{R}$, we define

$$\begin{aligned} S + x &= \{(w + x, d + x) \mid (w, d) \in S\}, \\ S \oplus S' &= \{(w_1 + w_2, \max\{d_1, d_2\}) \\ &\qquad \mid (w_1, d_1) \in S, (w_2, d_2) \in S'\}. \end{aligned}$$

Note that $S + x$ and $S \oplus S'$ can be computed in $O(|S|)$ and $O(|S| \cdot |S'|)$ time, respectively. We also define $\text{Pareto}(S) = \{s \in S \mid \nexists s' \in S, s' \preceq s\}$, i.e., eliminating all solutions in S off the Pareto curve. Note that this is equivalent to finding maximal points on a plane, which can be computed in $O(|S| \log |S|)$ time [24].

Theorem 3: *Pareto-DW finds the Pareto frontier of timing-driven routing in $O^*(3^n \cdot |S|^2)$ time, where $|S|$ is the largest number of solutions in $S_{v,Q}$ for any v, Q.*

This theorem can be proved by exploiting the recursive structure of Steiner trees, and the proof is omitted due to the page limit. We point out that by Theorem 2, the size of Pareto frontier $|S|$ is only polynomial in n. Therefore, in practice, we can solve timing-driven routing using $O^*(3^n)$ time, the same as the vanilla Dreyfus-Wagner. Without smoothed analysis, the time complexity may become $O^*(3^n) \cdot 2^{O(n)} \cdot 2^{O(n)} = 12^{O(n)}$, which is unacceptable even for $n \leq 10$ pins.

There are also many acceleration techniques for Pareto-DW. Although they cannot improve the worst-case complexity theoretically, they can effectively prune the states of the dynamic program. We introduce them in Section V.

B. A polynomial-time approximation algorithm

Pareto-DW can only solve small-degree instances due to the exponential complexity. Using Pareto-DW as a subroutine, we can design a polynomial-time approximation algorithm for timing-driven routing. The idea is to extend the Kalpakis-Sherman (KS) heuristic[1] [26] to a multi-objective version: We use divide-and-conquer to partition the routing plane into sub-problems. When the number of pins is small enough, we use Pareto-DW to compute the exact Pareto frontier. We formally describe the algorithm as follows.

Algorithm Pareto-KS(P):

1. If $|P| \leq \log n$, return Pareto-DW(P) to obtain the Pareto frontier.
2. We find a pin $p_i \in P$ such that there are at least $\lfloor |P|/2 \rfloor - 1$ pins on each side of p_i (divide on the x- or y-axis alternatively) to partition the pins into P_1 and P_2.

[1]In essence, the classic RSMT heuristic, FLUTE [4], is a variant of the KS algorithm. The effectiveness of KS in practice is the reason why we choose to adapt KS instead of other algorithms, e.g., Arora's PTAS for geometric Steiner trees [25] (although Arora's may achieve better approximation bounds).

3. Call Pareto-KS(P_1) and Pareto-KS(P_2) to obtain S_1 and S_2 recursively. We select the pin closest to r as the source pin.
4. Return the combination of the solutions in S_1 and S_2.

Definition 2 (Pareto approximation): Given a multi-objective minimization problem, and an algorithm that outputs a Pareto set S for any instance of the problem, we say the algorithm *c-approximates* the Pareto frontier, if, for any solution s on the Pareto frontier, we can find $s' \in S$, such that $s' \preceq c \cdot s$.

Theorem 4: *Pareto-KS is an $O(\sqrt{n/\log n})$-approximation algorithm in $\tilde{O}(n^2 \cdot |S|^2)$ time, where $|S|$ is the size of the largest Pareto set during the execution of Pareto-KS.*

Proof Sketch: We only prove the time complexity due to the page limit. The proof of the approximation ratio can be obtained by showing that timing-driven routing is a *sub-additive Euclidean functional* [26] and is omitted.

