A Delay-driven Iterative Technology Mapping Framework

Junfeng Liu, Liwei Ni, Lei Chen, Xing Li, Qinghua Zhao, Xingquan Li, Shuai Ma

***Abstract*—Technology mapping is the pivotal synthesis step that translates abstract logical models into technology-dependent implementations using the designated library, *e.g.,* standard cells for ASICs. The efficient solutions heavily rely on the gate selection guided by estimated delay. However, estimating these delays is sophisticated due to the absence of actual interconnect load and transition time during the mapping. In this paper, we revisit the difficulties of the delay-driven mapping problem and explore three key insights to address these. Inspired by the insights, we first design a structure-aware load-slew model that integrates input transitions and output loads for gate delay estimations. Benefiting from the model, we propose a delay-iterative framework that progressively reduces the overall circuit delay by further aligning library characteristics with logical network structures. Finally, experiments with 130nm and 7nm libraries show its superiority, which averagely reduces circuit delay by 10% with NLDM, and 6% in delay after P&R, as compared to ABC.**

***Index Terms*—Technology mapping, ASIC design, Technology-dependent optimization, Timing analysis, Delay estimation**

I. Introduction

TECHNOLOGY mapping is a crucial synthesis process in ASICs and other digital systems, converting technology-independent circuit descriptions into technology-specific netlists. In the context of ASICs, this mapping translates logical representations, *e.g.,* And-Inverted Graphs (AIG), into networks of standard cells that meet specific performance criteria including area, power, and delay [6], [7], [22], [36]. Over recent decades, the field has witnessed significant advancements in technology mapping optimization for ASIC synthesis [4], [8], [16], [17], [23], [24], [27], [37], [38].

It is known that area minimization of technology mapping is an NP-complete problem, whereas the minimum-delay is regarded as much more sophisticated [17], [25], [26], [34].

Manuscript received xx xxx xxxx; revised xx xxx xxxx and xx xxx xxxx; accepted xx xxx xxxx. Date of publication xx xxx xxxx; data of current version xx xxx xxxx. This work was supported by NSF of China under Grant 61925203, U22B2021, and Major Key Project of PCL (No. PCL2023A03). This paper was recommended by Associate Editor xxx. (*Corresponding author: Shuai Ma.*)

Junfeng Liu is now with Pengcheng Laboratory, Shenzhen, China. He was with the School of Computer Science and Engineering, Beihang University, Beijing, China.

Liwei Ni and Xingquan Li are with Pengcheng Laboratory, Shenzhen, China.

Lei Chen and Xing Li are with Huawei Noah's Ark Lab, Hong Kong, SAR.

Qinghua Zhao are with the School of Artificial Intelligence and Big Data, Hefei University, Hefei, China.

Shuai Ma are with the School of Computer Science and Engineering, Beihang University, Beijing, China (e-mail: mashuai@buaa.edu.cn).

Digital Object Identifier xx.xxxx/TCAD.xxxx.xxxxxxx.

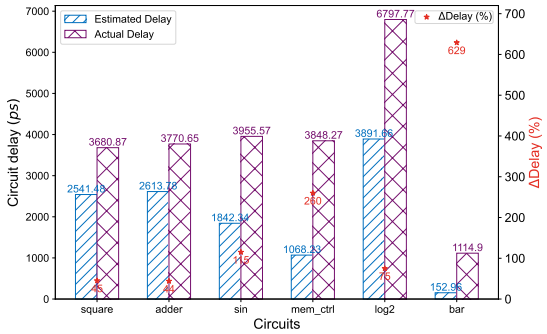

Fig. 1 Motivation Example: the significant gaps between the estimated delay (based on LIDM) and actual delay (based on NLDM) in ABC [4]. ΔDelay quantifies the relative difference between these models.

This complexity arises from the lack of accurate and real-time delay models, compounded by the fact that interconnect loads and transition time remain unknown during the mapping process. By the gate delay estimation model, the advancements in mapping optimization can be categorized into three groups: (1) Methods based on load-independent delay model (LIDM) [4], [5], [16], [30], [37]; (2) Techniques utilizing load-dependent delay model (LDDM) [17], [18], [31]; (3) Approaches employing implicit learning to estimate delays [23], [27].

All these methods employ dynamic programming to address the technology mapping, formulated as a DAG covering problem [30]. In LIDM, Rudell [30] first extends the tree-based min-area mapping to address minimum delay in linear time. Inspired by this, the synthesis tool ABC [4] further sequentially refines the area after the initial load-independent delay-oriented results. Besides, Wu *et al.* [37] and Calvino *et al.* [6] tackle the mapping issues of complex universal gates and multi-output cells, respectively. In LDDM, they directly count the fanout numbers of logical networks to estimate the total capacitive load driven by the gate [17], [18], [31]. The misalignment between library characteristics and mapped netlists restricts the potential of these models for minimum-delay mapping. In learning-based models, Neto *et al.* [27] and Liu *et al.* [23] formalize the problem from two learning perspectives: the first based on cut classification, and the second learning an estimation of the supergate delay *w.r.t.* cuts. They all require improved overall circuit delays as labels to guide the mapper, inevitably resulting in poor generalization.

As highlighted in existing studies, estimating the delay of candidate gates during the mapping process is critical as it fundamentally guides gate selections. Utilizing estimated gate delays, these methods iteratively update the nodes' arrival times

in topological order by dynamic programming [23]. However, gaps in gate delay estimation exacerbate the inaccuracies in circuit delay through cumulative updates, thereby constraining the optimization potential of the mapper. Take Fig. 1 as an example, the average gap in delay for these six circuits reached 152%, with the gap in bar circuit exceeding 600%.

Despite significant efforts in existing studies [3]–[5], [8], [16]–[19], [23], [25]–[27], [30], [31], [34], [37], [38], the following issues still remain in mapping optimization. (1) Neglecting the local structure of potential mapped netlists leads to inaccurate load estimations. (2) Insufficient consideration of the transition time at the input pin, however, is also a key factor in gate delay estimations. (3) The existing one-pass, delay-oriented matching approaches hinder the iterative refinement of mapping delays.

In this study, we address the challenges of mapping to minimize the overall circuit delay, and introduce a delay-driven iterative framework. The main contributions are as follows.

1) We first revisit the difficulties of the delay-driven mapping problem and explore three key insights via statistical analyses to address these.
2) By characterizing the local structures between logical networks and potential mapped netlists, we further design a structure-aware load-slew model that integrates input transitions and output loads for gate delay estimations.
3) To break the circular dependency in timing computation, we propose an iterative framework using Bayesian optimization and gradient descent to refine timing information within the model, progressively reducing circuit delays.
4) Experiments with $130\mathrm{nm}$ and $7\mathrm{nm}$ technology libraries show its superiority, it averagely reduces circuit delay of NLDM by 10% and 2% in area, and reduces 6% in delay after P&R, as compared to ABC [4].

The remainder of this paper is organized as follows: Section II provides the necessary background and the problem definition. Section III analyzes the key design insights of the framework. Section IV discusses a structure-aware load-slew model. Section V presents an iterative framework to fine-tune the model, further reducing circuit delays. Section VI presents experimental results and in-depth analyses, followed by the conclusion in Section VII.

II. Preliminaries and Problem Definition

In this section, we first present the background on different gate delay models, followed by an overview of the traditional mapping flow utilizing standard cells, and conclude with a description of the problem definition for technology mapping.

A. Gate Delay Model

Non-Linear Delay Model (NLDM). When using library files for ASIC technology mapping, the gate delay in the library is usually given in terms of lookup-tables, which is a non-linear function *w.r.t.* transition time and capacitive load driven by the gate. After mapping, the timing of the netlist can be efficiently analyzed by NLDM, which correlates these timing metrics with the variables of *input slew* and *output load*.

Unfortunately, it is impossible to precisely derive the actual input slew and output load before completing the mapping, due to the uncertain gate selections and their interconnections. Thus, we next examine two widely used gate delay models to estimate gate delays during the mapping [16], [19], [31].

Given a single-output gate (or cell) g, let $\delta(i,g)$ denote the delay from an input pin i to the output pin of gate g. The load c_g refers to the cumulative capacitance at the output of g. It is the sum of the input pin capacitances of all fanout pins and the loads of associated output wires.

Load-independent Delay Model (LIDM). The simplest gate delay model assumes that delay is unaffected by the capacitive load it drives within the network, *i.e.,* delay is only determined by the gate's intrinsic delay $\alpha(i,g)$ [16], [33], [34].

$$\delta(i,g) = \alpha(i,g) \quad = e \times \frac{c_o}{c_i} + PD \tag{1}$$

The load-independence property is well-considered in the gain-based timing model, which is widely used in synthesis tools, *e.g.,* ABC [4], [16]. As shown in Equation (1), in the abstraction of the gain-based method that incorporates the logical effort concept. The gate delay is estimated by modeling it as a linear function. This function estimates the delay by calculating the ratio of the gate's output to its input capacitance ($\frac{c_o}{c_i}$), employing a logical effort (e) to measure the gate's complexity, and accounting for the parasitic delay (PD) caused by the gate's internal capacitance. By providing a user-defined parameter *global gain* G to determine the target effort delay, with $\frac{c_o}{c_i} = \frac{G \times e_{inv}}{e}$, gate delay $\delta(i,g)$ is naturally independent of the load. Note that, the constant delay model [7] that assigns a fixed delay to all gates can be seen as a special case of LIDM.

Load-dependent Delay Model (LDDM). Taking into account the load-dependent coefficients in the gate delay model aligns more closely with the practical gate delay estimations [17], [18], [28], [31]. Typically, the delay $\delta(i,g)$ depends linearly on both the intrinsic delay and the output capacitance c_o.

$$\delta(i,g) = \beta(i,g) \times c_o + \alpha(i,g) \tag{2}$$

where $\beta(i,g)$ is the drive capability or load coefficient of the input pin i to the output pin of gate g. However, the actual load is unavailable in advance during the mapping process, and thus the number of logical network's fanouts or their arithmetic operations *e.g.,* $\mathrm{average}$, $\mathrm{nth\text{-}root}$ are employed to estimate the load coefficient [17], [19].

B. Standard Cell Mapping Based on Cuts

Boolean Network. Let $\mathcal{G}(\mathcal{V},\mathcal{E})$ represent a directed acyclic graph (DAG), where each node $n \in \mathcal{V}$ corresponds to a Boolean function and the edge $(x,y) \in \mathcal{G}$ corresponds to the wire connecting the nodes x and y. The terms *fanin* and *fanout* respectively refer to the incoming and outgoing edges of the node. An And-Inverter Graph (AIG) is a specific type of Boolean network that utilizes AND gates for its Boolean functions, with edges indicating whether signals are inverted.

