Sign-Off Timing Considerations via Concurrent Routing Topology Optimization

Siting Liu, Ziyi Wang, Fangzhou Liu, Yibo Lin, *Member, IEEE*, Bei Yu, *Senior Member, IEEE*, and Martin D. F. Wong, *Fellow, IEEE*

***Abstract*—Timing closure is considered across the circuit design flow. Generally, the early stage timing optimization can only focus on improving early timing metrics, e.g., rough timing estimation using linear RC model or prerouting path length, since obtaining sign-off performance needs a time-consuming routing flow. However, there is no consistency guarantee between early stage metrics and sign-off timing performance. Therefore, we utilize the power of deep learning techniques to bridge the gap between the early stage analysis and the sign-off analysis. A well-designed deep learning framework guides the adjustment of Steiner points to enable explicit early stage timing optimization. Cooperating with deep Steiner point adjustment, we propose the routing topology reconstruction to accelerate the convergence and hold a reasonable routing topology. Further, we also introduce Steiner point simplification as a post-processing technique to avoid unnecessary routing constraints. This article demonstrates the ability of the learning-assist framework to perform robust and efficient timing optimization in the early stage with comprehensive and convincing experimental results on real-world designs. With Steiner point adjustment alone, TSteiner^{Pt}, can help the state-of-the-art open-source router to obtain 11.2% and 7.1% improvement for the sign-off worst-negative slack and total negative slack, respectively. Under the additional joint optimization with routing topology reconstruction and simplification, TSteiner^{Rec} can further save 25.9% optimization duration with a better-sign-off performance.**

***Index Terms*—Graph neural networks, steiner trees, timing.**

I. Introduction

TIMING closure [1] is critical but challenging to meet after the complete VLSI physical design due to the explosive growth in transistors. Usually, the design stages like placement and routing (PnR) are invoked iteratively to subtly and tediously adjust the gate locations and metal wire connections. It is considerably time-consuming to satisfy the sign-off timing closure through numerous PnR iterations.

Received 5 April 2024; revised 30 July 2024 and 26 September 2024; accepted 27 October 2024. Date of publication 27 November 2024; date of current version 23 April 2025. This work was supported in part by the Research Grants Council of Hong Kong, SAR, under Project CUHK14211824, and in part by the MIND under Project MINDXZ202404. This article was recommended by Associate Editor I. H. R. Jiang. *(Corresponding author: Bei Yu.)*

Siting Liu, Ziyi Wang, Fangzhou Liu, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Yibo Lin is with the School of Integrated Circuits, Peking University, Beijing 100871, China.

Martin D. F. Wong is with the Computer Science Department, Hong Kong Baptist University, Hong Kong, SAR.

Digital Object Identifier 10.1109/TCAD.2024.3506216

Researchers have explored timing considerations in the early physical synthesis stages to address this timing closure issue. Retiming [2] in the logic synthesis stage relocates registers to reduce the cycle time while preserving the circuit's functionality. In the placement stage, the net-weighting [3] and differentiable timing objective function [4] are proposed for timing-driven global placement. The Lagrangian multipliers [5] help to optimize early timing metrics in the detailed placement stage. However, only the prerouting timing metrics, e.g., the Elmore model analysis on the generated Steiner trees, are considered in these works, which have a considerable gap to the accurate sign-off timing metrics.

Further, early stage timing optimization studies have also been pushed to the routing stage. Sign-off timing performance is significantly affected by the routing strategies. Modern routing frameworks always set wirelength as the first-order optimization objective. Specifically, the most widely used routers apply the Steiner minimum tree construction to split the multipin nets into a set of two-pin nets to control the routed wirelength. The generated Steiner trees limit the routing topology during the following routing process and somehow determine the post-route performance. To consider timing performance in the routing topology generation stage, researchers [6], [7] have set the path length minimization as the optimization target with strategies like min-max resource sharing [8] and [9] considers a novel Steiner shallow-light tree algorithm for path length control. In the detailed routing stage, timing optimization was also considered during track assignment via net weighting and detour [10], [11]. Besides optimizing metal wire connections, [12], [13] integrated timing-driven consideration to the layer assignment algorithms and benefited from the availability of different metal layers. Various efforts have been applied to the early physical synthesis stage to save the turnaround time under the timing closure requirement. Notably, all the works above are not directly targeted at the sign-off timing performance but optimize the early timing metrics due to the high-acquisition cost of sign-off timing evaluation. Therefore, the physical synthesis engine urgently demands efficient sign-off timing evaluation for a more accurate early stage timing consideration.

Fortunately, recent progress in machine learning has permitted fast and precise sign-off timing evaluation. For the first time, Barboza et al. [14] proposed a two-stage framework with carefully selected features (e.g., pin capacitance and net

length) for sign-off net delay prediction in the prerouting stage. PERT traversals [15] are then applied to obtain the global timing metrics, i.e., endpoint slack. For further promotion, He et al. [16] extracted more detailed timing-relative features from a look-ahead RC network. Moreover, to directly evaluate the global timing metrics, Guo et al. [17] developed an end-to-end graph learning model inspired by static timing analysis (STA). Reference [18] further adopts multimodal fusion to consider the effect of complicated timing optimization techniques successfully. Overall, these studies demonstrate the feasibility of predicting sign-off time performance with a deep learning framework and open new avenues for fast and accurate early stage timing optimization.

This article focuses on explicit sign-off timing optimization at the prerouting stage to accelerate the timing closure process. As mentioned before, Steiner tree construction is the routing topology generation step for the most widely used routing frameworks to reduce the problem complexity. In that case, we propose an efficient and effective deep learning-assist optimization framework, *TSteiner*, to optimize the routing topology for better-sign-off timing performance via routing topology optimization. We provide two versions of TSteiner in this work. Extended from the preliminary version, $TSteiner^{Pt}$ [19] with only learning-guided Steiner point adjustment, $TSteiner^{Rec}$ further adapts routing topology reconstruction to accelerate the optimization convergence and a post-optimization topology simplification technique to avoid unnecessary topology constraints.

TSteiner framework exploits the power of graph learning to build a precise sign-off timing evaluator that utilizes the relationship between the sign-off timing metrics and the routing topology. On top of this learning-assist timing evaluator, we build an adaptive optimization framework to iteratively adjust the routing topology, including Steiner point positions and the tree topology, in the early stage for better-sign-off timing performance. Our proposed TSteiner framework is integrated as the prerouting stage of a state-of-the-art (SOTA) open-source design flow (Fig. 1) and evaluated the sign-off timing performance on a set of open-source real-world designs from the previous learning-based timing prediction work [17].

The major contributions of this article are listed as follows.

1) For the first time, we propose a concurrent early stage timing optimization framework, *TSteiner*, based on a customized graph learning framework to guide routing topology optimization.
2) Extended from the preliminary work, $\text{TSteiner}^{\text{Pt}}$ [19], with learning-guided Steiner point adjustment alone, $\text{TSteiner}^{\text{Rec}}$ further includes routing topology reconstruction during the optimization iterations to accelerate convergence and a post-opt topology simplification technique.
3) The proposed TSteiner framework is fully automated with an adaptive stepsize scheme and the auto-convergence scheme, so there is no need to manually set the stepsize and the number of optimization iterations for designs.
4) We conduct comprehensive experiments on real-world designs by integrating our TSteiner into the modern SOTA open-source routing flow. Compared to an averaged 11.2% and 7.1% improvement for sign-off worst and total negative slack (tns) bring by the learning-guided Steiner point adjustment ($TSteiner^{Pt}$), the $TSteiner^{Rec}$ framework with additional reconstruction and simplification techniques can save 25.9% optimization runtime and gain an average of 13.9% and 8.0% on the worst-negative slack (wns) and tns.

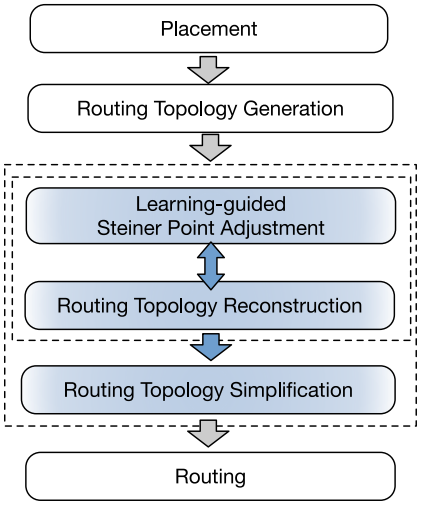

Fig. 1. Physical design flow with TSteiner techniques marked as blue blocks, including learning-guided Steiner point adjustment, routing topology reconstruction, and routing topology simplification.