Pareto-KS partitions the routing plane into $\Theta(n/\log n)$ sub-rectangles. In each rectangle, there are at most $\log n$ pins. The complexity of computing the Pareto frontier in each rectangle is hence $O(3^{\log n} \cdot \mathrm{poly}(\log n)) = \tilde{O}(n)$. Given two Pareto sets S_1, S_2, we can compute their Pareto sum in $\tilde{O}(|S_1| \cdot |S_2|)$ time. Therefore, the total complexity is $\tilde{O}(n) \cdot O(n/\log n) \cdot \tilde{O}(|S|^2) = \tilde{O}(n^2 \cdot |S|^2)$. $\square$

V. Practical Methods for Timing-Driven Routing

In this section, based on Pareto-DW and Pareto-KS, we present **PatLabor** (Pareto optimization with Lookup tables and local search), a practical method for timing-driven routing. PatLabor consists of two parts: For small-degree nets, we build lookup tables to efficiently and optimally solve the problem. For large-degree nets, we present a local search heuristic to approximate the Pareto frontier.

A. Lookup tables for small-degree nets

Although the Pareto-DW algorithm is fast for small-degree nets, it is not efficient enough to route millions of nets in VLSI designs. Motivated by the lookup-table-based FLUTE heuristic [4] for RSMTs, we store all potentially Pareto-optimal routing tree topologies in a lookup table for all instances with $n \le \lambda$ so that we can directly access the Pareto frontier efficiently. Following FLUTE, we set $\lambda = 9$. We use modified Pareto-DW to generate the lookup tables.

For each possible permutation of pins P and a source r, we can build a Hanan grid for pins and use $l_1, \ldots, l_{2n-2}$ to denote the lengths of grid edges as in Figure 3. For degree-n nets, there are at most $n!$ different patterns of the Hanan grid. As in Pareto-DW, we use $S_{v,Q}$ to denote the set of potential Pareto-optimal topologies with source v and pins Q. We follow the DP Equation (1). However, a solution in $S_{v,Q}$ is no longer $(w, d) \in \mathbb{R}^2$, but in the form of

$$\left(\sum_{i=1}^{2n-2} w_i l_i, \max_i \sum_{j=1}^{2n-2} d_{ij} l_j \right).$$

We instead use a vector $W = (w_i)$ and a matrix $D = (d_{ij})$ to represent a solution. By definition, we have the following proposition.

Lemma 1: *Given two solutions $(W^{(1)}, D^{(1)})$ and $(W^{(2)}, D^{(2)})$, $(W^{(2)}, D^{(2)})$ can be safely pruned if for any $l_1, \ldots, l_{2n-2} \ge 0$,*

$$\sum_{i=1}^{2n-2} (w_i^{(2)} - w_i^{(1)}) \ge 0 \wedge \max_i \sum_{j=1}^{2n-2} d_{ij}^{(1)} l_j \le \max_i \sum_{j=1}^{2n-2} d_{ij}^{(2)} l_j. \quad (2)$$

The condition in Equation (2) is a first-order proposition, and hence can be verified by an SMT solver. We use an SMT solver [27] to safely prune non-optimal solutions during the generation of lookup tables. Since the number of pins is small, the SMT solver is efficient enough.

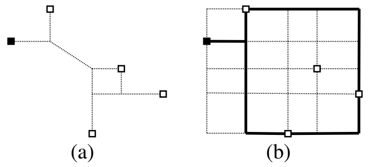


Fig. 5. Pruning techniques for Pareto-DW. The solid square is the source pin. (a) The pruned Hanan grid for Figure 3 after applying Lemma 2; (b) An example of the source on the outside of the bounding box for Lemma 3.

To accelerate generating lookup tables, we apply some reduction tricks. We reduce the number of Hanan grid patterns by breaking symmetries. There are in total $n! \cdot n$ choices of (r, P). However, if two patterns are equivalent under mirror and rotation transformations, only one pattern is needed to store in the table. Another important trick is to prune useless states in dynamic programming (Figure 5(a)).

Lemma 2: *A node $v = (x, y)$ on the Hanan grid is called a lower-left-corner node*[2]*, if there are no pins $p_i = (x_i, y_i) \in P$ such that $x_i \le x$, $y_i \le y$. The states $S_{v,Q}$ for any lower-left-corner node v can be safely pruned in Equation (1).*

We can also avoid subset enumeration in the dynamic program for many states (Figure 5(b)).