***k*-feasible Cut.** A feasible cut C of the node $n \in \mathcal{V}$ of the AIG is a set of nodes in its transitive fanin such that every path from PIs to n passes through at least one node in C.

Thus, for a node n with two fanins x, y, the set of *k-feasible* cuts $\Phi(n)$ can be computed by combining the set of cuts of x and y [4]:

$$\Phi(n) = \{\{n\}\} \cup \{s \cup t | s \in \Phi(x), t \in \Phi(y), |s \cup t| \leq k\} \quad (3)$$

Mapping Flow. Among mapping solutions under LIDM [4], [23], the synthesis tool ABC [4] is widely recognized for its exceptional performance and efficiency in runtime. Specifically, given a Boolean network, *e.g.,* AIG, and the k-feasible cuts of their nodes, we review the three main steps of ABC after dividing into positive and negative phases as usual. *(1) Delay-oriented matching.* By topological order ABC first seeks the best match for each node in the AIG aiming to find a gate with the minimum arrival time. The minimum arrival time is computed based on LIDM by Equation (1), which takes the maximum among the summations of every input arrival time and its corresponding pin-to-pin delay. *(2) Area-oriented matching.* After one pass of delay-oriented mapping, it further performs local and global area-oriented mapping on timing non-critical nodes under LIDM. This recovery involves replacing the best match with a slower but smaller gate, aiming to optimize the area without impacting the delay under LIDM. *(3) Covering.* After the matching phases, the covering phase operates in a reverse topological order. It covers the unvisited nodes with the best match retained from the matching phase.

C. Problem Definition

The delay-driven mapping problem is defined as follows.

Problem. *Given a Boolean network represented as an And-Inverter Graph (AIG, $\mathcal{G}$), a library $\mathcal{L}$, the mapping objective for $\mathcal{G}$ is to find a mapped network (netlist) constructed using gates from $\mathcal{L}$ that minimizes the overall circuit delay. This objective is measured by employing a non-linear delay model (NLDM), which aims to reduce the delay across the circuit.*

Different from mappers essentially focusing on finding the optimal *depth* under the assumption of gate delay with LIDM [4], [16], [34], we touch on the problem that directly orients to minimize the overall circuit delay.

III. Analyses of Framework Design

We first highlight the significance of the gate model in delay-driven mapping. Next, we explore three key insights in our iterative framework design facilitated by the gate model.

A. Analysis of Delay Models Coupled with Graph Structures

Based on our problem definition, developing a precise gate delay model is essential, as it serves as a bridge between gate selections during mapping and timing analysis of the mapped network. In dynamic programming methods, gate selections during the matching and covering phases are fundamentally guided by the delay estimations from gate delay models, *i.e.,* each gate's pin-to-pin delay $\delta(i, g)$, as shown in Equation (4).

$$\begin{aligned} A(n) &= \min_{M}\{A(n, M)\} \\ A(n, M) &= \max_{i}\{A(i) + \delta(i, g)\} \end{aligned} \quad (4)$$

Here, $A(n, M)$ represents the maximum arrival time at node n for a given match (or gate) M, which is calculated as the maximum of the sum of the arrival time $A(i)$ of each input i and their corresponding pin-to-pin delays $\delta(i, g)$.

That is, $\delta(i, g)$ serves as the cost function in the dynamic programming approach, directing its gate choice. On the one hand, $\delta(i, g)$ serves as an approximation of gate delay under unknown actual input slew and output load conditions, *e.g.,* using LIDM and LDDM [4], [17], [23] to estimate $\delta(i, g)$. On the other hand, the structure of the Boolean network *potentially reveals the behaviors* of input slew and output load, which are informative for gate delay estimations.

B. Analysis of Design Insights

Building on the emphasis placed on the gate model earlier, we perform a collaborative analysis with graph structures to enhance the accuracy of gate model $\delta(i, g)$. Three key insights from practical examples are yielded as shown in the next.

Insight 1: Inaccurate Load Estimation. Integrating output load into the gate model is widely recognized for enhancing gate delay accuracy [16], [17], [34]. However, accounting for the gate's unknown output load during mapping presents a significant challenge. To address this issue, the heuristic method approximates the load by counting the fanouts observed in the mapped netlist. The number of fanouts in the netlist generally indicates the trend of delay, as each gate output is typically connected to the inputs of multiple other gates, and each connection increases the capacitive load at the output node. Unfortunately, during mapping, even the netlist fanouts are not available. Hence it is a common practice[1] to use the number of fanouts observed in the Boolean network, *e.g.,* AIG to approximate the number of netlist fanouts. For example, Sentovich [31] directly uses the number of AIG's fanouts as the load coefficient, or using the arithmetic operations (average, nth-root, log) of the number of AIG's fanouts [19], [23]. These existing methods for load estimation involve too many simplifications and lead to inaccuracies due to insufficient structural information from the Boolean network.

Given that the fanout numbers in the netlist directly impact the estimation of gate delays. To understand how the number of fanouts of AIG affects the number of fanouts in the netlist, we conduct a statistical analysis by comparing fanout distributions in both AIG and netlists. It uses sin circuit of EPFL benchmark [2] and ABC [4], as shown in Fig. 2.

From Fig. 2, we have the following findings. (1) Typically, fanout numbers in a netlist are directly proportional to the fanout numbers at AIG nodes. It thus validates estimating netlist fanouts through AIG fanouts [4]. (2) The fanout numbers in the netlist generally fall within the range observed at AIG nodes. Notably, over 50% of AIG nodes with more than 4 fanouts see a reduction in fanout numbers after mapping to the netlist. This reduction is primarily due to nodes with larger fanout numbers being more likely to serve as internal

[1]Estimating load using the cut reference number for nodes remains challenging. One reason is that the cut reference number is often very large, leading to inaccurate gate delay estimates. For example, in the sin circuit, 38% of nodes have a cut reference number greater than 100.

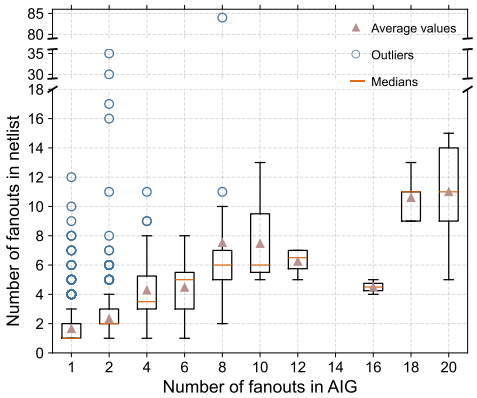

Fig. 2 Analysis Example: distribution of the number of fanouts in AIG and netlist in sin circuit.

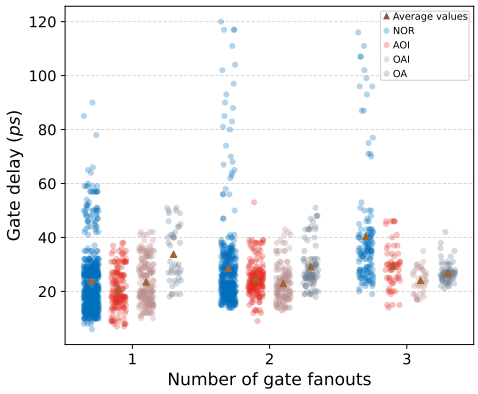

Fig. 3 Analysis Example: distribution of gate delays and the number of fanouts under different gates in sin circuit.

nodes of the *cut*. (3) There are exceptional cases where the fanout numbers of certain nodes significantly exceed their AIG counterparts (marked as outliers in Fig. 2). This significant increase in fanout numbers is primarily due to the *node's successors* using it as an input pin.

Although these outliers represent a small proportion, their substantially larger fanout numbers lead to increased load coefficients, significantly elevating gate delays. Moreover, such nodes are more likely to appear on the critical path. Take one of the outlines in Fig. 2 as an example, consider an AIG node with 2 fanouts that maps to a netlist node with 30 fanouts, which is specifically an OR-AND gate. This gate is precisely located on the critical path of the mapped sin netlist. Conversely, if the mappers are unable to recognize these nodes that could have large fanouts in the netlist, it could result in situations like those highlighted in the motivation example (in Fig. 1), in which there are significant gaps between estimated and actual delays.

Insight 2: Insufficiency of Load-only Coupled. Although integrating the output load into the gate model can improve the accuracy of gate delay estimation, considering only the load factor in the gate delay model is still insufficient.

To study the impact of fanout numbers in the netlist on gate delay, we conduct a statistical analysis by comparing gate delays and fanout numbers across various gates using sin circuit [2] and ABC [4], as illustrated in Fig. 3.

From Fig. 3, we have the following findings. (1) Gate delay is typically positively correlated with gate fanout numbers. This is mainly due to the increase in the load coefficient. (2) Gate delay cannot be considered a linear function of the gate fanout numbers, causing significant variations. This variability stems from the fact that gate output fanout only models the load on the output pin and does not linearly estimate the slew on the input pin. For example, even with precise fanout estimation from the AIG, the mean absolute error (MAE) for NOR gates with a fanout of one can be as high as 8.66. This occurs even though the delays for these gates vary from 6.1 to 89.5 ps, averaging 23.9 ps. Besides, as gate matching progresses, the inaccuracies inherent in a load-only coupled gate delay model accumulate. This could further lead to significant discrepancies between estimated and actual delays, as shown in Fig. 1.

Insight 3: Circular Dependency in Delay Computation. During the mapping workflow, there is a circular dependency in delay (timing) computation during the matching and covering phases. Specifically, (1) in the matching phase, gate selection requires timing information such as gate delay and arrival time, which are contingent upon the load and slew. However, they are not determined until the subsequent covering phase. (2) In the covering phase, it utilizes the arrival time computed in the matching phase to make more precise covering decisions, for minimizing the overall circuit delay. Consequently, this circular dependency creates a classic chicken-and-egg problem, complicating the mapping process. Besides, inaccuracies in arrival times could lead to suboptimal gate selections in the covering phase, thereby limiting the efforts to minimize circuit delay, as highlighted in Fig. 1.

From the analyses, the circular dependency must be broken in some manner, and we have the following findings. (1) As analyzed in insights 1 and 2, these strategies are also regarded as ways to break the circular dependency. The key idea is to accurately predict potential delays early in the matching phase. This can be achieved by utilizing the AIG's structure or by incorporating slew information from the library into the gate delay model. (2) Besides, another promising way involves employing a more refined iterative process, alternating optimizing between the matching and covering phases. That is, utilizing improved timing information, this way finetunes delay computation in the matching phase iteration, progressively converging towards a minimized circuit delay.