The remainder of this article is organized as follows. Section II introduces timing closure, the background of Steiner points, and the problem formulation. Section III presents algorithm details of the proposed TSteiner framework. Section IV presents and analyzes the experimental results of TSteiner. Finally, Section V concludes this article.

II. Preliminaries

A. Timing Closure

Ensuring the correct functionality of a circuit design flow heavily relies on meeting the sign-off timing requirements, which poses a critical challenge that needs to be addressed.

To analyze the timing, we extract timing paths from the netlist. Each timing path consists of a startpoint and an endpoint. The startpoint can either be a primary input (PI) or the output pin of a register, while the endpoint can be a primary output (PO) or the input pin of a register.

To achieve timing closure, the delay of each timing path must satisfy specific constraints, such as being less than a single clock cycle. A timing violation occurs when the slack, denoted as $s_e = r_e - a_e$, for a timing path endpoint e becomes negative. Here, r_e represents the required time for e, and a_e denotes the arrival time of e.

To evaluate the timing performance, we use two significant metrics here: 1) wns $w(\cdot)$ and 2) tns $t(\cdot)$. These metrics can be calculated as follows:

$$w(\cdot) = \min_e s_e$$
$$t(\cdot) = \sum_e \{\min\{0, s_e\}\}. \quad (1)$$

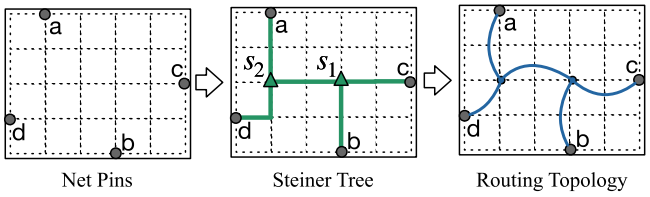

Fig. 2. Relationship between Steiner tree and routing topology. The green lines are the RSMT of the given net pins and the blue wires represent the routing topology following the generated Steiner tree.

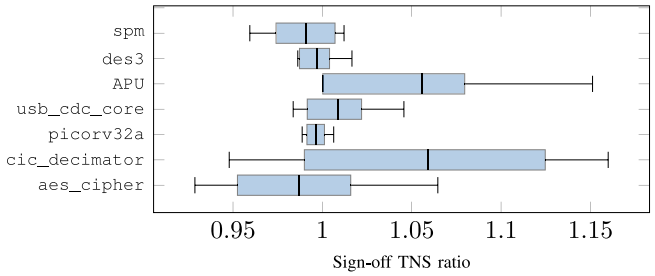

Fig. 3. Distribution of sign-off tns ratio of the updated solution with 10–50 times random Steiner point disturbance (within 5-grid region) to the original one. The larger the ratio deviates from 1, the greater the impact is.

While timing awareness has been incorporated into various stages of the physical design flow, previous academic endeavors have primarily concentrated on enhancing early timing metrics. These metrics typically involve evaluating linear RC timing models and path lengths.

This has motivated us to seek a more explicit approach to directly optimize the sign-off timing performance, specifically the wns and tns metrics.

B. Routing Topology and Steiner Tree

Routing is typically divided into two stages: 1) global routing and 2) detailed routing. In the global routing stage, rough routing is performed on a coarse grid graph to provide initial guidance for the subsequent detailed routing stage. Detailed routing, on the other hand, operates on a fine grid graph to establish connections between real metal wires while minimizing design rule violations, following the guidance provided by global routing.

The generation of a Steiner minimum tree plays a crucial role in the overall routing process. This step decomposes multipin nets into a set of two-pin nets, simplifying the complexity of the routing problem. The rectilinear Steiner minimum tree (RSMT) problem can be defined as follows,

Definition 1 (RSMT): Given a set of net pins P of n points, the RSMT construction is to find a set S of additional Steiner points such that the minimum spanning tree over $\{P \cup S\}$ has minimum cost., e.g., the sum of the rectilinear, or Manhattan, distances on all the connected edges.

As illustrated in Fig. 2, the modern Steiner minimum tree is constructed on the Hannan grid graph [20] using the given net pins, with the aim of minimizing the total wirelength required to connect all the net pins. According to the definition of Steiner trees in [20], we suppose a Steiner tree of n nodes can only have at most (n–1) Steiner points in this work. Recently, deep learning techniques are actively used in the Steiner tree construction to efficiently solve the RSMT problem [21], [22], [23]. Specifically, Liu et al. [22] proposed a novel Steiner tree sequence encoding and then applies reinforcement learning to efficiently generate a available RSMT solution with good wirelength. The generated Steiner tree determines the routing topology, as it provides the positions of Steiner points and the connections between them. Consequently, the placement of Steiner points and the connections in the Steiner tree significantly impact the subsequent routing topology and can impose limitations on the routing performance.

Techniques, such as the Steiner point adjustment, routing topology reconstruction, and routing topology simplification, are introduced in this work to optimize the routing topology. These techniques aim to optimize the positions of the Steiner points and the connections between two pins, with specific objectives, such as improving sign-off timing performance.

C. Timing Optimization via Steiner Point Adjustment

In general, cells in a circuit design have strict geometric constraints, such as nonoverlapping, as they represent physical objects. On the other hand, Steiner points serve as auxiliary points in the post-placement stage and do not have such strict geometric constraints or predefined sizes.

Remarkably, our research has revealed that a random disturbance in the positions of Steiner points can exert a noteworthy influence on the sign-off timing performance, as illustrated in Fig. 3, where the random disturbance is applied to all the Steiner points as same to the setting of our routing topology optimization. However, it is important to note that the impact of such random movement is considerably unstable, resulting in only a minor average performance impact (with a ratio close to 1.0).

These findings highlight the potential of utilizing Steiner point adjustment for timing optimization and emphasize the need for better guidance in determining their movement. Previous studies have focused on Steiner tree-based early stage timing optimization, considering longest path lengths as the objective. However, the overall timing performance includes the net delay and the cell delay, which influence each other in a complex way

Therefore, there is a clear demand for more effective algorithms that incorporate early stage timing-driven Steiner point adjustment to achieve better-timing performance.

D. Graph Neural Networks

Graph neural networks (GNNs) have become an attractive framework for mining graph data [24]. The most popular GNNs follow an iterative message-passage scheme. Given a graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$, a hidden embedding $\boldsymbol{h}_u^{(k)}$ corresponding to each node $u \in \mathcal{V}$ is updated according to information aggregated from u's graph neighborhood $\mathcal{N}(u)$ during each message-passing iteration in a GNN. The kth updating process can be expressed as

$$\begin{aligned} \boldsymbol{h}_u^{(k+1)} &= \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \text{AGGREGATE}^{(k)}\left(\boldsymbol{h}_v^{(k)} \quad \forall v \in \mathcal{N}(u)\right)\right) \\ &= \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right) \end{aligned} \tag{2}$$

where UPDATE and AGGREGATE are differentiable functions and $\boldsymbol{m}_{\mathcal{N}(u)}$ means the "message" aggregated from u's graph neighborhood $\mathcal{N}(u)$. With K iterations, we can define node u's embedding $\boldsymbol{z}_u$ as

$$\boldsymbol{z}_u = \boldsymbol{h}_u^K \quad \forall u \in \mathcal{V}. \tag{3}$$

GNNs based on the message-passing framework have shown superior efficacy in learning graph structures. Recently, GNNs have gained popularity in the electronic design automation (EDA) community [25] because the circuits can be naturally represented as graphs.

E. Problem Formulation

Definition 2 (Timing-Driven Routing Topology Optimization): Given an initial Steiner tree set $S_T = \{T^1, T^2, \ldots, T^n\}$, $T^i = (V_c^i, V_s^i, E^i)$, where V_c^i is the set of cell nodes, V_s^i is the set of Steiner nodes and E^i means the edges connecting V_c^i and V_s^i of the ith Steiner tree, our task is to refine the position (X_s, Y_s) of $V_s = \{V_s^i, 1 \le i \le n\}$ and optimize the connections in Steiner trees in the prerouting stage to obtain better-*sign-off* timing performance.

III. Algorithm

In this section, we will illustrate our solution, TSteiner, for timing-driven routing topology optimization. The overall flow is first mentioned to guide the following details on the optimization gradient generation and the routing topology optimization process.

A. TSteiner

Before diving into the details of the TSteiner framework, we first illustrate the overall flow in Fig. 4. The TSteiner can be split into three parts. First, given the initial (updated) Steiner trees, a differentiable sign-off timing prediction engine generates the corresponding topology optimization gradients. Then, TSteiner applies the topology optimization gradients to guide Steiner point refinement with routing topology reconstruction. Note that the optimization gradient generation and routing topology optimization are executed iteratively and alternately until the convergence is obtained. We applied the same sign-off timing prediction engine to evaluate the convergence. After all the optimization iterations, a routing topology simplification strategy is proposed to remove the redundant Steiner points to leave more optimization space for the subsequent stages.