Lemma 3: *For a node v on the Hanan grid and a subset of pins $S \subseteq P$, if v is outside $\mathsf{BB}(S)$ (the bounding box of S), we have $S_{v,Q} = S_{u,Q} + \|v - u\|_1$, where u is the projection of v on $\mathsf{BB}(S)$.*

The last technique is based on *separators* for *outer-planar* graphs. For the definition of separators, see any parameterized algorithm textbooks, e.g., Cygan et al. [28].

Lemma 4: *For a node v and a subset of pins $S \subseteq P$, if all pins in S lie on the boundary of the grid and we relabel them by $1, 2, \ldots, |S|$ in the clockwise order (the labels are circular, i.e., $|S|$ and 1 are adjacent), then, in Equation (1), it suffices to consider transitions $S_{v,Q_1} \oplus S_{v,Q \setminus Q_1}$ for Q_1 such that the pins have consecutive labels.*

Lemmas 2, 3 and 4 can greatly reduce the running time of Pareto-DW. We omit the proofs of lemmas due to the page limit.

B. Local search heuristics for large-degree nets

For nets with degree $n > \lambda$, we reduce the problem to small-degree nets and apply the lookup table. A direct method is to replace dynamic programming in Pareto-KS with lookup tables.

Remark 1: If we use lookup tables for nets with degree-λ or less in Pareto-KS, we can instead obtain an $O(\sqrt{n/\lambda})$-approximation bound and an $\tilde{O}(n\lambda|S|^2)$ time bound in Theorem 4. In practice, most nets have $n \le 50$ pins, and hence n/λ can be seen as $O(1)$.

Although Pareto-KS is good in theory, it is not good enough in practice. The main reason is that the divide-and-conquer framework cannot appropriately exploit the relative positions of the source pin r and a sub-problem. Following the philosophy of Pareto-KS, we instead apply a local search method. We still reduce the problem to sub-problems, but focusing on pins with large delay. The algorithm is formally described as follows.

PatLabor for $n > \lambda$:

1. We maintain a Pareto set of tree solutions $\mathcal{T}$. Initially, we call FLUTE [4] to build an RSMT T_0 and set $\mathcal{T} = \{T_0\}$.
2. Select a tree $T \in \mathcal{T}$ with the largest maximum delay $d(T)$. Choose $\lambda - 1$ pins in T based on a given policy π. Together with source r, regenerate the topology of these λ pins by accessing lookup tables and obtain a set of new tree topologies $T_1, \ldots, T_m$.

[2]We can similarly define corner nodes on the other three corners.

3. Let $\mathcal{T} \leftarrow \mathcal{T} \cup \{T_1, \ldots, T_m\}$. Eliminate solutions in $\mathcal{T}$ that are off the Pareto curve.
4. Repeat steps 2 and 3 for $\lfloor n/\lambda \rfloor$ times.

Note that if we restrict the local search to access each pin only once, the algorithm is indeed a variant of Pareto-KS. In step 2, after a local search iteration, the new topologies may be sub-optimal since the local topology may be intersected with other $n - \lambda$ pins. We use post-processing techniques as in SALT [5] to refine these issues.

An important component in PatLabor is the policy π to select pins on which we perform local search. We define a heuristic scoring function for each pin, and iteratively select $\lambda - 1$ pins with the maximum score. Suppose we have already selected λ' pins $p_1, \ldots, p_{\lambda'}$, for each unselected pin p, let

$$\begin{aligned}\mathsf{score}(p) &= \alpha_1 \cdot \|r - p\|_1 + \alpha_2 \cdot \mathsf{dist}_T(r, p) \\ &- \alpha_3 \cdot \min_{1 \le \lambda_0 \le \lambda'} \|p - p_{\lambda_0}\|_1 - \alpha_4 \cdot \mathsf{HPWL}(p, p_1, \ldots, p_{\lambda'}),\end{aligned}$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \ge 0$ are parameters. The first term $\|r - p\|_1$ is the rectilinear distance between source r and p. The second term $\mathsf{dist}_T(r, p)$ is the distance between r and p on the current tree topology T. We always select pins far from the source such that the delay is large. The third term $\min_{1 \le \lambda_0 \le \lambda'} \|p - p_{\lambda_0}\|_1$ is the minimum distance between p and already selected pins. The fourth term $\mathsf{HPWL}(p, p_1, \ldots, p_{\lambda'})$ is the half-perimeter wirelength of p and already selected pins. If no pins have been selected, these two terms are zero. After an iteration of local search, the selected pins are connected in a subtree. We hope these pins are near to each other. Thus, it is better for the first two terms to be large and the last two terms to be small.