IV. Structure-aware Load-slew Model

Based on the analyses, we first present the load-induced gate delay by characterizing the graph structure (inspired by Insight 1), then discuss the structure-aware slew-induced delay (inspired by Insight 2), followed by the completed gate model.

A. Structure-aware Load-induced Delay Estimation

Based on the analyses from Insight 1, specifying the potential fanout numbers in the mapped netlist using the AIG's graph structure could significantly enhance the accuracy of load estimation. Therefore, we first characterize the potential local structures within the AIG that may be mapped, and then estimate load-induced delay based on this local structure.

1) Local Structure Characterization: We first utilize the potential fanouts of node n to characterize the local structure that node n may map into within the netlist.

Definition 1 (Potential Fanout)**.** *Given a node in an AIG, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the potential fanout $\hat{n}$ of a node $n \in \mathcal{V}$ is defined as a match $\hat{m}$ located at $\hat{n}$ within the transitive fanout cone of n, where n is included in the set of inputs for $\hat{m}$.*

Taking an example in Fig. 4, the nodes 6 and 9 are the potential fanouts of the node 5, as 5 is one of the inputs of matches m_1 and m_2.

Although the transitive fanout cone could characterize potential fanouts, directly using its node number[2] to estimate a node's potential load is problematic due to two main issues: (1) *Redundancy in Potential Matches:* The transitive fanout cone often includes redundant potential matches, leading to overestimated load. For example, as shown in Fig. 4, considering match m_2 as a potential match for node 5 inherently includes node 6 and its corresponding match m_1. (2) *Loss of Dynamic Load Estimation:* Estimating load by counting all nodes within the cone may overlook dynamic variations in actual load. For example, although node 5 consistently has a potential node number of 8, there is a 50% chance that the actual fanout numbers in the netlist are fewer than 2, by Fig. 2.

To address these issues, we define τ-order fanouts and their tiers within the potential fanouts to more accurately estimate the load coefficient.

Definition 2 (τ-order Fanouts)**.** *The τ-order fanouts of a node v in an AIG are defined as all of the nodes that are reachable from n via paths that consist exactly of τ edges.*

This definition is intended to prevent the mixing of matches from different fanout orders, thereby reducing the impact of redundancy in potential matches on the load factor. As depicted in Fig. 4, nodes 6 and 9 are represented as the 1-order and 2-order fanouts of node 5, respectively. Their corresponding matches, m_1 and m_2, are thus distinctly categorized.

Based on the concept of τ-order fanouts, to discern the number of fanouts across different orders, we have defined the τ-order fanout tiers $F^{\tau}(\cdot)$.

Definition 3 (τ-order Fanout Tiers)**.** *The τ-order fanout tiers of a node n denoted as $F^{\tau}(n)$, is an array consisting of fanout counts at various orders for a given node n, i.e., $[F_0^{\tau}(n), \cdots, F_i^{\tau}(n), \cdots, F_{\tau}^{\tau}(n)]$*

Without loss of generality, we define 0-order fanout number $F_0^{\tau}(n) = 1$, where 0-order fanouts represent node n itself. Utilizing this definition, the tiered τ-order fanouts through levelization offer a dynamic view of load variations. Lower-order fanout tiers in $F^{\tau}(\cdot)$ are more likely to correspond to the actual fanout numbers in the netlist. The higher-order tiers facilitate the identification of nodes in the netlist with significantly larger fanouts, *i.e.,* outliers in Fig. 2.

Based on the τ-order fanout tiers' definition, we have developed an algorithm based on dynamic programming to compute

[2]If we assume all cuts can be implemented by gates, then the number of nodes in potential fanouts of node n represents n's cut references.

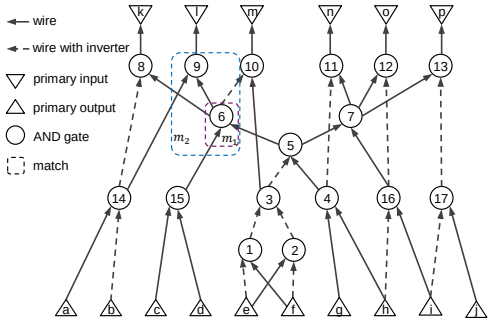

Fig. 4 Example of an AIG's local structure characterization. m_1 and m_2 are two matches of nodes 6 and 9 implementing of the function of cuts $\{5, 15\}$ and $\{5, 14, 15\}$, respectively.

the tier of each node, $F^{\tau}(\cdot)$, as illustrated in Algorithm 1. It takes a Boolean network, *i.e.,* AIG, as input and outputs the tiers of each node $F^{\tau}(\cdot)$. Specifically, (1) the 0-order fanout number $F_0^{\tau}(\cdot)$ for each node is initialized to 1, and 1-order fanout number $F_1^{\tau}(\cdot)$ is set to the actual number of direct fanouts of the node. Besides, for all primary output nodes, the fanout counts for all orders from 0 to τ are initialized to 0 (lines 3, 4). (2) For each order j from 2 to τ, it updates the fanout number for each node n by adding the $j-1$-order fanout numbers of each fanout node m of n (lines 5–11).

An Illustrative Example: In the AIG shown in Fig. 4, the 1-order fanout tier of node 5 is $[1, 2]$, as it reaches nodes $\{6, 7\}$ via one edge. The 2-order fanout tier of node 5 is $[1, 2, 6]$, as it reaches nodes $\{8, 9, 10, 11, 12, 13\}$ via two edges.

We next establish the relationship between τ-order fanouts and their potential fanouts.

Theorem 1. *Given that all k-feasible cuts in the AIG can be implemented by a gate, for any τ such that $\tau < k$, all τ-order fanouts are considered potential fanouts.*

Proof. Given that all k-feasible cuts can be implemented by a gate, that is, each k-feasible cut has a corresponding match *w.r.t.* the gate. Therefore, we prove the theorem from: for any node n and its τ-order fanouts $\hat{n}$, there exists a k-feasible cut wherein n is one of its leaf nodes, under $\tau < k$.

(1) By the definition of τ-order fanouts, there exists a path in the AIG from n to $\hat{n}$ consisting *exactly* of τ edges, represented as $n \to n_1 \to \ldots \to n_i \to \ldots \to n_{\tau-1} \to \hat{n}$. Here, each node is an AND gate, with n_i being one of the inputs to n_{i+1}. (2) Assume that the other inputs to these AND gates $\{n_1, \ldots, n_i, \ldots, n_{\tau-1}, \hat{n}\}$ are respectively $\{n_1', \ldots, n_i', \ldots, n_{\tau-1}', \hat{n}'\}$. (3) Since any path from the primary inputs (PIs) to $\hat{n}$ must pass through at least one node in the set $\phi = \{n, n_1', \ldots, n_i', \ldots, n_{\tau-1}', \hat{n}'\}$. By the definition of k-feasible cuts, the set ϕ forms a cut for node $\hat{n}$, where n is one of its leaf nodes. (4) Given that the set ϕ contains $\tau + 1$ nodes, and the set ϕ constitutes a k-feasible cut for $\hat{n}$, with n as one of the leaves. Note that, $\tau + 1 = k$, *i.e.,* we have $\tau < k$. □

From these, τ-order fanouts are potentially bounded by k-feasible cuts when $\tau < k$, *i.e.,* all τ-order fanouts are considered potential fanouts. In practice, we set $k = 5$ to compute all k-feasible cuts and $\tau = 3$ to determine the τ-order fanout tiers. This strategy has the following features: (1) This

local structure characterization helps reduce redundancy in the transitive fanout cone, thereby enabling a more accurate estimation of the load coefficient. (2) It further enables dynamic measurement of the varying effects on the load exerted by fanouts of different orders. (3) τ-order fanout tiers also avoid generating candidate matches for all possible loads [16], [17], [19], and reduce the pool of potential best matches.

2) Load-Induced Delay Estimation Using Local Structure: We further utilize the τ-order fanout tiers to estimate the contribution of load to gate delay $Ld(\cdot, g)$.

The proposed τ-order fanout tiers can "observe" the potential fanouts in the netlist, facilitating the estimation of potential driving loads in the AIG in advance. Additionally, to account for possible gate selections, we introduce a parameterized approach to dynamically estimate the impact of different tiers on gate delay. Therefore, inspired by Equation (1) and Equation (2), the load-induced delay of the i-th pin-to-pin in gate g, $Ld(i, g)$ is modeled as a linear function as follows:

$$Ld(i,g) = \kappa^L \times LD \times \sum_{j=0}^{\tau} \beta_j^L \times F_j^\tau(n) + PD^L \times \upsilon^L \quad (5)$$

where LD represents the load-dependent coefficient, quantifying the induced delay per unit load driven by the gate, PD^L denotes the intrinsic delay of the gate caused by its internal load variations, $F_j^\tau(n)$ refers to the τ-order fanout tiers for the root node n which approximate the potential load on gate g. The parameters κ^L, β_j^L, and υ^L are scaling factors that respectively adjust the contributions of LD, $F_j^\tau(n)$, and PD^L to the delay.

Specifically, by Equation (1), LD is calculated using e/c_i, which represents the incremental delay per unit of input capacitance for a logic gate. Since the actual load is unknown, κ^L enables flexible adjustment of LD according to the circuit's local fanout structure. This further aligns the circuit's behavior with empirical data or specific technology characteristics. For j-th tier among τ-order fanouts, $\beta_j^L \times F_j^\tau(n)$ adjust the contribution of different fanout tiers to the overall load. Each tier j from 0 to τ has its own factor that scales its impact on the delay, reflecting the possibility of different tiers being mapped. For example, when $\beta_2^L > \beta_1^L$, then node 5 in Fig. 4 could choose m_2 as the match. Thus, $\sum_{j=0}^{\tau} \beta_j^L \times F_j^\tau(n)$ represents the output capacitance c_o driven by gate g. PD^L reflects the zero-load delay for a given slew, while υ^L adjusts for different slew conditions. $PD^L \times \upsilon^L$ fine-tunes the parasitic delay, ensuring the estimated delay matches practical measurements.

Thus, unlike previous methods that approximate load using fixed fanouts [17], [19], [23], our method dynamically accounts for how a gate's output affects distant regions of the netlist by characterizing AIG local structures prior to mapping.

B. Structure-aware Slew-induced Delay Estimation

Based on the analyses from Insight 2, while characterizing local structures enables accurate estimation of load-induced delays, relying solely on the load factor in the gate delay model is insufficient. It is also essential to account for the impact of slew (transition time) at the input pin on delay.

