To Tackle Cost-Skew Tradeoff: An Adaptive Learning Approach for Hub Node Selection

***Abstract*—In chip design, *skew* is a pivotal factor that significantly influences the overall performance for routing. A major challenge is how to achieve an appropriate trade-off between the total wire-length cost and skew. Selecting hub nodes is an effective method to improve this cost-skew trade-off. In this paper, we propose a novel reinforcement learning-based method for hub node selection, where our key idea is leveraging an effective adaptive learning strategy. Moreover, our approach is particularly suitable for solving large-scale routing instances. The empirical results suggest that our method can achieve promising performance on both small-scale and large-scale clock nets, implying its potential practical significance in EDA.**

***Index Terms*—Skew, wirelength, MSPD.**

I. Introduction

In the past decades, the problem of designing efficient algorithms for various VLSI routing applications has attracted substantial attention [1]–[6]. The routing quality are closely related to several crucial metrics in chip design, which are the so-called "*PPA*", i.e., *performance*, *power*, and *area* [7]. The design procedure of interconnect trees relies on several widely studied optimization objectives, among which the total tree **cost** is the most prevalent, typically measured by wire length. Additionally, the **skew**, defined as the difference in signal travel time between the longest and shortest root-leaf paths, is also a crucial factor, as illustrated in Fig. 1. For different applications, these objectives may have different priorities.

In this paper, we focus on designing an effective routing algorithm for tackling the tradeoff between the wire-length and skew. For chip design, the "clock frequency" is an important performance indicator, as it determines the number of instructions that the processor can execute per second [8]. The clock frequency is the reciprocal of the clock cycle, which can be affected by several different factors in practice. In particular, the clock skew often seriously influences the clock cycle; as noted in [9], it can account for even more than 10% of the cycle time. Consequently, how to effectively address the clock skew issue is an imperative problem that we are confronted with for designing high-quality chips.

Actually, the skew issue is a long historical topic in the research of EDA. Several classical methods were proposed before, such as the methods of *Zero-Skew Tree (ZST)* and *Bounded-Skew Tree (BST)* [8]. The goal of ZST is to build a routing tree where the signal from the root node can reach all the terminals at the same time, so that the skew is zero. Boese and Kahng [10] presented the Deferred-Merge Embedding (DME) algorithm for building a ZST that can accomplish precise zero skew. However, a drawback of ZST is that it often results in long wire-length and it is difficult to be realized due to the limitations of the manufacturing process [8]. This motivates researchers to consider an alternative solution, Bounded-Skew Tree [11]–[13], which is a clock tree with a maximum skew bounded by a given threshold $UB > 0$. It is worth noting that when there is no skew restriction (i.e., $UB = \infty$), the BST problem is simply equivalent to the classical Rectilinear Steiner Minimum Tree (RSMT) problem [14]. Several heuristic methods were developed to address the bounded skew Steiner tree construction problem in the past decades [12], [13]. Cong et al. [11] provided an improved DME algorithm called "BST-DME", which is specifically designed for the BST problem. Additionally, several elegant approximation algorithms were also proposed for the skew problem [15], [16]. More recently, Han et al. [17] studied the tradeoff between cost and skew and formulated the bounded skew problem as a flow-based integer linear programming.

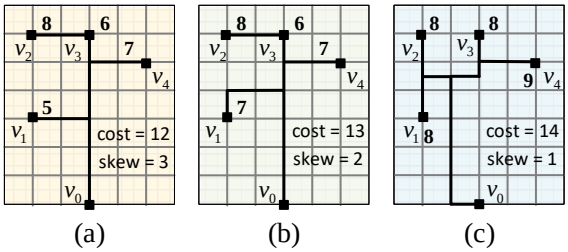


Fig. 1: This illustration shows that different routing priorities can lead to various topologies; v_0 is designated as the root node, and the number marked alongside each vertex is the distance to v_0 in the tree. Figure (a) depicts a design focusing on minimizing wire length [8], while Figure (b) illustrates a trade-off between cost and skew. Figure (c) primarily emphasizes skew, aiming to minimize the time difference for signals traveling from the root node to other nodes.

Recently, building upon the classic Prim-Dijkstra (PD) algorithm [18], Kahng et al. [19] introduced the **Multi-Source Prim-Dijkstra (MSPD)** algorithm for the cost-skew tradeoff problem, which will be detailed in Section II-B. The main insight of the MSPD approach is to select several centrally located "hub" nodes (termed "sources" in their paper) to serve as root connections and enforce a constraint that prevents other nodes from connecting directly to the root. Subsequently, the modified PD algorithm (i.e., "MSPD") is employed to generate the tree topology. From the examples illustrated in Fig. 2, we can see the advantage of utilizing hub nodes.