We select the best parameters for policy π using a *reinforcement learning* apparatus. We use *policy iteration* [29] to train the parameters α. Concretely, we first randomly sample the selection of pins and run PatLabor on randomly generated testcases. We collect a set of selections of pins $(p_1, \ldots, p_{\lambda-1})$ such that the algorithm performances are better, and use linear regression to maximize the score function on these collected data. Motivated by the philosophy of *curriculum learning* [30], we choose different values of $\alpha^{(n)}$ for different degrees of n. We initially train the value of $\alpha^{(n)}$ for $n = \lambda + 1 = 10$. Suppose we have already selected $\alpha^{(n)}$ for some degree-n. Then, we use $\alpha^{(n)}$ as a warm-start for degree-$(n+1)$ and further optimize $\alpha^{(n+1)}$. Finally, we obtain the policy π for each $n \ge 10$. We finish training when $n = 100$ since most nets in our experiments have at most 100 pins.

We can use statistical learning theory to guarantee the generalization ability of the learned parameters. The following theorem is obtained by an upper bound on the pseudo-dimension [31] of the performance metric. We omit the proof due to the page limit.

Theorem 5: *Let $\mathcal{D}$ be an arbitrary distribution of degree-n timing-driven routing instances. Suppose the parameters α are learned based on m i.i.d. samples $x_1, \ldots, x_m$ from $\mathcal{D}$. Let $\mathsf{perf}(x, \alpha)$ denote any performance metric of PatLabor with parameters α on instance x. Then, with probability at least $1 - \delta$, for any $\alpha \in \mathbb{R}^4$, we can bound the generalization gap by*

$$\left| \frac{1}{m} \sum_{i=1}^{m} \mathsf{perf}(x_i, \alpha) - \mathbb{E}_{x \sim \mathcal{D}}[\mathsf{perf}(x, \alpha)] \right| \le \tilde{O}\left(\sqrt{\frac{n}{m}} \right).$$

VI. Experimental Results

In this section, we present experimental results. Following previous work, we use the ICCAD-15 benchmark to evaluate PatLabor and other baselines. The benchmark consists of 8 designs, with about 1.3×10^6 nets (we omit degree-2 and -3 nets since they are trivial).

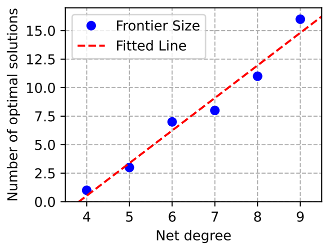


Fig. 6. Sizes of the maximum Pareto frontier from the ICCAD-15 benchmark. The fitted line ($y = 2.85x - 10.9$) is computed by linear regression.

TABLE II
Statistics of the Lookup Table.

Degree	#Index	#Topo	Size (MB)	Time
4	24	1.67	< 0.01	0s
5	220	4.6	< 0.01	0s
6	1008	10.67	< 0.01	0s
7	5824	32.52	0.19	4.9s
8	46880	107.05	6.23	276s
9	429516	378.05	240	4.68h
Total	483472	-	246	4.76h

#Index = the number of pairs (r, P), i.e., the source pin and the pin permutation.
#Topo = the average number of potentially optimal tree topologies.
Size = the storage size of lookup tables.
Time = (parallel) running time of generating the lookup table.