Algorithm 1 Compute τ-order Fanout Tiers

Input: Boolean network, *i.e.,* AIG.
Output: The τ-order fanouts tiers of each node $F^\tau(\cdot)$.

```
Init F_0^τ(·) and F_1^τ(·) to 1 and its fanout count, respectively;
Init F_j^τ(·) to 0 for all primary outputs for j from 0 to τ;
for order j from 2 to τ do
    for all node n in AIG do
        for all fanout node m of n do
            F_j^τ(n) += F_{j-1}^τ(m);
        end for
    end for
end for
```

The gate delay, $\delta(i, g)$, represents the delay from the input pin i to the output pin of gate g. It depends not only on the output load c_o of gate g but also on the transition time s_i at input pin i. The transition time at the input pin i, in turn, is influenced by the gate $\tilde{g}_i$ driving the pin i and the total capacitive load driven by $\tilde{g}_i$, as illustrated in Fig. 5. The gate delay induced by load c_o, $Ld(i, g)$, is well estimated by τ-order fanout tiers in Section IV-A. We next estimate the slew-induced delay $Sd(\cdot, g)$ by the local structures of $\tilde{g}_i$.

The transition time s_i at input pin i, is also the output transition time of gate $\tilde{g}_i$. In fact, s_i is a non-linear function of the input slew of $\tilde{g}_i$ and its output load. Our objective here is not to precisely estimate s_i, but rather to focus on the delay induced by the slew at input pin i. Consequently, we model the slew-induced delay $Sd(i, g)$ from the perspective of the load driven by gate $\tilde{g}_i$. Thus, inspired by Equation (1) and the characterized local structure, the slew-induced delay for the i-th pin-to-pin in gate g, $Sd(i, g)$, is expressed as follows:

$$Sd(i,g) = \kappa^S \times SD \times \sum_{j=0}^{\tau} \beta_j^S \times F_j^\tau(i) + PD^S \times \upsilon^S \quad (6)$$

Slew-induced delay, $Sd(i, g)$, also uses a linear model, specifically focusing on the input characteristics of the gate $\tilde{g}_i$.

Here, SD represents the slew-dependent coefficient, quantifying the incremental delay per unit of slew rate caused by gate $\tilde{g}_i$. PD^S denotes the intrinsic parasitic delay related to the slew characteristics of gate $\tilde{g}_i$. $F_j^\tau(i)$ refers to the τ-order fanout tiers for the pin node i. By adjusting the contribution of different fanout tiers to the overall slew, $\sum_{j=0}^{\tau} \beta_j^S \times F_j^\tau(i)$ measures the possible slew for input pin i. The parameters κ^S, β_j^S, and υ^S serve as scaling factors, similarly adjusting the contributions of SD, the input pin i fanout structure, and zero-slew delay PD^S to the overall slew-induced delay. In practice, we select the worst-case values of SD and PD^S from all pin-to-pin of gate $\tilde{g}_i$, due to the inability to determine in advance which pin in $\tilde{g}_i$ contributes the most significantly to the slew-induced delay.

Thus, by characterizing the local structures of gate $\tilde{g}_i$ driving input pin i of g, and aligning its slew characteristics, $Sd(i, g)$ offers an accurate estimation of slew-induced delay.

C. The Completed Gate Model

Based on the analyses above, combining the load-induced $Ld(i,g)$ and slew-induced $Sd(i,g)$ delays, we have derived the structure-aware load-slew model of gate g. It is a bi-linear function *w.r.t.* the estimated output load and input slew:

$$\delta(i,g) = \alpha \times Ld(i,g) + (1-\alpha) \times Sd(i,g) \tag{7}$$

where α is a weighting factor that balances the contribution of load-induced and slew-induced delays in the model. Fig. 5 illustrate these two parts of delay.

In practical technology libraries, the rising and falling of gate delay and transitions are typically characterized by different values. Thus, the pin-to-pin delay from input pin i to the output of gate g, $\delta(i,g)$, is specified by a tuple of four real numbers:

$$\delta(i,g) = \begin{cases} \delta(i,g)^{RR} \\ \delta(i,g)^{RF} \\ \delta(i,g)^{FR} \\ \delta(i,g)^{FF} \end{cases} \tag{8}$$

For instance, $\delta(i,g)^{RR}$ is the delay from the **r**ising edge at i to the **r**ising edge at the output of g. To handle this issue, four technology-dependent delay parameters, *i.e.,* LD, PD^L, SD, and PD^S, are extended into a tuple of four real numbers to accommodate distinct rising and falling values. For example, given a pin i of gate g, its load-dependent coefficient $LD = \{LD^{RR}, LD^{RF}, LD^{FR}, LD^{FF}\}$, each component addressing specific scenarios between input and output pins.

To construct technology-dependent delay parameters from the library, we calculate LD and PD^L of pin i at an average slew from the given lookup table. Specifically, LD is computed as $(d_1 - d_2)/((l_1 - l_2)/i_{\text{cap}})$, where the delay difference $(d_1 - d_2)$ between two different loads $(l_1 - l_2)$ is divided by the change in capacitance (i_{cap}). This regularization of load difference by the pin capacitance i_{cap} is done to standardize gates of different sizes, facilitating postmapping optimizations such as gate sizing. Given the average slew, PD^L represents the delay when the load capacitance is zero, capturing the intrinsic delay. Similarly, SD quantifies the contribution of unit slew to delay at an average load from the lookup table, while PD^S is the intrinsic gate delay when the slew is zero. Note that, since the actual transitions are unknown during the mapping process, the arrival time in Equation (4) is calculated based on the worst-case delay under the different rising and falling delays. Besides, during the mapping process, our framework utilizes Equation (7) for the different gate sizes, like the gain-based model in ABC.

In summary, although the actual input slew and output load are unknown prior to mapping, our structure-aware load-slew model addresses these challenges with several key advantages. (1) By analyzing AIG's local structures, it aligns potential mapping areas between AIG and netlist. These areas, (τ-order fanout tiers), enhance slew and load impact assessments. (2) The model approximates gate delay from both slew and load perspectives, inherently integrating network's structures and library characteristics, thereby bringing it closer to NLDM. (3) A parameterized gate model adapts to potential mapping choices, enhancing the model's flexibility. Thus, by replacing our gate model in Equation (4), the technology mapper can more effectively guide gate selection to minimize circuit delay.

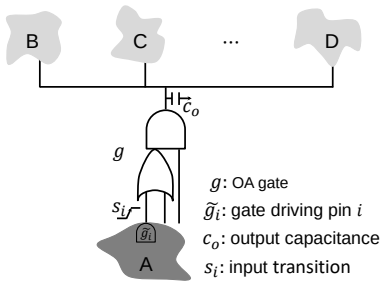

Fig. 5 Illustration of the structure-aware load-slew model computation. The OR-AND gate g is a possible matching implementing cut $\{1,2,4\}$ of node 5 depicted in Fig. 4. Region A represents the matched subgraph, while regions B, C, and D are candidate matchings potentially driven by gate g.

V. The Iterative Framework

Based on the analyses from Insight 3, the cyclical dependency of timing information essentially exists in the matching and covering phases. To break the cycle, (1) our structure-aware load-slew model has been developed, which enables gate delay estimations without actual covering, as detailed in Section IV. (2) In this section, we introduce an iterative process that utilizes timing information refined after each covering iteration to guide the gate delay estimation during the matching phase.

A. Framework Overview

In our parameterized structure-aware load-slew model, as illustrated in Equation (7), we group the parameters $\{\alpha, \kappa^L, \beta_j^L, \upsilon^L, \kappa^S, \beta_j^S, \upsilon^S\}$ into a set of *global parameters* θ, representing the impact of various factors on gate delay. For example, α balances the effects of load and slew on delay, while κ^L and κ^S adjust load- and slew-dependent delays to align with empirical data or technology characteristics. β_j^L and β_j^S capture the contributions of local AIG structures to load- and slew-induced delays, and υ^L and υ^S fine-tune parasitic delays related to load and slew. By fine-tuning these parameters to align the library characteristic with the circuit structure, thereby deriving an accurate gate delay model. Besides, the τ-order fanout tiers $F^\tau(n)$ for each node n quantify the potential fanouts that each node may map but also serve as *local parameters* in the gate delay model, capturing load variations at a fine granularity. Within the iterative framework, actual gate delays from previous mapping iterations are used to refine the model, enabling fine-tuned adjustments to the local parameters This process, in turn, minimizes overall circuit delay, ensuring the model aligns with practical performance outcomes.

The overall framework of our delay-driven technology mapper is illustrated in Algorithm 2, building upon the structure-aware load-slew delay model. Specifically, inspired by Bayesian optimization (BO) techniques [12], [13], [15], [32], the framework employs a gradient-free method to optimize global parameters $\theta = \{\alpha, \kappa^L, \beta_j^L, \upsilon^L, \kappa^S, \beta_j^S, \upsilon^S\}$ (lines 1–16, illustrated in Section V-B). It iteratively performs technology mapping to progressively identify global parameters that achieve improved Quality of Results (QoR).

Once a superior QoR is identified, the framework further optimizes local parameters $F^{\tau}(n)$ using the actual gate delays under the improved global parameters via gradient descent (lines 11–15, illustrated in Section V-C). Benefiting from the optimized parameters θ^*, and $F^{\tau}(n)^*$, it guides the candidate gate selection using the load-slew gate model, followed by covering the best matches (lines 17–18).

B. Bayesian Optimization for Global Parameters

Owing to the sample efficiency and the balanced exploration and exploitation capabilities [12], [29], Bayesian optimization has been widely used in logic synthesis flow [12], [15], [35]. Consequently, we have incorporated BO into our iterative framework to leverage these advantages.

BO is a method for iteratively addressing global optimization problems, featuring two key components, *i.e.,* a surrogate model and an acquisition function. In our framework, in each iteration t, the surrogate model selects an input θ for evaluation and acquires a corresponding black-box function value $\boldsymbol{y}$. We minimize the overall circuit delay by sequentially selecting inputs θ and evaluating their QoR as the objective function. The objective function is detailed as follows.

$$\boldsymbol{y}(t) \;=\; \frac{\text{delay}(t)}{\text{delay}_{exp}} + \frac{\text{area}(t)}{\text{area}_{exp}} \tag{9}$$

where $\text{delay}(t)$ and $\text{area}(t)$ represent the delay and area from the t-th iteration of technology mapping, respectively. The terms delay_{exp} and area_{exp} denote the minimum delay and area observed during mapping process under the initial n_0 sets of expert parameters. Among the iterative process, we select the global parameters yielding the smallest $\boldsymbol{y}(t)$ as the result of BO. Note that, although our load-slew gate model targets to minimize the mapped circuit delay, area is also considered in the objective function. This is because excessively optimizing for delay can lead to compromises in the area [23].