B. Optimization Gradient Generation

The foundation of our gradient generation framework is building the relationship between sign-off timing performance and Steiner point positions. Specifically, we utilize the trending GNN to construct an accurate sign-off timing evaluation model with Steiner point position information as input. The gradients for each Steiner point can then be generated automatically with

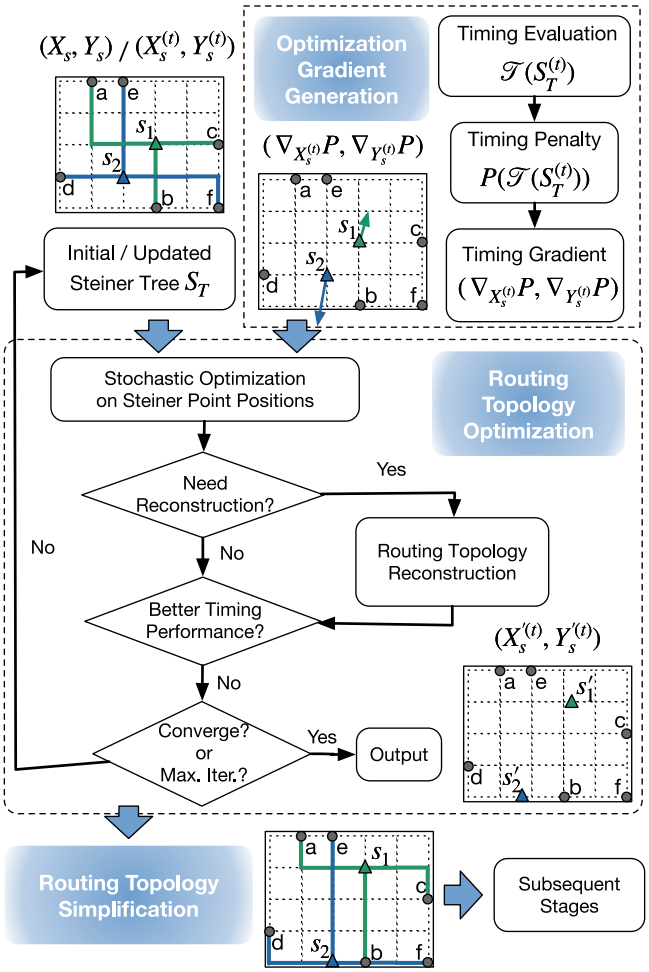

Fig. 4. Overall flow of TSteiner framework, including the optimization gradient generation, the routing topology optimization, and the routing topology simplification.

the model's backward propagation procedure. Specifically, the timing evaluation problem is formulated as follows,

Definition 3 (Sign-Off Timing Evaluation): Given a Steiner tree solution S_T, timing evaluation is to find an estimator $\mathcal{T}$ to evaluate the sign-off timing metrics $\mathcal{T}(S_T)$, i.e., arrival time at each pin.

Unfortunately, all previous ML-driven prerouting timing evaluators [14], [16], [17] did not consider Steiner points. In this article, we design a customized model to integrate Steiner trees into the SOTA prerouting timing prediction framework [17].

Timing Evaluation Model Initialization: Unlike [17], which builds the graph solely from netlist connections, we construct an additional graph from the Steiner trees. The two graphs are denoted as the netlist graph and Steiner graph, respectively, as shown in Fig. 5. Precisely, the netlist graph reflects connections between pins, which is heterogeneous with two edge types: 1) cell edge and 2) net edge. Each cell edge links one input pin of a cell with its output pin, while each net edge connects a net's drive pin to one of its sink pins. On the other hand, the Steiner graph is node-heterogeneous with two types of nodes, Steiner nodes and pin nodes, to distinguish Steiner

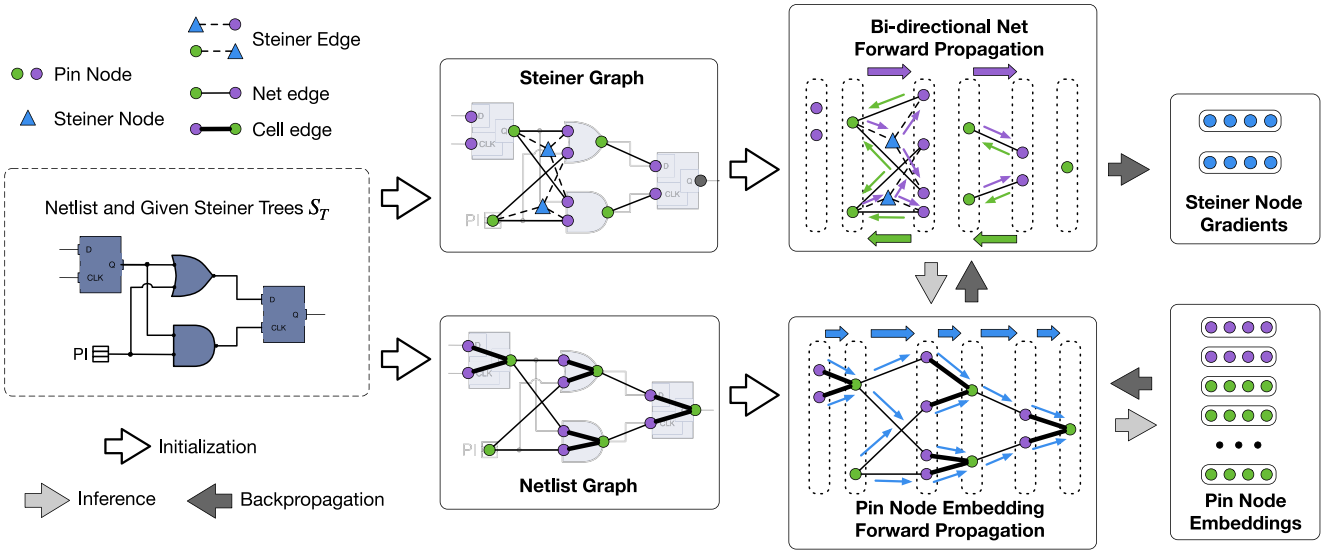

Fig. 5. Overview of our proposed gradient generation flow driven by a customized timing evaluation model. The purple and green dots represent pin nodes, while the blue triangle represents Steiner nodes. The forward inference starts from bidirectional propagation on the Steiner graph, followed by the propagation on the netlist graph. Steiner graph broadcast and Steiner graph reduce are included in the bi-directional net propagation. Further, the blue lines depict the pin embedding propagation on the netlist graph. The forward propagation is applied to obtain the pin arrival time evaluation, while the backward propagation is used to collect the Steiner node gradients.

points from pins. As for edges, they are also heterogeneous, with Steiner edges and net edges. Note that we level both the Steiner graph and netlist graph according to the source-sink relationship to facilitate the message-passing.

Timing Evaluation Model Inference: We propose a two-stage message-passing scheme with the above two graphs to fuse information from Steiner trees and netlists. The scheme begins by aggregating information from the Steiner graph, which can be divided into two steps: 1) broadcast and 2) reduce. In the broadcast step, information flows from each net's drive pin to sink pins along the Steiner edges, as denoted by the purple lines in Fig. 5. During broadcasting, information on the drive pin and associated Steiner points are aggregated to each sink pin and fused to update the features of the sink pins. Then, in the reduce step, the updated sink pins' features flow backward to the drive pin along the net edges to renew the drive pin's feature, as denoted by the green lines in Fig. 5. The steps above are repeated until the Steiner tree information is fully fused. In practice, we set three iterations.

After the message-passing on the Steiner graph, the Steiner point position information, which we are interested in, has been aggregated into the related pins' features. The updated pin features are then propagated on the netlist graph in topological order (denoted by the blue edges in Fig. 5) to generate pin node embeddings, which can be used for predicting pin-wise arrival time. Due to the page limitation, readers can refer to the propagation model in [17] for details about input features and the message-passing on the netlist graph.

Having the well-trained sign-off timing evaluator $\mathcal{T}$, the sign-off timing metrics (wns and tns) can be evaluated based on the predicted endpoint arrival time, as introduced in (1). Then, the timing penalty can be calculated using the following equation:

$$P(\mathcal{T}(S_T)) = \lambda_w w(\mathcal{T}(S_T)) + \lambda_t t(\mathcal{T}(S_T)) \tag{4}$$

where $w(\mathcal{T}(S_T))$ and $t(\mathcal{T}(S_T))$ denote the evaluated wns and tns. λ_w and λ_t are the weights for wns and tns, respectively.