A. The size of Pareto frontiers

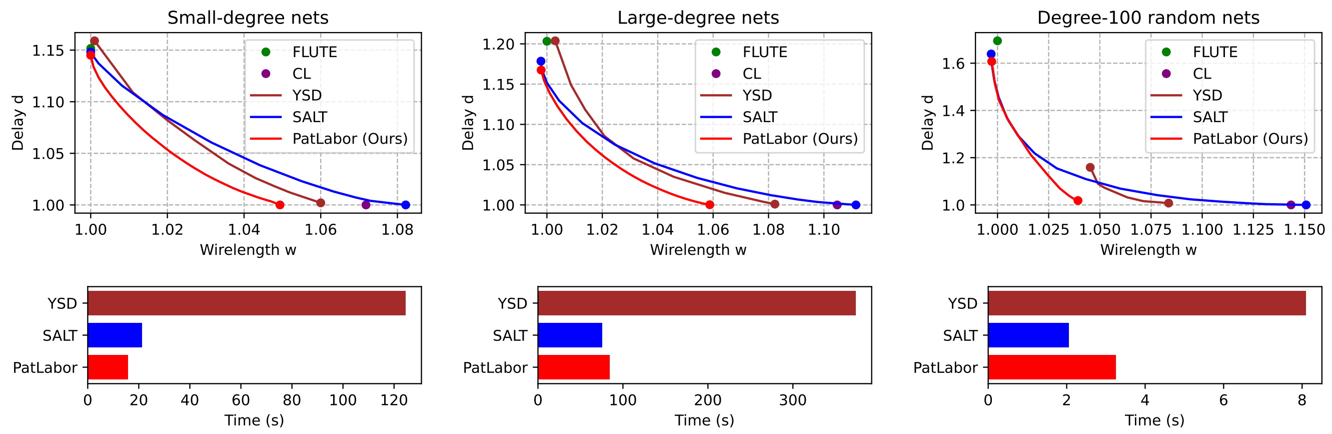
The analysis of Pareto frontiers in Section III is verified empirically. We obtain the number of solutions on the Pareto frontier. For all nets with degree $n \le 9$, the maximum size of the Pareto frontier is computed. The result is illustrated in Figure 6. Although the worst-case bound on the Pareto frontier size is exponential, the number of optimal solutions only grows roughly $\sim 2.85n$ on real-world testcases. For $n = 9$, the maximum Pareto frontier size is only 16. This verifies our analysis in Theorems 1 and 2, but also illustrates our bound is still slightly loose.

B. Generation of lookup tables

We present experimental results on generating lookup tables using the proposed Pareto-DW in Section V. Table II lists the statistics of our lookup tables. The lookup tables are generated on an Intel Xeon 2.30GHz CPU with 16 cores. The total number of tree topologies is about 1.7×10^8. Using 16 threads, we generate all lookup tables in less than 5 hours. The serial running time is 49.9 hours. As a rough comparison, FLUTE [4] only generates about 4.5×10^5 RSMT topologies[3] using 58.2 hours (on a 3.40GHz Intel Pentium 4 CPU). Our method is about $(1.7 \times 10^8)/(4.5 \times 10^5) \times (58.2/49.9) \approx 441\times$ on average faster than FLUTE for each topology.

Directly storing all topologies results in a huge table. We find that for a single set of pins with different sources, many topologies are the same. Moreover, some groups of pin sets have similar topologies with slight differences on the boundary pins. We group them into clusters and store only one topology for each cluster. This greatly reduces the table size. The tables can be read into memory in less than 5 seconds, occupying < 250MB memory space, which is negligible for routing millions of nets. It is also faster than loading neural network weights compared with YSD [6], which trains a model for each degree n and each weighted sum parameter.

[3]Our number of tree topologies is much larger than RSMT's since timing-driven routing trees (a) depend on the position of the source; (b) consider two optimization objectives.



(a) Small-degree nets ($n \leq 9$) (b) Large-degree nets ($n > 9$) (c) Degree-100 randomly generated nets

Fig. 7. Averaged Pareto curve and the total running time comparisons on (a) small-degree nets; (b) large-degree nets; (c) 100 randomly generated degree-100 nets. The green circle represents RSMT by FLUTE and the purple circle represents RSMA by CL. The wirelength w and the delay d are normalized by dividing $w(\text{FLUTE})$ and $d(\text{CL})$. Note that the Pareto curves for small-degree nets are averaged on non-optimal nets as in Table III.