Before BO iterations, we do not rely on random sampling to collect initial data points, as it requires $10^2 \sim 10^3$ iterations to converge [15]. In practice, by leveraging the specific characteristics of Equation (7), we provide well-informed initial guesses for the parameters, which can potentially reduce the number of iterations and enhance the efficiency of the framework. For example, we have pre-designed three sets of expert parameters, *i.e.,* $n_0 = 3$, where α is set to [0.2, 0.5, 0.8] to reflect the impacts of estimated gate slew and load on circuit delay, β_2^L is set to [0.08, 0.02, 0.01] to account for the contributions of higher-order fanout effects on load, as the 2-order fanout numbers of AIGs are typically larger than 10.

Surrogate Model. After collecting a set of n_0 expert parameters and their corresponding objective values $\boldsymbol{y}(t)$ (line 4 in Algorithm 2), a surrogate model is fitted to these evaluations (line 7). The model predicts the outputs of the black-box function across the input space by providing quantified estimates of prediction uncertainty. To accommodate behaviors under unexplored inputs with flexibility and sample efficiency, various surrogate models have been proposed for setting priors over unknown functions [12], [15], *e.g.,* Gaussian processes, random forests, and deep ensembles. In our delay-driven iterative framework, we opt for random forests as the surrogate model to mitigate the impact of outliers on the parameter.

Algorithm 2 Delay-Driven Iterative Framework

Input: AIG $\mathcal{G}$, library $\mathcal{L}$, global iteration number T.
Output: Mapped netlist implemented by $\mathcal{L}$.

1: Extract LD, PD^L, SD, and PD^S from $\mathcal{L}$;
2: Compute τ-order fanout tiers using Algorithm 1;
3: Init n_0 sets of expert parameters $\{\theta_0, \theta_1, \cdots \theta_{n_0}\}$ for gate model $\delta(i,g)$ with $\theta = \{\alpha, \kappa^L, \beta_j^L, \upsilon^L, \kappa^S, \beta_j^S, \upsilon^S\}$;
4: Collect data $\mathcal{D}_0 = \{(\theta_0, \boldsymbol{y_0}), (\theta_1, \boldsymbol{y_1}), \cdots, (\theta_{n_0}, \boldsymbol{y_{n_0}})\}$;
5: Set $\boldsymbol{y_{min}}$ to $\min(\boldsymbol{y_0}, \boldsymbol{y_0}, \cdots, \boldsymbol{y_{n_0}})$;
6: **for** t from 1 to T **do** ▷ Global parameter optimization
7: Fit surrogate model using $\mathcal{D}_{t-1}$;
8: Determine θ_{new} by maximizing an acquisition function;
9: Evaluate $\boldsymbol{y_{new}}$ by perform mapping guided by $\delta(i,g)$ with parameters θ_{new};
10: Update $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(\theta_{new}, \boldsymbol{y_{new}})\}$;
11: **if** $\boldsymbol{y_{min}} > \boldsymbol{y_{new}}$ **then** ▷ Local parameter optimization
12: Set $\boldsymbol{y_{new}}$ to $\boldsymbol{y_{min}}$;
13: Obtain the actual delay of gates $\mathcal{M}$ from NLDM;
14: Update $F^{\tau}(n)$ for each n in $\mathcal{M}$ using actual delay and $\delta(i,g)$ with θ_{new} via gradient descent;
15: **end if**
16: **end for**
17: Guide candidate matches using the load-slew gate model with the optimal parameters θ^* and $F^{\tau}(n)^*$.
18: Cover the found best matches.

Acquisition Function. Based on the posterior distribution fitted by the surrogate model, the acquisition function determines the new global parameters θ_{new} (line 8). Due to the effectiveness of the lower confidence bound (LCB) strategy in logic synthesis [12], we also use LCB as the parameter search strategy. LCB adjusts the uncertainty by incorporating a tuning parameter, which modifies the emphasis placed on the predictive value. Thus, the adjustment allows the strategy to balance exploration and exploitation in the search process.

C. Gradient Descent for Local Parameters

We globally align library characteristics with the circuit structure by tuning global parameters in Section V-B. Actually, in our gate model, we estimate potential netlist structures by analyzing τ-order fanout tiers $F^{\tau}(n)$, approached from the perspective of the Boolean network. Among the iterative framework, the potential netlist structures are concretized, *i.e.,* allowing us to obtain the actual mapped netlists. Consequently, these actual netlist structures enable fine-tuning of local parameters, supporting more precise optimization of potential mapping structures, from a fine-grained perspective of $F^{\tau}(n)$.

Our gate model, while explicitly formulated, does not yield an analytical solution for $F^{\tau}(n)$ with a specific set of global parameters. This limitation stems from the iterative and complex nature of the actual netlist structures, where

each gate may exhibit multiple real values. In practice, as the objective function $\boldsymbol{y}(t)$ shows improvement, the evolving netlists naturally direct the optimization towards more efficient mapped netlists, specifically guiding adjustment in $F^{\tau}(n)$. Therefore, we utilize gradient descent techniques to incrementally refine the mapping netlist structures of individual gates. This method allows us to reduce errors of gate delay estimation progressively and enhance the overall QoR of the mapped netlist.

From Algorithm 2, once BO identifies global parameters that yield an improved objective $\boldsymbol{y}(t)$, this indicates that the library characteristics and the circuit structure are well aligned under these parameters. Benefiting from the actual gate delays of the mapped netlist, we further refine the estimation of potential netlist fanout numbers $F^{\tau}(n)$ (lines 11–15). After obtaining the actual gate delays $\overline{\delta(i,g)}$ via NLDM, we compute the distance (gap) between our structure-aware load-slew model and the observed delays. Note that, instead of directly obtaining the NLDM values through the parameters in Equation (5) and Equation (6), we use feedback from the actual gate delay to refine our gate model. The distance is a function *w.r.t.* τ-order fanout tiers on the input pin i and root node n, is defined as:

$$f\left(F^{\tau}\left(i\right),F^{\tau}\left(n\right)\right)=\left(\overline{\delta(i,g)}-\delta(i,g)\right)^{2} \tag{10}$$

This distance highlights the gap between our estimated gate delay and the actual delay, providing direction for optimizing our load-slew model. Thus, we employ gradient descent to bring our load-slew model and the observed delays closer.

Specifically, to refine the τ-order fanout of each node, we first calculate the gradient of the j-th order fanout tiers of the root node n Equation (11). This gradient represents the sensitivity of the distance function $f(\cdot)$ with respect to $F_j^{\tau}(n)$. Similarly, for the current gate g, the gradient $\nabla F_j^{\tau}(i)$ of the input pin node i is also computed with j from 0 to τ.

$$\nabla F_j^{\tau}(n)=\frac{\partial f\left(F^{\tau}\left(i\right),F^{\tau}\left(n\right)\right)}{\partial F_j^{\tau}\left(n\right)} \tag{11}$$

Based on the calculated gradient $\nabla F_j^{\tau}(n)$, we procee with the local parameter update step to refine the potential fanout estimations. The update rule for the j-th order fanout tiers of the root node n at iteration $k+1$ is given by:

$$F_j^{\tau}(n)_{k+1}=F_j^{\tau}(n)_k-\eta\nabla F_j^{\tau}(n)_k \tag{12}$$

Here, η represents the learning rate, a positive scalar that determines the step size of the gradient descent. $F_j^{\tau}(i)$ is similarly updated. The update formula essentially adjusts $F_j^{\tau}(n)$ and $F_j^{\tau}(i)$ in the direction that most reduce the distance function $f(\cdot)$, thereby incrementally improving the accuracy of our model's delay estimation from a fine-grained perspective.

In practice, to mitigate the inherent disturbances in gate delays caused by unpredictable factors in the mapping process (*e.g.,* inverter insertions), we conservatively design the conditions for triggering local parameter optimization. For example, gradient descent is invoked when $|\overline{\delta(i,g)}-\delta(i,g)|>10$, in mult circuit [2], this results in 2 times local parameter updates for $T=10$, each affecting no more than 20% of the nodes.

Benefiting from the BO for global and gradient descent for local parameters updates, we have obtained the optimized parameters θ^{*}, and $F^{\tau}(n)^{*}$. Based on these, the framework further performs candidate gate selection to find the best matches, followed by the cover phase (lines 17–18).

VI. Experimental Results

A. Experimental Settings

We have implemented the iterative framework within the logic synthesis tool ABC [4], with the integration of BO tool HEBO [11]. When evaluating the delay after Place and Route (P&R), the framework is integrated into the physical toolkit iEDA [21]. To evaluate the performance of our framework, we have compared it with the traditional ASIC mapper ABC [4] and a learning-based delay prediction mapper AiMap [23]. For each comparative method, we perform buffering and sizing optimizations after the technology mapping, and compare the postmapping QoR. Specifically, the synthesis flow we use in ABC is as follows: `strash; map; topo; buffer; upsize; dnsize; stime`.

In our framework, we set global parameters as follows: α, κ^{L}, κ^{S} within $[0,1]$; υ^{S}, υ^{L} from $[0.5,2]$; β_0^{S}, β_0^{L} at 1; β_1^{S}, β_1^{L} within $[0,0.5]$; β_2^{S}, β_2^{L} from $[0,0.1]$; and β_3^{S}, β_3^{L} from $[0,0.02]$. The global iteration number T and the order τ are set to 10 and 3, respectively. For the given set of expert-designed parameters of θ is set to $\{0.26,0.68,\{1,0.11,0.04,0.0\},0.65,0.98,\{1,0.02,0.01,0.0\},1.42\}$. For the gain-based model in ABC, the default gain is set to 250, *i.e.,* `map -G 250`. All experiments were conducted on an Intel(R) Xeon(R) Platinum 8260 CPU at 2.40GHz, featuring 24 cores and 128GB of RAM.

For the datasets, we have performed evaluations using 10 arithmetic and 10 control circuits of EPFL combinational benchmarks [2] and 2 larger circuits from IWLS 2005 [1] and OpenABC-D [9]. These circuits vary widely in complexity, with node counts ranging from 174 to 802,919. To assess the mapping QoR, these circuits have been evaluated using two different standard cell libraries: the open-source ASAP $7\mathrm{nm}$ PDK [10] and SkyWater $130\mathrm{nm}$ PDK [14].