Timing Penalty Smoothing: As the formal formulations of wns and tns contain minimum or maximum operation, directly applying the above penalty for backward propagation leads to a cut-off in some timing paths. However, timing optimization should consider all of the pins and paths globally. To overcome the above drawback, we smooth the minimum and maximum operations in the computation of wns and tns. To be more specific, we replace the maximum operation with the Log-Sum-Exp function LSE as follows:

$$\mathrm{LSE}(x_1, x_2, \ldots, x_n) = \gamma \log\left(\sum_{i=1}^{n} \exp \frac{x_i}{\gamma}\right) \tag{5}$$

where γ is the parameter for the degree of smoothing, and a larger γ indicates smoother results and lower accuracy. Similarly, the minimum operation can be treated as the maximum operation of the inverse values. Finally, the smoothed penalty function $P_\gamma(\mathcal{T}(S_T))$ can be expressed as

$$P_\gamma(\mathcal{T}(S_T)) = \lambda_w w_\gamma(\mathcal{T}(S_T)) + \lambda_t t_\gamma(\mathcal{T}(S_T)) \tag{6}$$

where $w_\gamma(\cdot)$ and $t_\gamma(\cdot)$ denote the smoothed version of $w(\cdot)$ and $t(\cdot)$, respectively. With the smoothed penalty, the timing optimization gradients w.r.t. Steiner points positions $(\nabla_{X_s} P, \nabla_{Y_s} P)$ can be computed automatically via backward propagation, which is then used in our concurrent Steiner point refinement flow as described in Algorithm 1. Since we only set the feature of Steiner nodes' positions as "gradient required," no gradient will be calculated for other features, e.g., pin node positions.

C. Routing Topology Optimization

Having sign-off timing gradients $\nabla_{X_s^{(t)}} P$ w.r.t. the Steiner points' X coordinates $X_s^{(t)}$ in the tth optimization iteration, a

concurrent Steiner point refinement algorithm will be applied to update $X_s^{(t)}$ with the stochastic optimization algorithm stochastic optimizer (SO) described as follows:

$$\begin{aligned} m_x^{(t)} &= (1-\beta_1)\cdot \nabla_{X_s^{(t)}} P \\ v_x^{(t)} &= (1-\beta_2)\cdot \left(\nabla_{X_s^{(t)}} P \odot \nabla_{X_s^{(t)}} P\right) \\ X_s'^{(t)} &= X_s^{(t)} - \theta \cdot \frac{m_x^{(t)}}{\sqrt{v_x^{(t)}} + \epsilon} \end{aligned} \tag{7}$$

where θ is the stepsize to optimize Steiner point positions; and β_1, β_2, and ϵ are the hyperparameters. The update process for $Y_s^{(t)}$ is similar to (7) and shares the same hyperparameters.

To boost the performance of our method on designs with various scales, we propose an adaptive stepsize scheme that automatically generates customized stepsize θ fitting every design. Given the initial Steiner trees set S_T, our adaptive stepsize scheme *Adaptive_Theta* can be divided into three steps.

1) Obtain the initial timing gradient $(\nabla_{X_s} P, \nabla_{Y_s} P)$ w.r.t. the given Steiner point positions (X_s, Y_s).
2) Apply a small move

$$\begin{aligned} X_s' &= X_s + \alpha \nabla_{X_s} P \\ Y_s' &= Y_s + \alpha \nabla_{Y_s} P \end{aligned} \tag{8}$$

where α is a hyperparameter to control the scale of θ.
3) Obtain the updated timing gradient $(\nabla_{X_s'} P, \nabla_{Y_s'} P)$.

The adaptive stepsize is then calculated as follows:

$$\theta = \frac{\left|(X_s, Y_s) - (X_s', Y_s')\right|_2}{\left|(\nabla_{X_s} P, \nabla_{Y_s} P) - (\nabla_{X_s'} P, \nabla_{Y_s'} P)\right|_2}. \tag{9}$$

With the adaptive stepsize scheme, our concurrent Steiner point refinement framework is described in Algorithm 1. The algorithm begins by initializing variables and setting up the optimizer (lines 1–5). Followed is the main part of the algorithm that conducts refinement recursively until convergence. In the tth iteration, the Steiner point positions are updated using (7) to obtain the temporary Steiner trees $S_T'^{(t)}$ (line 7). $S_T'^{(t)}$ will be stored if it achieves better-(evaluated) sign-off timing performance (wns or tns). Elsewise, $S_T^{(t)}$ from the previous iteration will be restored (line 13). The optimization procedure stops when the sign-off timing metrics are fully optimized (line 19) or it reaches the maximum optimization iterations N (line 16).

Algorithm 1 Concurrent Steiner Point Refinement. The Adaptive Step Size Scheme (*Adaptive_Theta*) and SO Are Applied to Optimize the Steiner Point Positions Using (7)

Input: S_T: initial Steiner trees; $\mathcal{T}$: pretrained timing prediction model; N: maximum optimization iterations; μ: converge ratio.

1: $init\_wns \leftarrow w(\mathcal{T}(S_T))$; $best\_wns \leftarrow w(\mathcal{T}(S_T))$;
2: $init\_tns \leftarrow t(\mathcal{T}(S_T))$; $best\_tns \leftarrow t(\mathcal{T}(S_T))$;
3: $\theta \leftarrow Adaptive\_Theta(S_T)$;
4: t $\leftarrow 0$; $S_T^{(0)} \leftarrow S_T$; $X_s^{(0)} \leftarrow X_s$; $Y_s^{(0)} \leftarrow Y_s$;
5: Initialize the optimizer *SO* with θ;
6: **repeat**
7: $\quad S_T'^{(t)} \leftarrow SO(S_T^{(t)}, (\nabla_{X_s^{(t)}} P, \nabla_{Y_s^{(t)}} P))$;
8: $\quad wns \leftarrow w(\mathcal{T}(S_T'^{(t)}))$; $tns \leftarrow t(\mathcal{T}(S_T'^{(t)}))$;
9: $\quad$ **if** $wns > best\_wns$ or $tns > best\_tns$ **then**
10: $\quad\quad best\_wns \leftarrow wns$; $best\_tns \leftarrow tns$;
11: $\quad\quad S_T^{(t+1)} \leftarrow S_T'^{(t)}$;
12: $\quad$ **else**
13: $\quad\quad S_T^{(t+1)} \leftarrow S_T^{(t)}$;
14: $\quad$ **end if**
15: $\quad$ t $\leftarrow t+1$;
16: $\quad$ **if** $t \geq N$ **then**
17: $\quad\quad$ break;
18: $\quad$ **end if**
19: **until** $\frac{init\_wns - best\_wns}{init\_wns} > \mu$ or $\frac{init\_tns - best\_tns}{init\_tns} > \mu$
20: **return** $S_T^{(t)}$ (Resulting Steiner Trees)

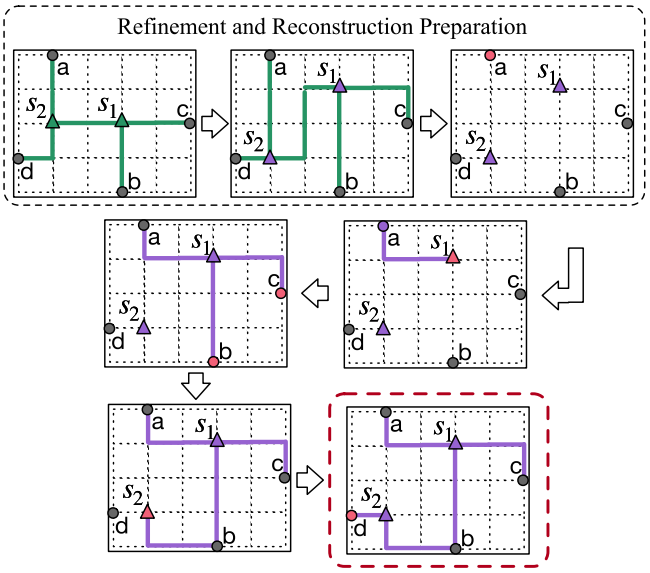

Fig. 6. Routing topology reconstruction with minimum spanning tree generation. The original nodes and tree segments are colored green, while the optimized ones are purple. The current root during the reconstruction process is colored in red and the final routing topology is marked with a red dashed circle.

D. Routing Topology Reconstruction

The Steiner point adjustment aims to optimize the Steiner point positions based on learned information and previous design data. While this adjustment can improve timing performance, it is possible for the moved Steiner points to deviate significantly from their original positions. This deviation may result in wirelengths that are difficult to control or manage effectively. In [19], we tried to set the same move boundary for each Steiner point.