TABLE III
THE RATIO OF NON-OPTIMAL NETS FOR $n \leq 9$.

n	#Net	PatLabor	YSD [6]	SALT [5]
4	364670	**0.0%**	**0.0%**	**0.0%**
5	256663	**0.0%**	0.3%	0.9%
6	103199	**0.0%**	7.8%	11.9%
7	75055	**0.0%**	23.3%	24.3%
8	42879	**0.0%**	36.0%	34.7%
9	62449	**0.0%**	49.5%	45.4%
Total	904915	**0.0%**	8.0%	8.4%

TABLE IV
THE TOTAL NUMBER OF SOLUTIONS ON THE PARETO FRONTIERS FOUND BY THREE METHODS FOR $n \leq 9$.

n	PatLabor	YSD [6]	SALT [5]
4	**364670**	**364670**	**364670**
5	**297636**	296634	295053
6	**137460**	128237	123590
7	**116941**	100012	92083
8	**77325**	49398	52767
9	**132487**	72105	77483
Total	**1.0**	0.898	0.893

C. Comparisons on Pareto curves

We implement PatLabor using C++ and compare it with two state-of-the-art timing-driven routing tree algorithms, SALT [5] and YSD [6]. As the open-source code of YSD is incomplete, we implement some components of YSD based on the paper. All experiments are executed on an Intel Xeon 2.30GHz CPU with 16 cores, except YSD, which uses a neural network model and is evaluated on an NVIDIA H100 GPU.

For **small-degree nets**, we first evaluate the ratio of non-optimal nets in Table III. An algorithm is non-optimal on a net if it cannot find at least one solution on the Pareto frontier. The usage of lookup tables ensures the optimality of PatLabor, while both YSD and SALT are non-optimal for $n \geq 5$. (We run YSD and SALT with different parameters to obtain Pareto sets.) Among 904,915 nets with degree $n \leq 9$, there are 72,393 (resp. 76,013) nets on which YSD (resp. SALT) cannot achieve the Pareto frontier. Specifically, for $n = 9$, there are 30,912 (resp. 28,352) out of 62,449 nets, on which YSD (resp. SALT) is non-optimal. In contrast, there are **zero** non-optimal nets for PatLabor. We also collect the total number of optimal solutions found by three methods for $n \leq 9$ in Table IV. There are 115,463 (resp. 120,873) out of 1,126,519 solutions on the Pareto frontier that YSD (resp. SALT) cannot find. Specifically, for $n = 9$, there are 60,382 (resp. 55,004) out of 132,487 solutions that YSD (resp. SALT) cannot find. PatLabor finds **all** Pareto-optimal solutions.

Figure 7(a) shows the Pareto curves of different methods for small-degree nets. Since the baselines achieve optimal Pareto curves on many small-degree nets, the curves are only averaged on **non-optimal** nets by SALT and YSD as in Table III (i.e., either SALT or YSD is non-optimal on these nets). PatLabor achieves the tightest Pareto curve and is the fastest method due to the use of lookup tables[4]. In total, PatLabor is about $1.35\times$ faster than SALT.

For **large-degree nets**, the result is presented in Figure 7(b). PatLabor also achieves the tightest Pareto curves. However, due to the running time of combining Pareto sets, PatLabor is about 11.6% slower than SALT, but still much faster than YSD. Note that most nets in the ICCAD-15 benchmark have less than 50 pins. To compare the performances of different methods comprehensively, we also randomly synthesize 100 nets with 100 pins, and evaluate all methods. The result is presented in Figure 7(c). The Pareto curve of PatLabor is almost the same as SALT's for low total wirelength w, but is tighter than SALT's for high total wirelength. Note that YSD uses a divide-and-conquer framework for large-degree nets, which performs poorly for wirelength minimization.

VII. Conclusion

In this work, we study the timing-driven routing tree problem. We first show the size of Pareto frontiers is only polynomial under the smoothed analysis setting. Then, we propose theoretical algorithms and a practical method, PatLabor, for timing-driven routing. Finally, experimental results show that PatLabor obtains tighter Pareto curves compared with state-of-the-art methods. A promising direction of future work is to extend our approach to other metrics of routing trees, such as congestion, multi-patterning lithography, and reliability. It would also be interesting if timing-driven routing trees could be integrated into global routing or even global placement engines.

[4]Note that YSD is much slower than SALT and PatLabor. Although it is claimed in the paper [6] that YSD is more efficient than SALT, their comparison is not fair. YSD uses batch inference on GPUs for the neural model and SALT is only evaluated serially. In fact, SALT (and PatLabor, of course) can be accelerated by multi-threading on the CPU.