B. QoR of our Mapping Framework

Exp-1: Evaluation of Mapping Results. To assess the effectiveness of our framework relative to ABC [4] and AiMap [23], we evaluate their QoR across ASAP $7\mathrm{nm}$ and Sky $130\mathrm{nm}$ technology libraries. The QoR metrics include area($\mu\mathrm{m}^2$) and delay(ps), with circuit delays assessed via NLDM or P&R timing analysis supported by iEDA [21].

Exp-1.1: Test of Mapping QoR on ASAP 7nm library. Using the ASAP $7\mathrm{nm}$ technology library, we evaluate the mapping results of our framework alongside ABC and AiMap on 22 benchmark circuits, as detailed in TABLE I. From TABLE I, we have the following findings.

(1) Our framework achieves a significant reduction in delay, with an average achievement of 11% in delay across 22 circuits. Besides, better delay outcomes are observed in 18 out of the 22 benchmark circuits. (2) Compared with ABC, our framework reduces delay by 10% while only incurring

TABLE I Comparison of mapping QoR on ASAP 7nm with ABC and AiMap. "–" refers to the results that are not reported.

Circuits	#Nodes	ABC [4]		AiMap [23]		Ours		Ours/AiMap		Ours/ABC	
		Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area	Delay	Area	Delay
log2	32,060	26,919.81	4,765.43	23,691.45	4,608.79	28,500.52	4,414.87	1.20	0.95	1.05	0.92
square	18,484	15,630.46	2,916.26	15,557.21	2,890.35	17,356.50	2,152.05	1.11	0.74	1.11	0.73
adder	1,020	902.09	3,123.14	1,146.80	2,466.12	1,035.06	2,226.16	0.90	0.90	1.14	0.71
sin	5,416	5,384.57	2,594.42	4,445.15	2,613.81	5,137.29	2,410.90	1.15	0.92	0.95	0.92
div	57,247	64,868.64	41,645.60	61,078.07	54,430.88	60,944.63	44,038.93	0.99	0.80	0.93	1.05
hyp	214,335	223,186.44	191,985.36	–	–	239,115.97	191,622.59	–	–	1.07	0.99
max	2,865	2,429.14	1,879.23	2,233.19	2,094.85	2,452.71	1,873.01	1.09	0.89	1.00	0.99
sqrt	24,618	22,926.76	71,590.62	22,068.29	79,211.18	24,495.10	69,358.84	1.10	0.87	1.06	0.96
mult	27,062	25,598.51	3,324.70	20,082.61	3,319.79	24,683.59	2,859.38	1.22	0.86	0.96	0.86
bar	3,336	2,864.21	191.49	2,417.95	199.60	2,918.33	161.16	1.20	0.80	1.01	0.84
dft	245,046	232,772.38	589.76	–	–	211,119.09	594.55	–	–	0.90	1.00
netcard	802,919	623,919.94	590.22	–	–	670,043.88	566.39	–	–	1.07	0.95
Geomean		22,577.98	4,159.76	–	–	23,131.55	3,786.07	**1.10**	**0.85**	**1.01**	**0.90**
priority	978	782.42	2,110.00	879.93	2,245.19	890.20	2,182.72	1.01	0.97	1.13	1.03
cavlc	693	532.11	137.31	523.01	128.48	531.18	134.05	1.01	1.04	0.99	0.97
arbiter	11,839	9,746.44	628.10	–	–	9,657.33	594.28			0.99	0.94
i2c	1,342	991.21	151.21	1,084.99	135.58	1,000.77	126.32	0.92	0.93	1.00	0.83
voter	13,758	19,513.17	1,027.13	–	–	17,595.38	1,021.54			0.90	0.99
int2float	260	190.12	107.96	185.69	118.04	179.63	106.14	0.96	0.89	0.94	0.98
ctrl	174	118.27	78.00	114.31	72.55	123.64	61.44	1.08	0.84	1.04	0.78
dec	304	389.11	82.12	454.66	77.96	432.27	63.32	0.95	0.81	1.11	0.77
memctrl	46,836	33,950.87	1,330.66	–	–	34,235.94	1,170.94			1.00	0.87
router	257	242.14	396.28	230.48	451.23	233.75	398.48	1.01	0.88	0.96	1.00
Geomean		1,188.96	317.18	–	–	1,199.73	290.79	**0.99**	**0.90**	**1.00**	**0.91**

TABLE II Comparison of mapping QoR on Sky 130nm.

Circuits	ABC [4]		Ours		Ours/ABC	
	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area	Delay
log2	116,342.83	27,041.88	111,779.70	25,345.28	0.96	0.93
square	70,929.27	14,787.94	69,933.32	14,075.02	0.98	0.95
adder	5,024.82	14,989.45	4,570.63	14,397.89	0.90	0.96
sin	23,488.78	13,227.36	21,324.20	12,887.86	0.90	0.97
div	280,581.59	301,243.12	253,514.39	270,445.28	0.90	0.89
hyp	927,249.31	1,436,186.25	884,136.69	1,268,164.88	0.95	0.88
max	12,084.09	12,303.26	12,296.79	11,396.12	1.01	0.92
sqrt	151,436.48	386,521.16	114,621.18	340,405.50	0.75	0.88
mult	107,376.73	17,877.35	94,474.36	17,449.49	0.87	0.97
bar	13,237.70	959.49	12,993.71	856.98	0.98	0.89
dft	947,417.38	3,212.88	879,614.88	2,902.46	0.92	0.90
netcard	2,906,318.50	3,497.67	2,761,997.75	3,002.90	0.95	0.85
Geomean	105,092.35	23,702.55	97,275.42	21,793.37	**0.92**	**0.91**
priority	4,789.59	15,458.25	3,684.78	13,129.57	0.76	0.84
cavlc	2,393.55	907.34	2,516.16	790.48	1.05	0.87
arbiter	49,361.09	3,944.17	61,870.59	3,809.66	1.25	0.96
i2c	4,529.34	963.60	4,558.12	815.74	1.00	0.84
voter	74,600.30	5,309.04	56,057.51	4,557.42	0.75	0.85
int2float	825.79	724.01	855.82	702.16	1.03	0.96
ctrl	614.34	455.84	571.80	417.11	0.93	0.91
dec	1,898.07	475.91	1,889.31	476.06	0.99	1.00
memctrl	152,540.05	7,218.72	153,327.05	6,438.28	1.00	0.89
router	1,217.42	2,481.06	1,137.34	2,322.72	0.93	0.93
Geomean	5,667.50	1,950.28	5,461.96	1,772.72	**0.95**	**0.90**

TABLE III Comparison of timing after P&R on Sky 130nm.

Circuits	ABC [4]			Ours			Ours/ABC		
	#OF	TNS	WPD	#OF	TNS	WPD	OF	TNS	WPD
log2	8.8e4	-621.5	21.2	8.8e4	-600.5	20.9	0.99	0.96	0.98
square	1.9e3	-26.4	11.4	1.9e4	-10.3	10.9	1.01	0.39	0.96
adder	12	-18.0	11.3	2	-1.71	10.3	0.16	0.09	0.91
sin	4.0e3	-10.0	10.8	4.2e3	-6.4e3	10.6	1.03	0.64	0.98
div	3.7e3	-3.7e4	229.3	3.7e3	-3.7e5	230.8	1.01	1.00	1.00
hyp	5.8e5	-1.3e5	1.4e3	4.8e5	-1.0e5	1.2e3	0.83	0.78	0.78
max	162	-230.6	10.9	328	-89.67	10.4	2.02	0.38	0.95
sqrt	928	-1.2e4	307.3	760	-1.3e4	301.9	0.81	1.04	0.98
mult	2.6e3	-178.1	14.1	1.9e3	-185.8	14.2	0.72	1.04	1.00
bar	1.4e3	0.0	0.8	722	0.0	0.7	0.50	–	0.96
dft	2.1e7	0.0	2.6	2.1e7	0.0	2.41	0.99	–	0.92
netcard	8.1e7	0.0	2.6	8.1e7	0.0	2.40	1.00	–	0.91
Geomean	3.2e3	-2.4e3	28.6	2.4e3	-1.7e3	27.2	0.78	0.58	0.94
priority	14	-1.9	11.1	0	-2.5	10.9	0.0	1.28	0.99
cavlc	26	0.0	0.7	52	0.0	0.6	2.0	–	0.92
arbiter	228	0.0	3.2	284	0.0	3.0	1.25	–	0.97
i2c	348	0.0	0.7	6	0.0	0.7	0.02	–	0.97
voter	518	0.0	4.3	238	0.0	4.2	0.46	–	0.99
int2float	28	0.0	0.6	10	0.0	0.5	0.36	–	0.98
ctrl	2	0.0	0.4	0	0.0	0.346	0.0	–	0.97
dec	1.0e3	0.0	0.4	5.1e3	0.0	0.3	5.08	–	0.85
memctrl	9.3e4	0.0	5.5	9.9e4	0.0	5.4	1.07	–	0.97
router	2	0.0	2.0	0	0.0	1.8	0.0	–	0.93
Geomean	432.1	-1.9	1.51	271.1	-2.5	1.44	0.63	1.28	0.95

"OF" denotes resource overflows, "TNS" represents total negative slack (ns), and "WPD" stands for worst path delay (ns).

a 1% penalty in area on arithmetic circuits. We achieved a significant delay reduction in the adder circuit, lowering it from 3123.14ps to 2226.16ps. Additionally, for control circuits, our framework reduced delay by 9% without incurring any area penalty. (3) Compared with AiMap, our framework also achieves a substantial improvement in the QoR, *i.e.,* an average 12.5% reduction in delay while incurring an additional 4.5% area expense. AiMap [23], which learns supergate delays, encounters disruptions in delay estimations during area recovery. As a result, it mainly optimizes area, with limited improvements in delay.

Exp-1.2: Test of Mapping QoR on Sky 130nm library. To assess the scalability of our framework across different libraries, we compare its performance with ABC on the benchmarks using 130nm technology library, as shown in TABLE II. From TABLE II, we have the following findings.