Having the optimized Steiner point positions, we propose routing topology reconstruction to save the unnecessary detours as shown in Fig. 6. As described in Definition 1, the Steiner minimum tree is to obtain the minimum spanning tree to connect the point set, including both net pins and Steiner points, under the minimum cost. Therefore, we adopt Prim's algorithm in TSteiner to rebuild the routing topology with

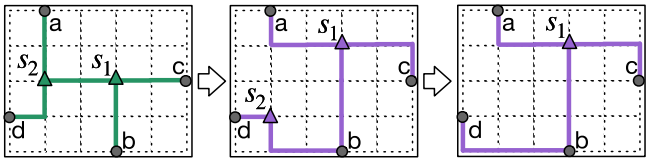

Fig. 7. Routing topology simplification by removing redundant Steiner points to alleviate unnecessary topology limitations. Here, s_2 is removed since it only connects two nodes in the given routing topology.

the source pin as the start root and control the signal paths with optimized point positions. Note that the popular Steiner tree construction algorithms are not applied since they will generate more Steiner points and make the process more and more complicated. The updated routing topology is reflected on the Steiner graph in the previously introduced gradient generation flow.

Our proposed optimization flow uses the routing topology reconstruction in specific cycles to optimize our iterations, as shown in Fig. 4. Further, the routing topology applied in the subsequent routing stages also takes advantage of the routing topology reconstruction.

E. Routing Topology Simplification

With the cooperation of routing topology optimization and reconstruction, our TSteiner framework will output results with optimal performance after convergence. It is worth noting that our graph learning model only capture the position information from Steiner point and calculate the distance based on the Steiner graph connection information, which makes our model treats the Steiner graph with a similar Manhattan distance as similar timing performance. However, if the Steiner point only connect to 2 pins, the subsequent routing process will obey this unnecessary limitation and bring unnecessary detours. In Fig. 6, it can be noticed that s_2 no longer plays the role of a Steiner point but only connects two nodes after reconstruction. This causes s_2 to be redundant and will limit the routing performance. Therefore, we propose the routing topology simplification as a post-processing technique.

As illustrated in Fig. 7, TSteiner will remove the redundant Steiner points, like s_2 here, and directly build a routing topology edge between the two nodes they are connected to. The idea of our routing topology simplification is direct but very important for routing topology optimization with the routing tree reconstruction technique to alleviate unnecessary topology constraints. We also conducted an ablation study to show the importance of routing topology simplification for an optimization flow with reconstruction.

IV. Experiment Results

A. Experiment Setting

Experiment Environment: We develop our concurrent routing topology refinement framework with DGL [26], Pytorch [27], and C++. The customized timing evaluation model is trained and tested on a Linux machine with 16 Intel Xeon Gold 6226R cores (2.90 GHz), 1 GeForce RTX 3090 Ti graphics card, and 24 GB of main memory. The input features of Steiner graph and netlist graph described in Fig. 5 are extracted after the initial routing topology generation as shown in Fig. 1. We obtain the pin arrival time (ground truth labels) from Cadence Innovus's reports[1] after TritonRoute [28] completes the detailed routing. Note that the cell delay is around 50×–200× over the neighbor net delay under Skywater 130-nm process design kit (PDK). Cooperating with the bi-directional net forward propagation, our TSteiner framework would reduce the cell delay by optimizing the input transition and output load with updated routing topologies. Further, we verify the efficacy of the TSteiner$^{\text{Rec}}$ framework over the previous early stage timing optimization methods by integrating before the modern SOTA routing flow to guide the early stage timing-driven routing topology refinement. Note that the integrated routing flow is running on the same machine.

Experiment Flow: The initial routing topology solution is generated by a widely used Steiner minimum tree construction algorithm, FLUTE [29], followed by the edge shifting technique [30] for congestion alleviation. Then, the proposed routing topology refinement framework is applied to optimize the Steiner point positions and the routing topology. Having the optimized routing topology, the modern SOTA open-source routing flow, CUGR [31] as the global router and TritonRoute [28] as the detailed router, is followed to generate the final routing solution. Note that the maximum DR iterations in TritonRoute here is fixed to 20 iterations. Both of them are running with eight threads. Dr.CU [32] is not used here since it cannot produce available detailed routing solutions with too many design rule violations on our generated real-world designs. In addition, we apply the industry-leading commercial tool, Cadence Innovus, to complete the placement stage and obtain sign-off timing analysis reports.[2] All the design flows are solution-deterministic.

Experiment Dataset: We prepare ten real-world open-source designs for evaluation. All the designs are obtained from OpenCores [33] and synthesized with the open-source 130-nm PDK [34] as the same configuration to [17]. Note that since some open-source designs in [17] can not be routed successfully by the open-source routing flow, we use the ten success designs as our evaluation benchmark here.

B. Sign-Off Timing Prediction Performance

The circuit benchmarks are split into training and testing sets with the benchmark statistics listed in Table I. The training and testing sets are determined by design scale in order to make balance. Note that the separate strategy is only for the graph learning model training and all the designs are applied to evaluate our proposed routing topology refinement flow. The timing evaluation model is trained on the training set with a learning rate of $5e-4$.

Our sign-off timing evaluation model is employed to predict the arrival time on each pin. Furthermore, we extract the predicted arrival time on the endpoints to compute the required timing metrics (wns and tns). Specifically, the R^2 score is an

[1] `report_cell_instance_timing [get_cells -hier -hsc /]`.

[2] Use the Tcl command "`timeDesign -postRoute`" with OCV mode.

TABLE I
BENCHMARK STATISTICS. THE TEN BENCHMARKS ARE RANDOMLY SPLIT INTO THE UPPER SIX BENCHMARKS FOR TRAINING AND THE LOWER FOUR FOR TESTING. "# ENDPOINTS" REPRESENTS THE NUMBER OF TIMING PATHS AND "CLOCK" MEANS THE CLOCK CYCLE (NS)

Benchmark	# Nodes		# Edges		# End	Clock(ns)
	Cell	Steiner	Net	Cell		
chacha	15700	5398	44468	41204	1972	26.01
cic_decimator	781	196	2112	1982	130	10.00
APU	2897	1154	8373	7918	427	17.00
des	14652	5487	43065	40432	2048	20.00
jpeg_encoder	55264	15982	170520	161743	4420	15.78
spm	238	63	645	516	129	3.00
aes_cipher	11532	7323	37085	35825	659	10.09
picorv32a	13622	4542	41030	38191	1879	24.73
usb_cdc_core	1642	625	4632	3999	626	9.00
des3	47410	20004	136257	125093	8872	7.59
Total Train	89532	28280	269183	253795	9126	
Total Test	74206	32494	219004	203108	12036	

important metric to evaluate the coefficient of determination in statistics (the closer to 1, the better), and the formal formulation of R^2 score with ground truth $\{g_1, g_2, \ldots, g_n\}$ and the predicted value $\{y_1, y_2, \ldots, y_n\}$ is

$$\bar{g} = \frac{1}{n}\sum_{i=1}^{n} g_i, R^2 = 1 - \frac{\sum_i (g_i - y_i)^2}{\sum_i (g_i - \bar{g})^2}. \quad (10)$$

Our proposed timing evaluation model can accurately predict sign-off timing metrics. Particularly, the R^2 scores of the arrival time prediction on all pins are 0.9959 in the training cases and 0.9280 in the testing cases, indicating that our timing evaluation framework well models the pin arrival time. Since the calculations of wns and tns are based on the endpoints' arrival time, we also list the R^2 scores for endpoint-only prediction. The results show that the proposed evaluation model consistently performs well on the endpoints.

C. Sign-Off Timing Optimization Performance

In this article, TSteiner$^{\text{Rec}}$ includes the full three stages, optimization gradient generation, routing topology optimization with reconstruction and simplification, while the TSteiner$^{\text{Pt}}$ [19] only concludes the optimization gradient generation and the routing topology optimization without reconstruction. Here, TSteiner$^{\text{Rec}}$ utilizes the same evaluation model to TSteiner$^{\text{Pt}}$ without further training. We also set SALT [9] as targeting at the minimal path length to compare our TSteiner$^{\text{Rec}}$ to the traditional early stage timing consideration in routing topology generation.

Within both concurrent routing topology optimization flows, TSteiner$^{\text{Pt}}$ [19] and TSteiner$^{\text{Rec}}$, we initialize λ_w as -200.0 and λ_t as -2.0. To smooth the penalty function described in Section III-B, we set γ as 10.0. α in adaptive stepsize generation is set to 5.0, and the converge rate μ in the concurrent Steiner point refinement stage is set to 0.1. Starting from the fifth iteration, we increase λ_w and λ_t by 1% in each following iteration since the Steiner points may have already been optimized enough with five iterations. The routing topology reconstruction is called by every five optimization iterations in TSteiner$^{\text{Rec}}$ to save the reconstruction runtime. Lastly, For each design, we constrain the largest moving distance according to the width and length of the global routing grid graph. All the reported metrics are obtained from Cadence Innovus.