However, there is still a gap for efficiently identifying the hub nodes. **The first challenge** comes from the high computational cost, since an optimal algorithm would require

an exhaustive search for the best combination of hub nodes. Although the method proposed in [19] could reduce the complexity to a certain extent by only considering those centrally located node candidates, the resulting smaller solution space could affect the final performance of routing. Moreover, **the second challenge** is that there is currently no effective method capable of adaptively deciding how many hub nodes should be chosen. The existing algorithms require specifying a maximum quantity of hub nodes, and then enumerating different combinations that have the sizes within such an upper bound. Due to the importance of hub nodes in the skew problem, the open-source project *OpenROAD* ("Foundations and Realization of Open, Accessible Design") in collaboration with TILOS AI Institute recently launched an innovation-driven competition in June 2023[1], whose objective is to devise efficient hub node selection algorithms to achieve favorable cost-skew tradeoff.

To tackle those challenges in addressing the cost-skew tradeoff problem, we propose an adaptive learning-based framework. In particular, our framework takes account of the global space for determining the number of chosen hub nodes and their locations. **The main contributions** of our paper are outlined as follows:

1) We propose a novel "Selector $\rightleftarrows$ Conductor" idea that can be seamlessly integrated with the existing reinforcement learning method. The "Selector" network is for selecting hub nodes, and the "Conductor" network is used for monitoring the current environment and actions to determine when to halt operations. This cooperation mechanism between them yields an adaptive learning process, and also enhances our model's ability to manage larger-scale instances.
2) To evaluate the performance of our proposed method, we conducted a set of experiments on both small-scale and large-scale problems using diverse datasets sourced from synthetic data and the OpenROAD competitions. In the experiments, our learning-based method achieved the best Pareto frontier in terms of cost and skew across various test cases and demonstrated satisfying efficiency.

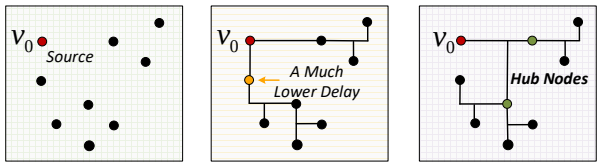
II. Preliminaries

A. Problem Formulation

An instance of a clock routing problem comprises n nodes, denoted as $V = \{v_0, v_1, \cdots, v_{n-1}\}$, with v_0 acting as the **root node** and other nodes serving as the **sinks**. The goal is to construct a routing tree such that the signal from the root, v_0, can be transmitted to the other nodes. Suppose a routing tree $T = (V, E)$ is constructed to connect all the nodes in V. In this routing tree T, let $e_{i,j}$ denote the edge linking v_i with v_j; its edge cost is defined as the Manhattan distance $|x_j - x_i| + |y_j - y_i|$ (suppose $v_i = (x_i, y_i)$ and $v_j = (x_j, y_j)$). Consequently, the ***cost*** of the clock tree, denoted as $\text{cost}(T)$, is the sum of all edge costs, i.e.,

$$\text{cost}(T) = \sum_{e_{ij} \in E} |x_j - x_i| + |y_j - y_i|. \tag{1}$$

[1]https://github.com/TILOS-AI-Institute/multi-source-Prim-Dijkstra



(a) (b) (c)

Fig. 2: This figure illustrates the role of hub nodes. Figure (a) shows the locations of all nodes. Figure (b) depicts a routing tree without adding hub nodes, leading to significant skew; the orange sink experiences a much lower delay than the other black sinks. Figure (c) presents a routing tree generated by the MSPD algorithm, demonstrating a reduction in skew through the aid of hub nodes.

The clock ***skew*** refers to the difference between the highest and lowest arrival times of the clock signal across the $n-1$ sinks. Formally, we define:

$$\text{skew}(T) = \max_{v_i, v_j \in V} |t(v_0, v_i) - t(v_0, v_j)|, \tag{2}$$

where $t(v_0, v_i)$ denotes the signal delay between the nodes v_0 and v_i, estimated by using either the *linear delay* or the *Elmore delay* model [8]. The linear delay model equates delay directly with the path length between v_0 and v_i, ignoring other details in the circuit, whereas the Elmore model is more complex that also takes the resistance and capacitance into account. Due to its simplicity for calculation, the linear model remains popular in commercial tools [8].