(1) Better results than those achieved on ASAP 7nm, our framework significantly improves mapping QoR, achieving an average reduction of 9.5% in delay and 6.5% in area across 22 circuits with Sky 130nm library. (2) Although the benchmarks are the same, the QoR improvements achieved by our framework vary across different technology libraries. For example, hyp and netcard circuits exhibit the most significant optimizations within Sky 130nm, with delay reductions of 12% and 15%, respectively. This indicates that our framework effectively aligns circuit structures with technology characteristics, optimizing gate selection under varying technology con-

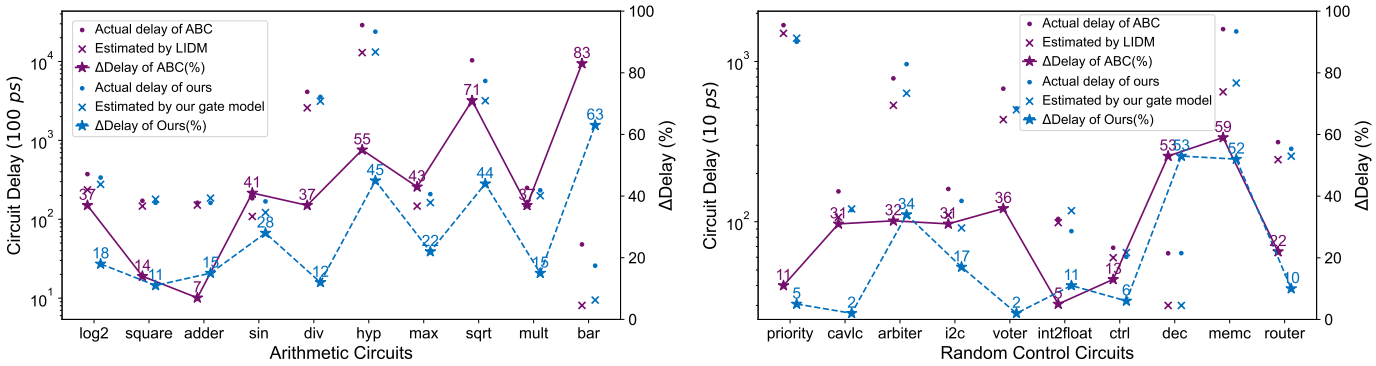

Fig. 6 Gap analysis between estimated gate delay and NLDM with Sky 130nm: ABC's gate delay is estimated by LIDM, while ours uses a structure-aware load-slew model. The markers "·" and "×" correspond to the left y-axis, and "⋆" to the right y-axis.

TABLE IV Ablation study of our framework design with Sky 130nm library on arithmetic circuits.

Circuits	Ours		*w/o* Gradient Descent		*w/o* Bayesian Optimization		ABC		ABC *w* Gain Search		ABC *w* &nf	
	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)
log2	111,779.7	25,345.3	112,534.2	25,156.8	112,024.9	25,659.7	116,342.8	27,041.9	113,928.0	27,445.8	114,045.6	28,147.2
square	69,933.3	14,075.0	69,954.6	14,094.5	70,670.3	14,214.0	70,929.3	14,787.9	68,822.3	15,190.9	68,517.0	15,036.2
adder	4,570.6	14,397.9	4,453.0	14,854.3	5,290.1	14,270.7	5,024.8	14,989.5	4,295.4	14,055.8	5,316.4	14,494.6
sin	21,324.2	12,887.9	21,484.4	12,790.0	23,946.7	12,619.3	23,488.8	13,227.4	22,370.2	13,309.7	22,547.9	13,862.4
div	253,514.4	270,445.3	253,514.4	270,445.3	264,698.9	271,967.5	280,581.6	301,243.1	226,320.8	359,093.6	324,687.7	289,773.9
hyp	884,136.7	1,268,164.9	877,780.6	1,268,140.4	889,161.5	1,277,024.9	927,249.3	1,436,186.3	907,945.8	1,463,113.5	1,128,720.0	1,509,230.3
max	12,296.8	11,396.1	12,185.4	11,494.0	12,560.8	11,417.5	12,084.1	12,303.3	12,084.1	12,303.3	11,528.6	13,121.8
sqrt	114,621.2	340,405.5	103,780.8	374,990.4	114,856.4	340,940.4	151,436.5	386,521.2	113,916.3	444,653.4	163,633.2	401,344.7
mult	94,474.4	17,449.5	98,199.2	16,827.9	98,897.4	17,110.2	107,376.7	17,877.4	104,428.9	17,497.3	101,144.5	18,206.9
bar	12,993.7	857.0	12,587.1	897.6	13,337.8	886.1	13,237.7	959.5	12,792.3	968.2	12,365.6	1,017.4
dft	879,614.9	2,902.5	879,721.2	2,901.6	918,608.5	2,895.7	947,417.4	3,212.9	907,315.2	3,264.3	873,187.4	3,393.8
netcard	2,761,997.8	3,002.9	2,724,101.3	3,466.8	3,535,209.3	3,523.9	2,906,318.5	3,497.7	2,790,608.8	3,323.5	2,510,547.8	3,641.4
Geomean	97,275.4	21,793.3	96,194.9	22,300.8	103,196.4	22,122.0	105,092.3	23,702.5	97,309.0	24,242.5	104,900.4	24,413.8
Geo. Ratio	**0.92**	**0.91**	**0.91**	**0.94**	**0.98**	**0.93**	**1.00**	**1.00**	**0.92**	**1.02**	**0.99**	**1.03**

ditions, and demonstrates scalability across different libraries.

From the findings of *Exp-1.1* and *Exp-1.2*, the notable delay reduction achieved by our framework originates from our structure-aware load-slew gate model. This model anticipates the non-linear delay model of the gate by aligning library characteristics with the circuit structure. Specifically, (1) the essential timing characteristics of each gate, such as LD, PD^L, SD, and PD^S, are summarized from the library, reflecting the physical behaviors of the gates. (2) By integrating these timing features into the potential mapping structure between the AIG and netlist, our gate model can predict possible gate delays before mapping. (3) Additionally, by tuning global and local parameters, we have successfully harmonized these two elements to further minimize the overall circuit delay.

Exp-1.3: Test of Timing after Physical Design. To further validate the benefits of our framework in enhancing timing for physical design, we have incorporated it as a front-end step in the ASIC design flow, utilizing the physical toolkit iEDA [22]. Employing Sky 130nm library, we compare the post-P&R timing metrics of netlists generated by ABC with those from our mapping framework. The evaluated metrics include overflow (OF), total negative slack (TNS), and worst path delay (WPD), as detailed in TABLE III. Note that, during the floorplan stage in iEDA, each circuit is allocated an identical overall chip area, which is then followed by the placement and routing phases. Therefore, circuits that exhibit a higher degree of overflow during the routing phase typically indicate a larger potential area requirement. From TABLE III, we have the following findings.

(1) After P&R, the netlists generated by our framework exhibit a notable reduction in both WPD and TNS, with average decreases by 7% and 6%, respectively, across 22 benchmarks. This is because the fanout numbers in the netlist can be used to estimate wire delays, and we have approximately incorporated the nodes' fanout into our gate model. Moreover, the 30% reduction in resource overflows during the routing phase using the Sky 130nm library is closely correlated with the 6.5% area reduction achieved during the technology mapping evaluation in our framework (2) Specifically, on arithmetic circuits, hyp and netcard circuits exhibit the most reductions in WPD after P&R, decreasing by 22% and 9% respectively. These results are consistent with the mapping QoR achievement observed in *Exp-1.1* and *Exp-1.2*. (3) A joint analysis of TABLE II and TABLE III, suggests a positive correlation between the timing under NLDM and post-P&R. Circuits that perform well under NLDM generally exhibit superior post-P&R timing.

C. In-Depth Analysis of our Mapping Framework

Exp-2: Gap Analysis: Estimated Delay vs. NLDM. Before mapping, gate delay in ABC is estimated by LIDM, while our framework employs a structure-aware load-slew model. After the completion of the mapping and post-mapping, the circuit delay is estimated using NLDM, which is considered the actual delay. We analyze the gap between these estimated delays and actual delays using Sky 130nm, as illustrated in Fig. 6. ΔDelay is defined as the ratio of the absolute difference between the estimated and the actual delay to the estimated delay. From Fig. 6, we have the following findings.

(1) Our estimations of gate delay are more accurate than those by ABC, with the gap between our estimated gate model

TABLE V Comparison of fanout number distribution after technology mapping (Sky 130nm library).

Circuits	#Nodes	ABC [4]							Ours						
		Distribution of #fanouts							Distribution of #fanouts						
		0-100	100-300	300-500	>500	Max	Level	Delay(ps)	0-100	100-200	200-300	300-500	Max	Level	Delay(ps)
div	57,247	50,158	34	1	0	354	2,253	301,243.12	53,788	11	0	0	237	3,118	270,445.28
hyp	214,335	146,066	299	2	0	334	16,260	1,436,186.25	165,184	249	0	0	181	16,918	1,268,164.88
sqrt	24,618	21,300	53	0	0	222	2,761	386,521.16	18,810	21	0	0	114	4,029	340,405.50
dft	245,046	176,659	163	9	11	2,746	32	3,212.88	169,574	140	0	0	256	41	2,902.46
netcard	802,919	540,172	83	8	250	29,502	25	3,497.67	572,129	195	31	116	32,768	32	3,002.90
mult	27,062	18,772	0	0	0	77	154	17,877.35	16,873	0	0	0	77	262	17,449.49
memctrl	46,836	29,803	28	1	0	383	64	7,218.72	30,341	26	1	0	324	92	6,438.28
Improvement						1.00	1.00	1.00					**0.55**	**1.35**	**0.89**

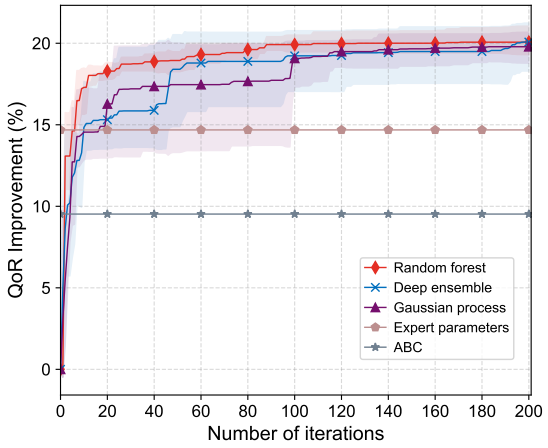

Fig. 7 QoR improvement of sin circuit over iterations under different global parameter search strategies (Sky 130nm library).

and the actual delay being only 27%, compared to ABC's 42%. Note that, due to unpredictable factors during the mapping process, *e.g.,* inverter insertions, we cannot precisely estimate gate delay before mapping. (2) Besides, by leveraging our gate model, the mapper effectively guides the gate selection, facilitating a reduction in circuit delay with an average achievement of 9.5%. This supports the analysis of the delay model and underscores the three key insights that informed the design of our iterative framework, as discussed in Section III.