With the above setting, Table II shows the detailed sign-off timing metrics comparison on the wns, tns, and the number of violated timing paths(#vios). As illustrated in Table II, directly considering path length as the early timing metric to optimization, as introduced in SALT [9], majorly optimize the wns but is hard to optimize the global tns. jpeg_encoder's routing guides generated by the CUGR global router with SALT [9] conflicts the track assignment requirement in TritonRoute, resulting in unavailable routing solution. Therefore, we remove jpeg_encoder from the column "w/ SALT." On the other hand, concluded from Table II, with an early stage timing optimization strategy, either TSteiner$^{\text{Pt}}$ [19] or our TSteiner$^{\text{Rec}}$, before the modern routing flow indeed helps to improve the sign-off timing performance on both the local wns and the global tns.

In detail, on the one hand, TSteiner$^{\text{Pt}}$ can obtain an averaged 11.2%, 7.1%, and 3.3% improvement on the wns, tns, and the number of violated timing paths, respectively, with 32% runtime overhead. On the other hand, the TSteiner$^{\text{Rec}}$ with additional routing topology reconstruction and simplification techniques can further extend the optimization improvements on wns and tns to 13.9% and 8% with only 14.3% runtime overhead. At the same time, since the absolute values of the worst and tns largely depend on the set clock frequency, we focus on the comparison of wns here, because it can represent the maximum frequency allowed for each design. We can see from Table II, TSteiner$^{\text{Rec}}$ can optimize the wns up to 58.1% while TSteiner$^{\text{Pt}}$ [19] can only bring up to 45.8% benefit. Also, we separate the designs in Table II and give the averaged performance for the training and testing designs, respectively. The results show that whether on the model training set or the test set, the optimization performance is good enough.

Further, we statistic the slack distributions in different stages to show the necessity of adapting sign-off timing performance to guide the early stage optimization. We pick des3 and jpeg_encoder as the examples in Fig. 8(a) and (b), respectively. To begin with, the pre-CTS slack distribution has a huge gap to the post-Route one as shown in the purple and blue distributions. Then, by comparing the post-Route slack distribution with or without our proposed TSteiner$^{\text{Rec}}$ optimization stage, we can find that our learning-guided routing topology refinement can not only optimize the wns, but also optimize the whole distribution closer to 0.0.

The above performance analysis demonstrates that our TSteiner$^{\text{Rec}}$ with full routing topology optimization flow is better than simply refining the Steiner point positions in [19]. Also, the early stage timing optimization is crucial for the sign-off timing performance.

D. Runtime Analysis

To better understand the overhead of running time and whether the early stage optimization can benefit the subsequent running process, we list the runtime breakdown in Table III.

TABLE II
SIGN-OFF TIMING METRICS COMPARISON BY INTEGRATING DIFFERENT EARLY STAGE TIMING OPTIMIZATION STRATEGIES INTO THE MODERN OPEN-SOURCE ROUTING FLOW. HERE, WNS, TNS, AND #VIOS REPRESENT THE WORST-NEGATIVE SLACK, THE TOTAL NEGATIVE SLACK, AND THE NUMBER OF TIMING VIOLATED PATHS

Benchmark	CUGR [31] + TritonRoute [28]			w/ SALT [9]			w/ TSteiner^{Pt} [19]			w/ TSteiner^{Rec}		
	wns	tns	# vios	wns	tns	# vios	wns	tns	# vios	wns	tns	# vios
chacha	-48.538	-26259.1	1378	-47.364	-26305.4	1410	-46.68	-25375.7	1372	-42.3	-23774.6	1368
cic_decimator	-2.834	-169.981	72	-2.875	-170.809	72	-2.724	-161.436	72	-2.581	-151.783	72
APU	-2.265	-33.713	25	-2.454	-36.445	25	-2.221	-33.598	25	-2.357	-33.209	25
des	-7.352	-405.427	341	-6.352	-1068.4	359	-3.987	-227.331	285	-3.082	-255.496	302
jpeg_encoder	-74.342	-64909.2	1967		—		-70.629	-60789.1	2007	-71.783	-60359.9	1946
spm	-0.817	-65.866	126	-0.786	-64.266	126	-0.782	-63.846	126	-0.766	-62.802	126
aes_cipher	-11.246	-1516.9	512	-8.596	-1475.2	512	-8.38	-1434.2	504	-8.202	-1435.9	510
picorv32a	-17.762	-441.607	67	-17.899	-443.315	58	-17.686	-434.443	56	-17.604	-437.726	64
usb_cdc_core	-5.914	-1365.2	347	-6.031	-1392	348	-5.823	-1343.1	346	-5.785	-1340.8	346
des3	-7.048	-1890	1512	-6.24	-1881.5	1507	-5.668	-1879.6	1509	-5.384	-1867.4	1503
All Aver.	1.000	1.000	1.000	0.953	1.188	0.993	0.888	0.929	**0.967**	**0.861**	**0.920**	0.981
All BEST	1.000	1.000	1.000	0.764	0.973	0.866	0.542	**0.561**	**0.836**	**0.419**	0.630	0.886
Train Aver.	1.000	1.000	1.000	0.984	1.431	1.019	0.892	0.897	**0.975**	**0.858**	**0.883**	0.978
Test Aver.	1.000	1.000	1.000	0.919	0.998	0.966	0.882	0.977	**0.954**	**0.866**	**0.977**	0.986

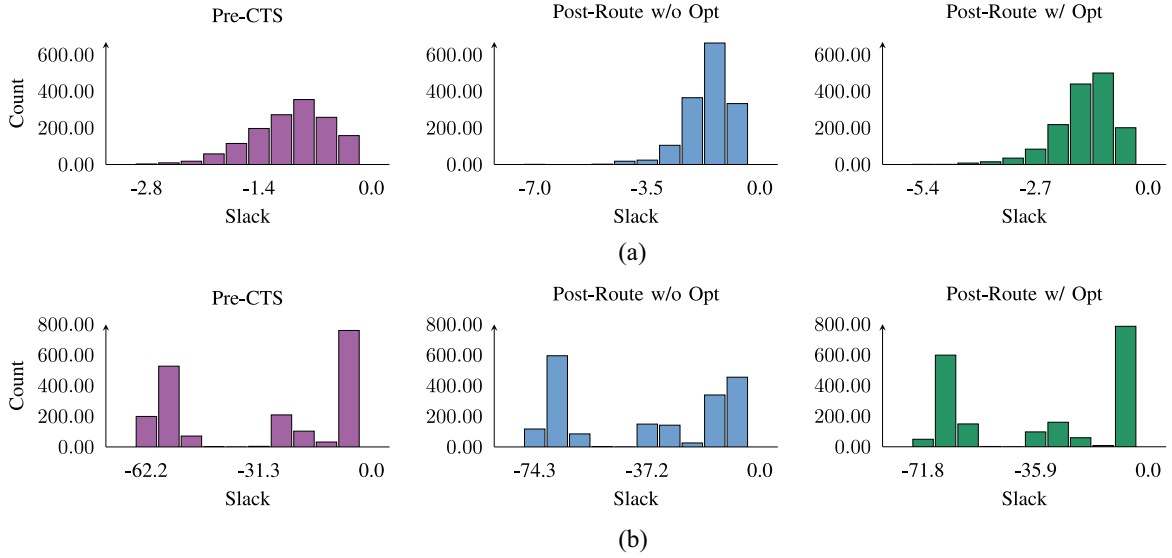

Fig. 8. Slack Distribution Comparison Among Different Settings, Including Pre-CTS, Post-Route Without Our TSteiner^{Rec} Optimization Stage, and Post-Route With TSteiner^{Rec}. Here, We Select Two Different Designs des3 and jpeg_encoder and Only Statistic the Negative Slack. "Count" Means the Number of Endpoints Within Each Slack Range. (a) des3. (b) jpeg_encoder.

In Table III, opt represents the early stage timing optimization part, while global and detailed are separate global and detailed routing flows. As illustrated in Table III, the utilization of early stage optimization can save the running time of the detailed routing stage by 6.6% and 8.6% on average. It points to the fact that early stage timing optimization can further reduce design rule violations since it reduces routing detours and optimizes the use of routing resources. The running time of the global routing stage increases slightly since the update of Steiner point and routing topology with optimization results is included. It is also worth noting that the running time of routing topology optimization can be largely saved by 25.9% by adding the routing topology reconstruction technique. The reason for the acceleration is that the routing topology reconstruction helps the convergence of the optimization iterations and the routing topology simplification technique works to remove unnecessary routing topology constraints. We further illustrate the runtime trend with the increasing cell numbers in Fig. 9. Compared to TSteiner^{Pt}, TSteiner^{Rec} is more robust

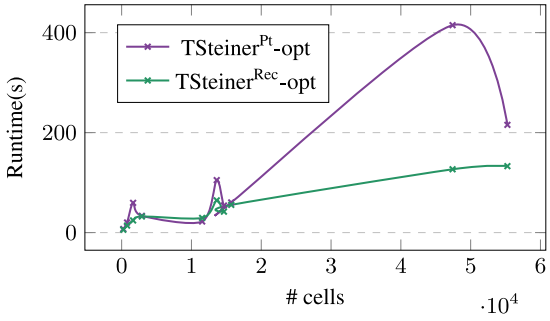

Fig. 9. Optimization runtime scalability comparison over the increasing number of cells.