As discussed in Section I, a central problem for practical clock routing is designing an effective algorithm for achieving a pleasant **cost-skew tradeoff**. Formally, given a node set V and a parameter $\alpha \in [0, 1]$, our objective is to construct a routing tree T to minimize:

$$\alpha * \text{cost}(T) + (1 - \alpha) * \text{skew}(T). \tag{3}$$

B. Multi-Source Prim-Dijkstra (MSPD)

Multi-Source Prim-Dijkstra (MSPD) [19] is a variant of the PD algorithm [18]. Given a clock routing instance as shown in Fig. 2(a), a major difference of MSPD is that the algorithm selects a set of hub nodes and only these designated nodes can link directly to the root. The purpose of this constraint is to avoid large skew (see the comparison between Fig.2(b) and Fig.2(c)). It is noteworthy that the authors of [19] proposed two algorithms for hub node selection. The first one is an optimal algorithm; however, it incurs a significantly high computational cost due to its exhaustive search for the best combination of hub nodes. To avoid such a large combination space, the authors also introduced an alternative approach that only enumerates the combinations of the potential hub nodes located centrally to the root; the "centrality" is measured by several pre-specified parameters (we refer the reader to their article [19] for more details). Although this method reduces the size of the candidate set, it results in a narrower solution space that could downgrade the obtained solution quality. Moreover, the prior methods usually require specifying a maximum quantity of hub nodes, which may not be easy to determine and thus also restricts the capability to optimize the

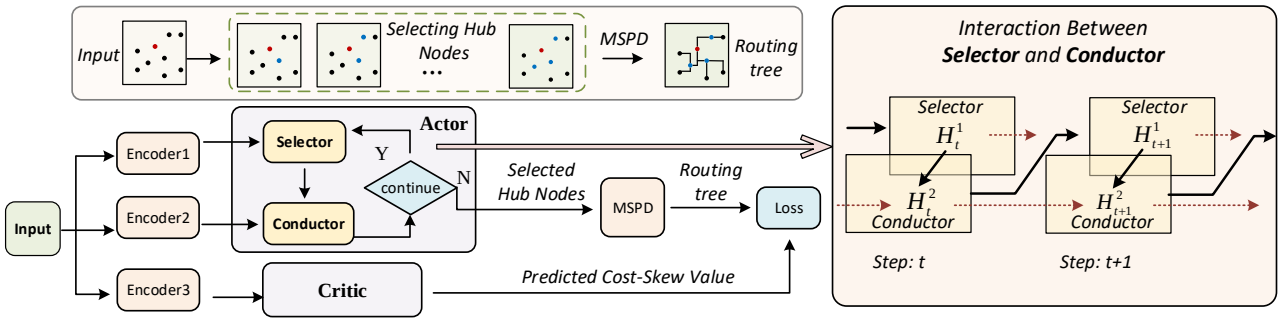


Fig. 3: Our proposed RL-based framework, whose core neural components are the Selector and Conductor networks. The Selector network determines the hub node in each step, while the Conductor network monitors the environment and existing actions, assessing the need for further actions. The right figure illustrates the detailed interaction between these networks; the black arrow indicates the execution order, while the dotted brown arrow represents information passed from previous steps. A running example is also shown in this figure (top left): given an input node set, the red node is the root, and the hub nodes (labeled in blue) can be determined by the Selector and Conductor networks. Then, the instance is processed by the MSPD algorithm and eventually, the routing tree with cost-skew tradeoff is obtained.

objective of (3). These practical issues motivate us to develop an efficient and adaptive method for hub nodes selection in Section III.

C. Reinforcement Learning

Our proposed method utilizes Reinforcement learning (RL), which is an important machine learning methodology that has been extensively studied for many different real-world applications [20]. RL uses an **agent** to perform the learning process, where the agent can make decisions by taking actions in a specified environment. This RL process consists of interacting with the environment, evaluating the current state, and receiving rewards based on the outcomes of its actions. In particular, the **Actor-Critic** RL framework [20] has been successfully applied to solve several routing tasks in EDA [4], [5]. The "Actor" network typically involves a sequence of decision-making steps, such as connecting a node to the existing tree; ultimately this network will generate a routing tree. Meanwhile, the "Critic" network is employed to facilitate the training of the Actor network.

For completeness, we introduce the three important terminologies of RL that will be used throughout Section III: 1) **State:** At any given time step, the state includes the locations of all nodes in V, the root node v_0, and the currently selected set of hub nodes; 2) **Action:** At each step t, the agent selects a new hub node that has not been previously chosen; subsequently, the agent must also determine whether to continue selecting additional hub nodes; 3) **Reward:** In our model, the reward is computed at the conclusion of the RL process and is combined with the critic's prediction to assess the quality of the selected hub nodes.

Our method also follows the above overall framework; however, our proposed method needs to select specific hub nodes during the tree construction, necessitating some significant new ideas for this purpose.

III. Our Proposed Framework

In this section, we introduce our proposed approach. First, we analyze the technical challenges and provide an overview of our framework below.

The challenges and our high-level idea. We are aware of several recently proposed RL-based methods for routing problems, such as the method of [4] for minimizing the wirelength and the method of [5] for minimizing the timing-driven objective. Nevertheless, we are still confronted with some new significant challenges in addressing the issue of cost-skew tradeoff. Both of those methods are designed for generating the tree using a Markov chain, with the aid of some clearly defined termination criteria. But in our problem, our aim is to select several key hub nodes. The immediate issues for realizing such a model include how to handle the uncertainty of the hub nodes and how to overcome the absence of a predefined termination criterion (note that even the number of needed hub nodes is also unknown in advance). Thus, our task is to design some flexible and easy-to-implement method that can adaptively select the hub nodes and efficiently construct the routing tree with a qualified cost-skew tradeoff.