References

[1] I. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, pp. 147–54, 2014.

[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," in *International Symposium on Physical Design*, 2018, p. 10–17.

[3] W. Li, R. Liang, A. Agnesina, H. Yang, C.-T. Ho, A. Rajaram, and H. Ren, "DGR: Differentiable global router," in *Design Automation Conference*, 2024.

[4] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2008.

[5] G. Chen and E. F. Y. Young, "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1217–1230, 2020.

[6] L. Yang, G. Sun, and H. Ding, "Towards timing-driven routing: An efficient learning based geometric approach," in *International Conference on Computer Aided Design*, 2023, pp. 1–9.

[7] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 30, no. 1, pp. 104–114, 1976.

[8] A. Kahng and G. Robins, "A new class of iterative Steiner tree heuristics with good performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 7, pp. 893–902, 1992.

[9] J. Liu, G. Chen, and E. F. Young, "REST: Constructing rectilinear Steiner minimum tree via reinforcement learning," in *58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1135–1140.

[10] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, "The rectilinear Steiner arborescence problem," *Algorithmica*, vol. 7, no. 1–6, p. 277–288, 1992.

[11] J. Córdova and Y.-H. Lee, "A heuristic algorithm for the rectilinear Steiner arborescence problem," 1994.

[12] K.-S. Leung and J. Cong, "Fast optimal algorithms for the minimum rectilinear Steiner arborescence problem," in *1997 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 3, 1997, pp. 1568–1571.

[13] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM Journal of Applied Mathematics*, vol. 32, pp. 826–834, 1977.

[14] W. Shi and C. Su, "The rectilinear Steiner arborescence problem is NP-complete," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 729–740, 2005.

[15] M. Elkin and S. Solomon, "Steiner shallow-light trees are exponentially lighter than spanning ones," in *IEEE 52nd Annual Symposium on Foundations of Computer Science*, 2011, pp. 373–382.

[16] J. Cong, A. Kahng, G. Robins, M. Sarrafzadeh, and C. Wong, "Provably good performance-driven global routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 6, pp. 739–752, 1992.

[17] S. Khuller, B. Raghavachari, and N. Young, "Balancing minimum spanning trees and shortest-path trees," *Algorithmica*, vol. 14, no. 4, p. 305–321, 1995.

[18] R. Scheifele, "Steiner trees with bounded RC-delay," *Algorithmica*, vol. 78, no. 1, p. 86–109, 2017.

[19] W. Li, Y. Qu, G. Chen, Y. Ma, and B. Yu, "TreeNet: Deep point cloud embedding for routing tree construction," in *Asia and South Pacific Design Automation Conference*, 2021, pp. 164–169.

[20] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 14, no. 2, pp. 255–265, 1966.

[21] A. Herzel, S. Ruzika, and C. Thielen, "Approximation methods for multiobjective optimization problems: A survey," *INFORMS Journal on Computing*, vol. 33, no. 4, pp. 1284–1299, 2021.

[22] D. A. Spielman and S.-H. Teng, "Smoothed analysis: an attempt to explain the behavior of algorithms in practice," *Communications of the ACM*, vol. 52, no. 10, p. 76–84, 2009.

[23] S. E. Dreyfus and R. A. Wagner, "The Steiner problem in graphs," *Networks*, vol. 1, no. 3, pp. 195–207, 1971.

[24] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, pp. 132–133, 1972.

[25] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," *Journal of the ACM*, vol. 45, no. 5, p. 753–782, 1998.

[26] K. Kalpakis and A. T. Sherman, "Probabilistic analysis of an enhanced partitioning algorithm for the Steiner tree problem in $\mathbb{R}^d$," *Networks*, vol. 24, no. 3, pp. 147–159, 1994.

[27] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[28] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*. Springer, 2015.

[29] D. Bertsekas, *Reinforcement Learning and Optimal Control*, 2019.

[30] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, p. 41–48.

[31] R. Gupta and T. Roughgarden, "A PAC approach to application-specific algorithm selection," *SIAM Journal on Computing*, vol. 46, no. 3, pp. 992–1017, 2017.