TABLE III
RUNTIME BREAKDOWN OF DIFFERENT EXPERIMENTAL FRAMEWORKS, PURE MODERN ROUTING FLOW, THE MODERN ROUTING FLOW WITH TSTEINER^{Pt} [19], AND THE MODERN ROUTING FLOW WITH TSTEINER^{Rec}. THE RUNTIME OF THE ROUTING TOPOLOGY OPTIMIZATION FRAMEWORK (OPT), GLOBAL ROUTING (GLOBAL), DETAILED ROUTING (DETAILED), AND TOTAL RUNTIME (TOTAL) ARE LISTED BELOW

Benchmark	CUGR + TritonRoute				w/ TSteiner^{Pt} [19]				w/ TSteiner^{Rec}			
	opt	global	detailed	total	opt	global	detailed	total	opt	global	detailed	total
chacha	-	19.795	460.328	480.123	60.331	21.642	422.929	504.902	55.475	22.316	436.812	514.603
cic_decimator	-	0.592	133.202	133.794	20.174	0.529	123.118	143.821	14.084	0.549	108.209	122.842
APU	-	2.299	92.8207	95.1197	33.563	2.479	84.9509	120.9929	32.274	2.415	88.294	122.983
des	-	10.725	233.176	243.901	54.075	11.423	215.101	280.599	41.902	10.87	216.017	268.789
jpeg_encoder	-	61.433	832.052	893.485	215.533	62.916	819.977	1098.426	132.912	70.955	826.348	1030.215
spm	-	0.307	11.5409	11.8479	7.195	0.279	10.0188	17.4928	5.679	0.302	11.165	17.146
aes_cipher	-	16.781	523.066	539.847	22.222	17.83	488.46	528.512	29.333	17.518	446.902	493.753
picorv32a	-	12.622	393.664	406.286	104.957	13.109	354.906	472.972	64.453	13.276	354.262	431.991
usb_cdc_core	-	1.247	53.413	54.66	59.515	1.156	55.1131	115.7841	24.302	1.117	47.264	72.683
des3	-	37.863	520.151	558.014	414.908	40.841	485.725	941.474	126.668	37.708	470.508	634.884
Average	-	1.000	1.000	1.000	1.000	1.017	0.934	1.320	0.741	1.024	0.914	1.143

TABLE IV
DETAILED ROUTING SOLUTION PERFORMANCE COMPARISON, INCLUDING ROUTED WIRELENGTH(WL), THE NUMBER OF METAL VIAS(#VIAS), AND THE NUMBER OF DESIGN RULE VIOLATIONS(#DRV). NOTE THAT THE WIRELENGTH METRIC IS IN μM

Benchmark	CUGR + TritonRoute			w/ TSteiner^{Pt} [19]			w/ TSteiner^{Rec}		
	wl (e+03)	#vias	#drv	wl (e+03)	#vias	#drv	wl (e+03)	#vias	#drv
chacha	1257.427	126600	2	1258.011	126898	2	1257.956	126620	2
cic_decimator	16.466	5586	3	16.413	5593	3	16.456	5589	3
APU	101.179	23031	3	101.454	23101	3	101.198	23038	3
des	682.828	115698	5	682.788	115599	5	683.854	115606	4
jpeg_encoder	2969.654	439126	1	2973.304	439561	1	2971.789	439146	1
spm	4.394	1553	2	4.399	1544	2	4.379	1551	2
aes_cipher	984.971	109574	5	984.527	109443	3	985.146	109521	4
picorv32a	727.216	109293	38	727.472	109311	37	727.725	109372	35
usb_cdc_core	49.351	12396	0	49.117	12407	0	49.298	12354	0
des3	2680.848	372583	48	2684.367	372768	49	2681.884	372415	50
Average	1.0000	1.0000	1.000	0.9999	1.0001	0.955	0.9999	0.9995	0.951

in terms of running time with the increasing number of cells.

E. Routing Solution Quality

Besides the timing performance, we also care about the routing solution quality on wirelength, the number of metal vias, and the number of design rule violations, even though these metrics are not directly included in our proposed routing topology framework. The design rule checking results are also reported by Cadence Innovus. As listed in Table IV, our proposed early stage routing topology optimization framework can obtain comparable routing solution quality. Further, the listed results demonstrate that the routing topology reconstruction and simplification techniques can better control the routing quality compared to TSteiner^{Pt} [19].

F. Necessity of Routing Topology Simplification

As mentioned before, routing topology simplification is really important to the optimization flow with routing topology reconstruction. Note that our proposed simplification technique is not used in TSteiner^{Pt} [19] since it does not change the routing topology structure but just updates the Steiner point positions. As illustrated in Fig. 10, The sign-off timing performance is out of control by removing the simplification but just keeping the routing reconstruction due to the impact of redundant Steiner points. The simplification here is only called for once for the post-processing but plays a crucial role in controlling the overall performance.

G. Steiner Point Adjustment

Since the graph learning model in TSteiner^{Rec} is worked as a closed box to provide optimization guidance, we statistic the Steiner point adjustment distance and plot the distribution in Fig. 11. As shown in Fig. 11, most Steiner points are moved within a 5-grid distance. Specifically, several designs, APU, chacha, cic_decimator, usb_cdc_core, and des, have more than 50% Steiner points not moved. This may inspired us that recognizing the most critical 20% Steiner points to optimize is important for early stage optimization.

V. CONCLUSION

This article highlights the significant impact of routing topology on sign-off timing performance. We introduce

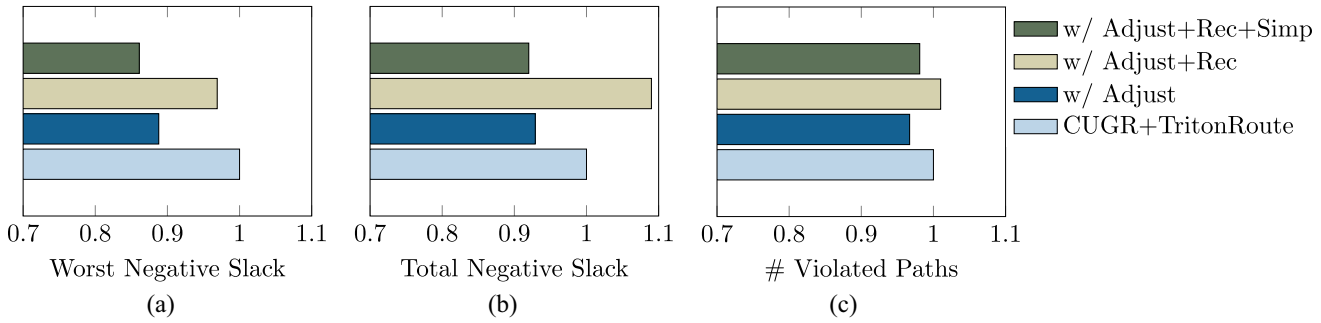

Fig. 10. Ablation study to show the necessity of routing topology simplification. Three comparisons on the averaged wns, tns, and the number of violated timing paths are elaborated. We set the original performance without any early stage optimization as 1.0 for reference and list the sign-off timing performance ratio. Note that "w/ Adjust" means TSteiner^{Pt} while "w/ Adjust+Rec+Simp" is TSteiner^{Rec}. (a) wns. (b) tns. (c) # Violated paths.

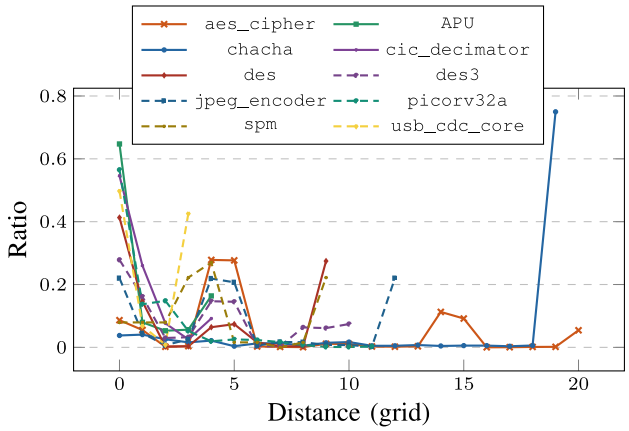

Fig. 11. Steiner point adjustment distance distributions.