Our high-level idea relies on a novel "**Selector $\rightleftarrows$ Conductor**" reinforcement learning framework. To identify the most appropriate candidates for hub nodes, we design a "Selector" network that is dedicated to hub node selection. More importantly, we also integrate a "Conductor" module into the classical Actor module, which can help the model monitor the environment and existing actions, so as to determine the necessity of further actions. The architecture of our whole model is built upon this adaptive Selector $\rightleftarrows$ Conductor mechanism, as shown in Fig. 3.

We present the technical details in the following parts. In Section III-A, we introduce the Encoder module that is used for representing the input data. The "Selector" and "Conductor" modules are two key parts to realize the adaptive learning in our framework, which are respectively introduced in Section III-B and Section III-C. Finally, we describe the Critic network and the training method in Section III-D.

Superscripts of the notations. For ease of presentation, in the following sections, we use the superscripts from $\{0, 1, 2, 3\}$ to differentiate the parameters and variables in different mod-

ules. For example, W^0 denotes the parameters in the Encoder modules, W^1 is for the Selector network, W^2 is for the Conductor network, and W^3 is for the Critic network.

A. Encoder

The goal of the Encoder is to embed an input instance V (which contains n vertices in $2D$) into a higher dimensional space $\mathbb{R}^d$ with some appropriately set value d; this new representation will serve the following modules as shown in Fig. 3. Our Encoder model is based on the seminal **attention mechanism** in deep learning [21], which is inspired by human visual attention, enables the model to weigh the importance of different inputs and allows the model to focus on different parts of the input sequence. Note that we have three encoders for the Selector, Conductor, and Critic networks, respectively (as shown in Fig. 3), where they have the same structure but with different trained parameters.

Given the routing instance V containing n nodes, we build the input to our encoder that is the form of a matrix $X_{\texttt{ini}} \in \mathbb{R}^{n\times 3}$. The 3-dimensional information includes the 2D coordinate information (x, y) and a label indicating whether it is a root node (labeled as 1 if it is a root node, and 0 otherwise). Then we embed these nodes into a higher dimensional space for enriching their encoded information: the input vector $X_{\texttt{ini}} \in \mathbb{R}^{n\times 3}$ is processed through a feed-forward neural network layer, obtaining the output $\hat{X}_{\texttt{ini}} \in \mathbb{R}^{n\times d_0}$ (where the integer d_0 is a pre-specified hyperparameter in our model). Formally, our feed-forward neural network can be calculated as:

$$\hat{X}_{\texttt{ini}} = \max\left\{\vec{0}, X_{\texttt{ini}} \times W_1^0 + \vec{b_1^0}\right\} \times W_2^0 + \vec{b_2^0}, \tag{4}$$

where $\vec{0}$ is the zero vector, and $W_1^0 \in \mathbb{R}^{3\times d_0}, W_2^0 \in \mathbb{R}^{d_0\times d_0}, \vec{b_1^0} \in \mathbb{R}^{d_0}, \vec{b_2^0} \in \mathbb{R}^{d_0}$ are the parameters to be trained in this feed-forward layer. Considering the importance of the root node, we need to ensure that the information of the root node v_0 should be accessible to other nodes. For this purpose, we concatenate the embedding of the root node with that of each other node: the obtained new embedding is $X \in \mathbb{R}^d$ ($d = 2d_0$), and each row

$$X[i] = \left[\hat{X}_{\texttt{ini}}[i], \hat{X}_{\texttt{ini}}[0]\right] \tag{5}$$

is the augmented embedding of the node v_i, where $\hat{X}_{\texttt{ini}}[i]$ is the i-th row of $\hat{X}_{\texttt{ini}}$, i.e., the original embedding of v_i, and $\hat{X}_{\texttt{ini}}[0]$ is the original embedding of the root.

Then, the embedding X is fed into the "multi-head" attention layer [21]. The function of this layer is to aggregate and capture information from all the nodes. Through this multi-head attention layer, the information from various nodes is aggregated to create a new representation for each node, and thus an enriched embedding can be obtained. As mentioned at the beginning of this subsection, we have this Encoder network before each of the downstream Selector, Conductor, and Critic networks. We denote the corresponding outputs from these three Encoders as E^1, E^2, and $E^3 \in \mathbb{R}^{n\times d}$, respectively.

B. Selector Network

The function of the Selector network is to identify the nodes that have high potential to become the hubs. The methodology for identifying hub nodes leverages the **Pointer Network** [22], which has shown significant promise in addressing various combinatorial optimization problems with sequence generation tasks, such as the planar Travelling Salesman Problem (TSP) [22] and the Rectilinear Minimum Spanning Tree (RMST) problem [4].

Before diving into the details of our Selector and Conductor networks, we illustrate this **Selector** $\rightleftarrows$ **Conductor** cooperation mechanism in Fig. 3 (right). To assist the Selector and Conductor in making decisions, we define hidden states $H_t^1 \in \mathbb{R}^{d_1}$ and $H_t^2 \in \mathbb{R}^{d_1}$ (initially set to $\vec{0}$) to encode the environment at step t, where d_1 is a hyperparameter. At step t, the Selector network selects a new hub node based on the hidden state H_t^1. The action of selecting the hub node is encoded and can be passed to the Conductor to update its hidden state, resulting in H_t^2, thereby assisting the Conductor in making decisions. If the Conductor chooses to continue selecting hub nodes, the Selector progresses to step $t+1$. It is important to note that the encoding of the action at step t will be employed to update the hidden state of the Selector, yielding H_{t+1}^1. Subsequently, the Selector will use H_{t+1}^1 to select hub nodes as described above.

Now, we introduce more details for the Selector and leave the details for the Conductor to Section III-C. At step t, leveraging H_t^1, the probabilities of those n nodes being selected as the hub node can be obtained by the following n-dimensional vector:

$$p_t^{\texttt{sel}} = \text{Softmax}\left(\tanh\left(\text{Mask}\left(z_t^1\right)\right)\right), \tag{6}$$

where the value $z_t^1 = {g^1}^\top \tanh\left(W_1^1 {E^1}^\top + W_2^1 H_t^1\right)$ with $W_1^1 \in \mathbb{R}^{d_2\times d}$, $W_2^1 \in \mathbb{R}^{d_2\times d_1}$, and $g^1 \in \mathbb{R}^{d_2}$ being the parameters to be trained (the integer d_2 is a pre-specified hyperparameter), and E^1 represents the node embeddings for the Selector network. The computation mechanism of Eq. 6 is based on the Pointer Network [22], and let us explain it in detail. The value $z_t^1 \in \mathbb{R}^n$ represents the logits for selecting different nodes. For the nodes that have already been selected, we use $\text{Mask}(z_t^1)$ to set their values to $-\infty$, thereby preventing them from being re-selected. The "softmax" function normalizes this vector and outputs the probabilities of those n nodes to be selected, i.e., the vector $p_t^{\texttt{sel}} \in \mathbb{R}^n$. Note that during the training phase, selections are made according to the probability distribution defined by $p_t^{\texttt{sel}}$, whereas in the testing phase, the model directly chooses the node with the highest probability.

Suppose the Selector network selects the i-th node ($1 \le i \le n-1$). It is important to embed this action to assist subsequent decisions. To achieve this, we use $A_t^1 \in \mathbb{R}^{d_1}$ to encode the selection action. Since the selected hub node will be connected to the root node, the encoding of the i-th node $E^1[i] \in \mathbb{R}^d$ should be combined with the encoding of the root node $E^1[0]$:

$$A_t^1 = W_a^1 E^1[i] + W_r^1 E^1[0], \tag{7}$$

where $W_a^1 \in \mathbb{R}^{d_1 \times d}$ and $W_r^1 \in \mathbb{R}^{d_1 \times d}$ are the parameters to be trained. $W_r^1 E^1[0]$ and $W_a^1 E^1[i]$ represent the transformed embeddings of the root node and the currently selected hub node, respectively. After the action is encoded, the hidden state of the Selector for step $t+1$ is updated as follows:

$$H_{t+1}^1 = \max\left(0, \max\left(H_t^1, W_u^1 A_t^1\right)\right), \tag{8}$$

where $W_u^1 \in \mathbb{R}^{d_1 \times d_1}$ is the parameter to be trained. The new hidden state H_{t+1}^1 is obtained by integrating the previous state H_t^1 with the current action encoding $W_u^1 A_t^1$.

C. Conductor Network

The Conductor network monitors the environment and assesses existing actions to determine whether to select additional hub nodes. Similar to the Selector, it also maintains a hidden state H_t^2 that records information from previous actions. At step t, the Conductor needs to perceive the action made by the Selector; thus, the selection action of the Selector at this step should be encoded to facilitate better decision-making by the Conductor. We employ the same method as illustrated in Eq. 7 and Eq. 8 to encode the action and update the hidden state of the Conductor, but with different trained parameters.