TSteiner, a concurrent routing topology optimization framework designed to improve sign-off timing performance during the early physical synthesis stage. We utilize a customized graph learning model to bridge the gap between the early physical synthesis solution and the sign-off stage. This model guides the concurrent adjustment of Steiner points using stochastic optimization. We also introduce two connection-related techniques, routing topology reconstruction, and simplification, to accelerate the convergence and prevent unnecessary topology constraints in subsequent stages. As shown in the comprehensive experimental results, adjusting Steiner points alone (TSteiner^{Pt}) improves wns by 11.2% and tns by 7.1%, with an acceptable runtime overhead. Additionally, incorporating routing topology reconstruction and simplification techniques (TSteiner^{Rec}) can not only reduce TSteiner duration by 25.9% but also achieve better-sign-off timing performance. Our findings advocate for considering sign-off timing performance during early stage routing topology generation, including adjusting Steiner points and edges. By adopting our concurrent routing topology optimization framework, designers can improve sign-off timing performance and ensure effective physical synthesis. An advanced technology may bring new challenges to the future early stage timing optimization works. Further, the learning-guided optimization flow can also be extended to support more objectives and features, e.g., skew, congestion, routing detour and the pin accessibility, since the deep learning technique is useful to differentiate these hard-to-formulate objectives.

REFERENCES

[1] A. B. Kahng, "New game, new goal posts: A recent history of timing closure," in *Proc. 52nd Annu. Design Autom. Conf.*, 2015, pp. 1–6.

[2] C. Chu, E. F. Young, D. K. Tong, and S. Dechu, "Retiming with interconnect and gate delay," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2003, pp. 221–226.

[3] P. Liao, S. Liu, Z. Chen, W. Lv, Y. Lin, and B. Yu, "DREAMPlace 4.0: Timing-driven global placement with momentum-based net weighting," in *Proc. IEEE/ACM Design, Autom. Test Europe (DATE)*, 2022, pp. 939–944.

[4] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *Proc. 59th ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1315–1320.

[5] P. Liao, D. Guo, Z. Guo, S. Liu, Y. Lin, and B. Yu, "DREAMPlace 4.0: Timing-driven placement with momentum-based net weighting and Lagrangian-based refinement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 10, pp. 3374–3387, Oct. 2023.

[6] C. J. Alpert, A. B. Kahng, C. Sze, and Q. Wang, "Timing-driven Steiner trees are (practically) free," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2006, pp. 389–392.

[7] C. J. Alpert et al., "Prim-Dijkstra revisited: Achieving superior timing-driven routing trees," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2018, pp. 10–17.

[8] S. Held, D. Müller, D. Rotter, R. Scheifele, V. Traub, and J. Vygen, "Global routing with timing constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 406–419, Feb. 2018.

[9] G. Chen and E. F. Young, "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 6, pp. 1217–1230, Jun. 2020.

[10] D. Wu, J. Hu, M. Zhao, and R. Mahapatra, "Timing driven track routing considering coupling capacitance," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2005, pp. 1156–1159.

[11] X. Gao and L. Macchiarlo, "Track routing optimizing timing and yield," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2011, pp. 627–632.

[12] Y. Wei, Z. Li, C. Sze, S. Hu, C. J. Alpert, and S. S. Sapatnekar, "CATALYST: Planning layer directives for effective design closure," in *Proc. IEEE/ACM Design, Autom. Test Europe (DATE)*, 2013, pp. 1873–1878.

[13] D. Liu, B. Yu, S. Chowdhury, and D. Z. Pan, "TILA-S: Timing-driven incremental layer assignment avoiding slew violations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 231–244, Jan. 2018.

[14] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[15] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2003, pp. 621–625.

[16] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead RC network," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1213–1218.
[17] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1207–1212.
[18] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.
[19] S. Liu, Z. Wang, F. Liu, Y. Lin, B. Yu, and M. Wong, "Concurrent sign-off timing optimization via deep Steiner points refinement," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.
[20] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, 1966.
[21] P.-Y. Chen et al., "A reinforcement learning agent for obstacle-avoiding rectilinear Steiner tree construction," in *Proc. ACM Int. Symp. Physical Design (ISPD)*, 2022, pp. 107–115.
[22] J. Liu, G. Chen, and E. F. Young, "REST: Constructing rectilinear Steiner minimum tree via reinforcement learning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2021, pp. 1135–1140.
[23] A. B. Kahng, R. R. Nerem, Y. Wang, and C.-Y. Yang, "NN-Steiner: A mixed neural-algorithmic approach for the rectilinear Steiner minimum tree problem," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 13022–13030.
[24] W. L. Hamilton, "Graph representation learning," in *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, San Rafael, CA, USA: Morgan Claypool Publ., 2020, pp. 1–159.
[25] Z. Wang et al., "Functionality matters in netlist representation learning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 61–66.
[26] M. Wang et al., "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2019, *arXiv:1909.01315*.
[27] A. Paszke et al., "Automatic differentiation in PyTorch," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2017, pp. 1–4.
[28] A. B. Kahng, L. Wang, and B. Xu, "TritonRoute: The open-source detailed router," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 3, pp. 547–559, Mar. 2021.
[29] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.
[30] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2006, pp. 464–471.
[31] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young, "CUGR: Detailed-routability-driven 3-D global routing with probabilistic resource model," in *Proc. ACM/IEEE Design Autom. Conf.*, 2020, pp. 1–6.
[32] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Young, "Dr. CU 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–7.
[33] "OpenCores." 1993. [Online]. Available: https://opencores.org
[34] "SkyWater open source PDK." 2020. [Online]. Available: https://github.com/google/skywater-pdk

Siting Liu received the B.S. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2020. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2024.

Her research interests include physical synthesis, machine learning application, and GPU acceleration in VLSI CAD algorithms.

Dr. Liu is a recipient of the Best Paper Award from DATE 2022 and the Best Paper Award Nomination from DATE 2021.

Ziyi Wang received the B.S. degree from the Department of Computer Science and Technology, Fudan University, Shanghai, China, in 2021. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests include graph learning applications in electronic design automation and logic synthesis.

Fangzhou Liu received the B.E. degree in electronic science and engineering from Nanjing University, Nanjing, China, in 2023. She is currently pursuing the Ph.D. degree with The Chinese University of Hong Kong, Hong Kong, supervised by Prof. Bei Yu.

Her research focuses on applying machine learning techniques to EDA and logic synthesis optimization.

Yibo Lin (Member, IEEE) received the B.S. degree in microelectronics from Shanghai Jiaotong University, Shanghai, China, in 2013, and the Ph.D. degree in electrical and computer engineering from The University of Texas at Austin, Austin, TX, USA, in 2018, advised by Prof. David Z. Pan.

He worked as a Postdoctoral Researcher with the University of Texas at Austin from 2018 to 2019. He currently is an Assistant Professor with the School of Integrated Circuits, Peking University, Beijing, China. His research interests include physical design, machine learning applications, and heterogeneous computing in VLSI CAD.

Dr. Lin is a recipient of the Best Paper Awards at Premier EDA/CAD journals/conferences like IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, DAC, DATE, and ISPD.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Prof. Yu received 11 Best Paper Awards from ICCAD 2024 and 2021 and 2013, IEEE TSM 2022, DATE 2022, ASPDAC 2021 and 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, six ICCAD/ISPD Contest Awards, IEEE CEDA Ernest S. Kuh Early Career Award in 2021, DAC Under-40 Innovator Award in 2024, and Hong Kong RGC Research Fellowship Scheme Award in 2024. He has served as a TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees.

Martin D. F. Wong (Fellow, IEEE) received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, Canada, in 1979, and the M.S. degree in mathematics and the Ph.D. degree in CS from the University of Illinois at Urbana-Champaign (UIUC), Champaign, IL, USA, in 1981 and 1987, respectively.

He was a Bruton Centennial Professor of CS with The University of Texas at Austin, Austin, TX, USA, and the Edward C. Jordan Professor of ECE with UIUC. From August 2012 to December 2018, he was the Executive Associate Dean of the College of Engineering, UIUC. From January 2019 to August 2023, he was the Dean of Engineering and the Choh-Ming Li Professor of Computer Science and Engineering with The Chinese University of Hong Kong, Hong Kong. Since August 2023, he has been with Hong Kong Baptist University (HKBU), Hong Kong, as the Provost and a Chair Professor of Computer Science. He has published around 500 papers and graduated over 50 Ph.D. students in EDA. His main research interest is in electronic design automation.

Prof. Wong is a Fellow of ACM.