At step t, based on the hidden state H_t^2, we compute:

$$z_t^2 = {g^2}^\top \tanh\left(W_1^2 {E^2}^\top + W_2^2 H_t^2\right), \tag{9}$$

where $W_1^2 \in \mathbb{R}^{d_2 \times d}$, $W_2^2 \in \mathbb{R}^{d_2 \times d_1}$, and $g^2 \in \mathbb{R}^{d_2}$ are parameters to be trained, and E^2 represents the node embeddings for the Conductor network. The term $z_t^2 \in \mathbb{R}^n$ is expected to produce a vector that integrates information from the hidden state H_t^2 and the node embeddings. Based on z_t^2, the probability of whether to continue selecting hub nodes at step t can be computed as:

$$p_t^{\texttt{con}} = \text{Sigmoid}\left(\vec{\mathbf{1}} \cdot \left(\tanh\left(z_t^2\right)\right)\right), \tag{10}$$

where $\vec{\mathbf{1}} \cdot \left(\tanh\left(z_t^2\right)\right)$ represents the sum of the elements of $\tanh\left(z_t^2\right)$. Eq. 10 aggregates information from different nodes to inform the decision on whether to continue selecting a hub node. If $p_t^{\texttt{con}} > 0.5$, the process proceeds to the next step; otherwise, the Selector will stop.

D. Critic and Training Method

In reinforcement learning, particularly in an Actor-Critic model, the Critic module is often used to assist the learning process [20]. Different from the previous RL-based networks [4], [5], our Critic module tries to predict the "cost-skew tradeoff value" of the input instance V, where the predicted value is denoted as $\texttt{Cr}(V)$. Recall that our Critic network receives the input $E^3 \in \mathbb{R}^{n \times d}$ from the Encoder, as described in Section III-A. Initially, we compute an "attention-weighted" representation of E^3: $\text{Attn}(E^3) = \text{softmax}\left(\tanh\left(E^3\right) g^3\right)^\top E^3$, where $g^3 \in \mathbb{R}^d$ is a trainable parameter vector, and $\text{Attn}(E^3)$ represents a weighted sum of the encoded information from different nodes. Then, two fully connected layers are utilized to obtain the predicted cost-skew value:

$$\texttt{Cr}(V) = \max\left(0, \text{Attn}(E^3) W_1^3 + \vec{b_1^3}\right) W_2^3 + b_2^3, \tag{11}$$

where $W_1^3 \in \mathbb{R}^{d \times d_3}$, $\vec{b_1^3} \in \mathbb{R}^{d_3}$, $W_2^3 \in \mathbb{R}^{d_3 \times 1}$, and $b_2^3 \in \mathbb{R}$ are trainable parameters (with d_3 being a hyperparameter). After obtaining $\texttt{Cr}(V)$, we introduce the training method of the Actor and Critic networks.

Let θ_1 and θ_2 denote the parameters for the Actor and Critic, respectively. Suppose the batch size is $N \geq 1$ and the set of sampled instances for training are $\mathcal{V} = \{V_1, V_2, \cdots, V_N\}$. For each $1 \leq i \leq N$, assume $\mathcal{T}_i$ is the trajectory executed by the Actor, which follows a Markov decision process. Each step consists of selecting a hub node and deciding whether to continue. The probability of trajectory $\mathcal{T}_i$ under the Actor's policy is denoted as $p_{\theta_1}(\mathcal{T}_i)$. This probability is calculated as the product of the probabilities of the actions taken, including selecting hub nodes and making continuation decisions. The cost-skew value of the solution produced by the Actor is denoted by $\texttt{Ac}(V_i, \mathcal{T}_i)$ for each i. The loss for the Actor, denoted as $\mathcal{L}_{\text{A}}(\theta_1 \mid \mathcal{V})$, is calculated by:

$$\frac{1}{N} \sum_{i=1}^{N} \left[\texttt{Cr}(V_i) - \texttt{Ac}(V_i, \mathcal{T}_i) + \sum_{t=1}^{|\mathcal{T}_i|} R_t \gamma^t\right] \log p_{\theta_1}(\mathcal{T}_i | V_i), \tag{12}$$

where $|\mathcal{T}_i|$ denotes the number of hub nodes chosen by the Actor in trajectory $\mathcal{T}_i$. R_t and γ respectively represent the intermediate reward and the decay factor, which are used to balance the exploration and exploitation in RL [20].

The loss for the Critic $\mathcal{L}_{\text{C}}(\theta_2 | \mathcal{V})$ is based on the Mean Squared Error (MSE) function:

$$\frac{1}{N} \sum_{i=1}^{N} \|\texttt{Cr}(V_i) - \texttt{Ac}(V_i, \mathcal{T}_i)\|_2^2. \tag{13}$$

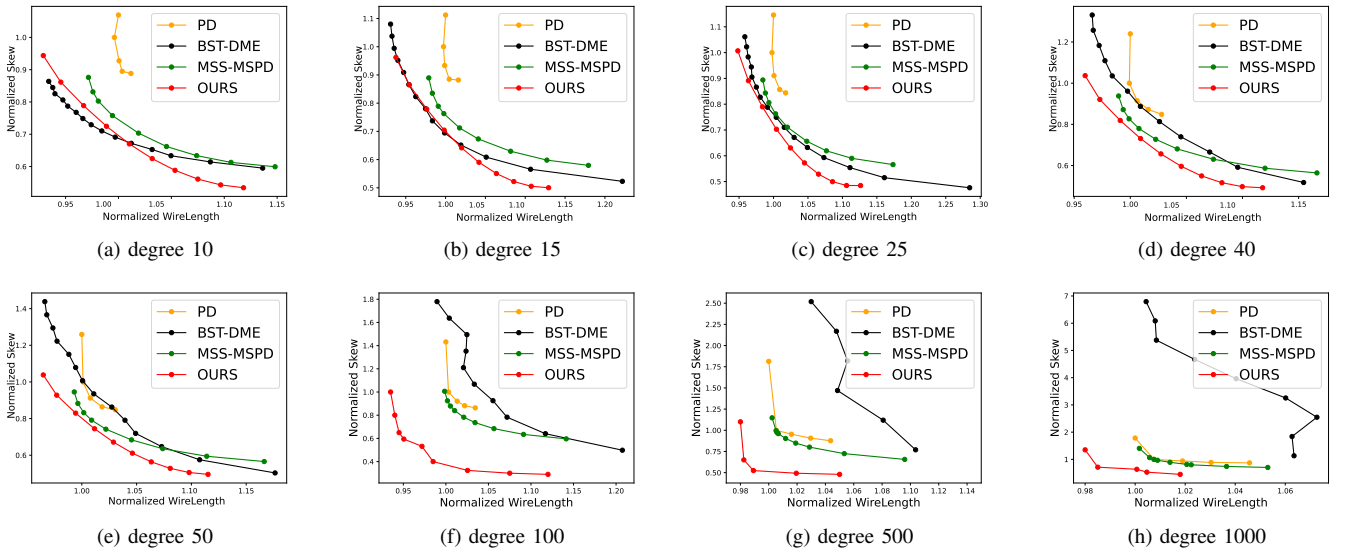
Both the Actor and Critic networks are trained using the Stochastic Gradient Descent (SGD) method [23].

IV. EXPERIMENTAL RESULTS

We trained our model using PyTorch and conducted our experiments on a 64-bit Linux system equipped with eight NVIDIA RTX A6000 GPUs. We compare the performance of our algorithm with three representative baselines: (1) the "BST-DME" algorithm outlined in [11]; (2) a heuristic **m**ulti-**s**ource **s**election method introduced in [19], denoted as "MSS-MSPD"; (3) the "PD" algorithm [18], [24] introduced in Section II-B. Our algorithm is labeled as "OURS". We present our results separately for small-scale ($n \leq 50$) and large-scale ($n > 50$) instances.

A. Small-Scale

We trained our models for degrees ranging from 10 to 50 on randomly generated training data. For degree 10, we trained the model with mini-batch size B = 2048 for $100k$ iterations. Subsequently, the batch size was diminished for



(a) degree 10 (b) degree 15 (c) degree 25 (d) degree 40

(e) degree 50 (f) degree 100 (g) degree 500 (h) degree 1000

Fig. 4: Figures (a) to (e) show the comparison results of our framework with the BST-DME, MSS-MSPD, and PD algorithms for data with degrees of 10, 15, 25, 40, and 50, respectively. Figures (f) to (h) show the comparison results for large degrees of 100, 500, and 1000, respectively.

each ascending degree. In this part, we compare the experimental performance for nets with degrees ≤ 50, using the test data (300 instances for each degree) derived from the recent OpenROAD contest [25], which is generated by a well-designed random algorithm, ensuring the diversity of distribution.

The Prim's algorithm and Dijkstra's algorithm are commonly used representative algorithms for optimizing cost and skew [18], so we particularly take their obtained values "$\mathbf{cost}(\mathbf{T_P})$" and "$\mathbf{skew}(\mathbf{T_D})$" as the baseline values, where T_P and T_D are the trees obtained by Prim's and Dijkstra's algorithms, respectively. Suppose T is the tree obtained by an algorithm (either our method or a baseline), then we evaluate this algorithm's performance by the ratios $\frac{cost(T)}{cost(T_P)}$ and $\frac{skew(T)}{skew(T_D)}$, where are respectively referred as "**normalized wirelength**" and "**normalized skew**". The comparative performance of these algorithms is illustrated in Fig. 4 (a)-(e). It can be observed that our algorithm achieves the best Pareto frontiers, and the performance difference between "OURS" and the other algorithms gradually increases as the scale increases. To more clearly illustrate this trend, we also conducted the experiment on larger scale instances in the next section.

B. Large-Scale

We compare the algorithms on randomly generated large-degree networks (degrees of 100, 500, and 1000, with 100 instances for each degree). The results are shown in Fig. 4 (f)-(h). It can be seen that the BST-DME algorithm has relatively poor performance on large-scale instances. Since the BST-DME algorithm has to obey skew constraints, the wirelength cost caused by this constraint accumulates rapidly with the expansion of scale, and our framework maintains a superior performance relative to other algorithms. It is worth noting that we directly use the model trained in degree 100 for degrees 500 and 1000 without fine-tuning, which implies the effectiveness of applying our model to large-scale problems. The runtimes of various algorithms across different degrees are shown in Table I. Since most of the computational load of our learning-based model belongs to the training stage, the running time of solving a new instance in the test stage is quite small. Our runtime is comparable with PD and slightly slower than BST-DME, but our algorithm enjoys significantly better performance than these two baselines as shown in Fig. 4.

TABLE I: Average runtime performance over 1000 samples for small-scale tests and over 100 samples for large-scale tests.

Degree	BST-DME(s)	MSS-MSPD(s)	PD(s)	OURS(s)
10	0.022	0.007	0.001	0.003
15	0.045	0.074	0.005	0.007
25	0.057	0.107	0.009	0.009
40	0.081	0.240	0.014	0.015
50	0.094	0.400	0.018	0.019
100	0.202	2.575	0.103	0.104
500	1.216	136.6	1.387	1.415
1000	2.480	1008	4.392	5.362

V. Conclusion

In this paper, we propose an adaptive learning-based approach to select hub nodes for the cost-skew tradeoff problem. The key idea relies on a novel Selector $\rightleftarrows$ Conductor cooperation mechanism integrated with the RL framework. Our experimental results illustrate the promising performance of our model on small and large-scale nets. In the future, we can consider further speeding up our method through other aspects. For example, it is interesting to consider whether the MSPD algorithm can be implemented on GPUs. It is also worth investigating whether our adaptive "Selector $\rightleftarrows$ Conductor" RL framework can be applied to solve other challenging optimization problems in EDA.

REFERENCES

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

[2] 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, 2007.

[3] M. D. Moffitt, "Maizerouter: Engineering an effective global router," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 11, pp. 2017–2026, 2008.

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

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

[6] R. Cheng, X. Lyu, Y. Li, J. Ye, J. Hao, and J. Yan, "The policy-gradient placement and generative routing neural networks for chip design," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 350–26 362, 2022.

[7] V. Soteriou, N. Eisley, H. Wang, B. Li, and L.-S. Peh, "Polaris: A system-level roadmap for on-chip interconnection networks," in *2006 International Conference on Computer Design*. IEEE, 2006, pp. 134–141.

[8] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer, 2011, vol. 312.

[9] M. R. Guthaus, G. Wilke, and R. Reis, "Revisiting automated physical synthesis of high-performance clock networks," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–27, 2013.

[10] K. D. Boese and A. B. Kahng, "Zero-skew clock routing trees with minimum wirelength," in *[1992] Proceedings. Fifth Annual IEEE International ASIC Conference and Exhibit*. IEEE, 1992, pp. 17–21.

[11] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-skew clock and steiner routing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 3, no. 3, pp. 341–388, 1998.

[12] D. J. Huang, A. B. Kahng, and C.-W. A. Tsao, "On the bounded-skew clock and steiner routing problems," in *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, 1995, pp. 508–513.

[13] A. B. Kahng and C.-W. A. Tsao, "Practical bounded-skew clock routing," *High Performance Clock Distribution Networks*, pp. 87–103, 1997.

[14] M. R. Garey and D. S. Johnson, "The rectilinear steiner tree problem is np-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[15] A. Z. Zelikovsky and I. I. Mandoiu, "Practical approximation algorithms for zero-and bounded-skew trees," *SIAM Journal on Discrete Mathematics*, vol. 15, no. 1, pp. 97–111, 2001.

[16] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai, and A. Tomkins, "Minimizing wirelength in zero and bounded skew clock trees," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 4, pp. 582–595, 2004.

[17] K. Han, A. B. Kahng, C. Moyes, and A. Zelikovsky, "A study of optimal cost-skew tradeoff and remaining suboptimality in interconnect tree constructions," in *Proceedings of the 20th System Level Interconnect Prediction Workshop*, 2018, pp. 1–8.

[18] C. J. Alpert, T. C. Hu, J.-H. Huang, A. B. Kahng, and D. Karger, "Prim-dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 7, pp. 890–896, 1995.

[19] A. B. Kahng, S. Thumathy, and M. Woo, "An effective cost-skew tradeoff heuristic for vlsi global routing," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] 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 *Proceedings of the 2018 International Symposium on Physical Design*, 2018, pp. 10–17.

[25] TILOS-AI-Institute, "Mspd," Github, 2023. [Online]. Available: https://github.com/TILOS-AI-Institute/multi-source-Prim-Dijkstra