Exp-3: Analysis of Ablation Study. We conduct ablation studies to assess the contribution of each component in our framework and justify its design, as detailed in TABLE IV. The variants are designated as follows: (i) w/o Gradient Descent, where the framework does not apply gradient descent to optimize local parameters $F^{\tau}(n)$; (ii) w/o Bayesian Optimization, where the framework forgoes the optimization of global parameters θ, relying solely on our structure-aware load-slew model with heuristic parameters for mapping; (iii) ABC, where we use default command `map` in ABC, as described in Section VI-A; (iv) ABC w Gain Search, where the BO for the user-defined gain in LIDM is applied (*i.e.,* adjusting parameters for `map -G` in ABC). This allows us to evaluate LIDM's capability to model global capacitance across different circuits to evaluate the capability of LIDM to model global capacitance across different circuits. (v) ABC w `&nf`, where we employ the new AIG manager (GIA) introduced in ABC9, and perform mapping with `&nf`. From TABLE IV, we have the following findings.

(1) It is observed the delay achievement drops 3% when the gradient descent for optimizing $F^{\tau}(n)$ is removed. This decline is attributed to the actual gate delays obtained from the mapped netlists being overlooked, resulting in the τ-order fanout tiers in the AIG that are locally inaccurate not being promptly corrected. This finding validates the analysis of *Insight 3* regarding circular dependency in delay computation, as discussed in Section III-B. (2) The QoR achievement further drops 3% when the local and global parameters for optimizing $F^{\tau}(n)$ and θ are both omitted. This is because the timing obtained after each covering iteration cannot be used to guide the gate delay adjustments. This observation further supports the analysis of *Insight 3*. (3) Even when solely employing our structure-aware load-slew gate model, accompanied by heuristic parameters, an 8% delay reduction is achieved compared to ABC. This outcome corroborates the analysis of *Insight 1 and Insight 2*, which address the issues of inaccurate load estimation and the insufficiency of load-only coupled, as detailed in Section III-B. (4) Adjusting the gain of LIDM in ABC mainly results in a trade-off between area and delay [16]. The gain search improves delay in only 3 out of 12 circuits, and achieves a 7% reduction in area at the cost of a 2% delay penalty. (5) Compared to `map`, `&nf` command results in a 3% increase in delay, with a 1% reduction in area. In other words, our framework achieves a 12% average reduction in delay compared to `&nf`.

Exp-4: Analysis of Fanout Numbers. We compare the distribution of fanout numbers for circuits with more than 20,000 nodes, where the fanout numbers are aggregated without postmapping, as shown in TABLE V. From TABLE V, we have identified the following findings.

(1) Our method tends to generate netlists with smaller fanout numbers. Specifically, the maximum fanout in these larger circuits generated by our method is reduced by an average of 45%. This improvement is attributed to our structure-aware load-slew gate model, which accounts for the potential impact of fanout on delay. (2) We have compared the levels of mapped netlists between our method and ABC. An interesting observation is that the level is not a definitive criterion for delay, *i.e.,* a netlist with a smaller level does not necessarily correlate with reduced circuit delay. In our experiments, even though the level of the netlist increases by 35%, the corresponding netlist delay decreases by 8%. This finding aligns with those of the study in [20] concerning the relationship between LUT level and P&R delay in FPGA mapping.

Exp-5: Analysis of QoR *w.r.t.* BO Iterations. We analyze QoR improvements after postmapping over iterations of different global parameter search strategies across five random seeds. We compare three different surrogate models of BO, *i.e.,*

TABLE VI Overall runtime (Sky 130nm library). Area in 1e3 μm^2, delay in 1 ns, runtime ("RT") in 1 s, and #node in 1e3. "Ours -e" denotes only using one set of expert parameters.

Cir.	#Nodes	ABC [4]			Ours			Ours -e		
		Area	Delay	RT	Area	Delay	RT	Area	Delay	RT
div	57.2	280.6	301.2	235.4	253.5	270.4	1,078.6	264.7	272.0	55.1
hyp	214.3	927.2	1,436.2	564.1	884.1	1,268.2	2,653.9	889.2	1,277.0	145.7
sqrt	24.6	151.4	386.5	69.5	114.6	340.4	554.8	114.9	340.9	37.8
mult	27.1	107.4	17.9	16.6	94.5	17.4	114.7	98.9	17.1	7.8
memc.	46.8	152.5	7.2	16.3	153.3	6.4	158.18	152.0	6.6	10.4
dft	245.0	947.4	3.2	50.1	879.6	2.9	383.3	918.6	2.9	24.4
netc.	802.9	2,906.3	3.5	476.4	2,762.0	3.0	4,282.3	3,535.2	3.5	309.2
Impr.		1.00	1.00	1.00	**0.91**	**0.90**	**11.56**	**0.96**	**0.92**	**0.41**

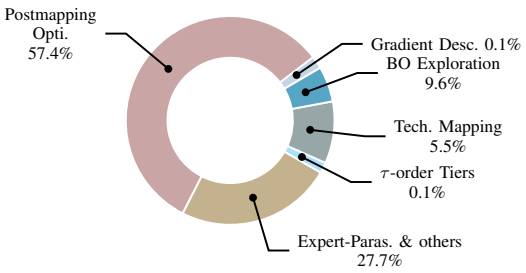

Fig. 8 The runtime breakdown of our framework on sqrt circuit.

random forest, deep ensemble, and Gaussian processes, against 1 set of expert parameters of our framework and ABC (the map command with default gain parameter of 250), as shown in Fig. 7. These QoR are calculated by Equation (9) based on the results of mapping and postmapping, in alignment with the flow in Algorithm 2. From Fig. 7, it is evident that employing random forest as the surrogate model for BO yields the best results, almost 0.3% better than using Gaussian processes. This justifies the design of the random forest for surrogate modeling in our delay-driven iterative framework.

Exp-6: Analysis of Runtime. We assess the total runtime of our framework for circuits with more than 20,000 nodes in TABLE VI, followed by an analysis of key factors contributing to the total runtime by a detailed breakdown in Fig. 8.

Exp-6.1: Overall Runtime Comparison. From TABLE VI, in circuits with more than 20,000 nodes, our framework with a single set of expert-designed parameters ("Ours -e"), is on average 59% faster than ABC while achieving an 8% reduction in delay and 4% reduction in area. Although the mapping time for our method is comparable to ABC, the postmapping process for the generated netlist is typically much faster than ABC. For instance, in the sqrt circuit, ABC inserts 48% more buffers and explores 20% more paths for sizing compared to "Ours -e". When employing our complete framework for mapping, it averages 11.56 times slower than the default ABC. The primary contributor to the time cost is the use of BO implemented by PyTorch, and the evaluation of the BO objective function. Note that "Ours -e" trades off mapping QoRs, sacrificing a 2% increase in delay compared to "Ours".

Exp-6.2: Runtime Breakdown. Fig. 8 illustrates the overall runtime breakdown for the sqrt circuit within our delay-driven iterative framework. Postmapping optimization among BO takes 57.5% of the total runtime, while the gradient descent for local parameters and τ-order tiers computation each contribute only 0.1%. The large portion of time consumed by the iterative framework is primarily due to the evaluation of the BO objective function in Equation (9).

VII. Conclusions

By analyzing the local structures between AIG and potential mapped netlists, we developed a structure-aware load-slew gate model, enabling accurate estimation of the gate's nonlinear delay before mapping. Utilizing our model to guide gate selections, we introduced a delay-iterative framework to minimize circuit delay progressively. As a result, we observed an average reduction of 10% in delay and 2% in area with NLDM and 6% in delay after P&R, as compared to ABC.

From the analyses and experiments, it is evident that the opportunity to optimize mapping algorithms aimed at reducing circuit delay lies in effectively aligning the local structures of logical networks with potential netlists. Such findings could inspire further research into mapping algorithms.

References

[1] C. Albrecht. Iwls 2005 benchmarks. In *Proc. International Workshop for Logic Synthesis (IWLS)*, volume 9, 2005.
[2] L. Amarú, P.-E. Gaillardon, and G. De Micheli. The epfl combinational benchmark suite. In *Proc. International Workshop on Logic Synthesis(IWLS)*, 2015.
[3] L. Amarú, P. Vuillod, J. Luo, and J. Olson. Logic optimization and synthesis: Trends and directions in industry. In *Proc. Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 1303–1305. IEEE, 2017.
[4] R. Brayton and A. Mishchenko. Abc: An academic industrial-strength verification tool. In *Proc. International Conference on Computer-Aided Verification (CAV)*, pages 24–40. Springer, 2010.
[5] A. T. Calvino and G. De Micheli. Technology mapping using multi-output library cells. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.
[6] A. T. Calvino, A. Mishchenko, H. Schmit, E. Mahintorabi, G. De Micheli, and X. Xu. Improving standard-cell design flow using factored form optimization. In *Proc. ACM/IEEE Design Automation (DAC)*, pages 1–6. IEEE, 2023.
[7] S. Chatterjee. *On algorithms for technology mapping*. University of California, Berkeley, 2007.
[8] S. Chatterjee and et al. Reducing structural bias in technology mapping. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 25(12):2894–2903, 2006.
[9] A. B. Chowdhury, B. Tan, R. Karri, and S. Garg. Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis. *arXiv preprint arXiv:2110.11292*, 2021.
[10] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric. Asap7: A 7-nm finfet predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.
[11] A. Cowen-Rivers and et al. Hebo: Pushing the limits of sample-efficient hyperparameter optimisation. *Journal of Artificial Intelligence Research*, 74, 2022.
[12] Z. Gao and D. S. Boning. A review of bayesian methods in electronic design automation. *arXiv preprint arXiv:2304.09723*, 2023.
[13] H. Geng and et al. Ptpt: Physical design tool parameter tuning via multi-objective bayesian optimization. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 42(1):178–189, 2022.
[14] Google. SkyWater-PDK. https://github.com/google/skywater-pdk, 2021. Accessed: May 1, 2024.
[15] A. Grosnit and et al. Boils: Bayesian optimisation for logic synthesis. In *Proc. Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 1193–1196. IEEE, 2022.
[16] B. Hu, Y. Watanabe, A. Kondratyev, and M. Marek-Sadowska. Gain-based technology mapping for discrete-size cell libraries. In *Proc. ACM/IEEE Design Automation (DAC)*, pages 574–579, 2003.
[17] S.-C. Huang and et al. A dynamic accuracy-refinement approach to timing-driven technology mapping. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 538–543. IEEE, 2008.

[18] D. Jongeneel and R. H. Otten. Technology mapping for area and speed. 29(1):45–66, 2000.
[19] S. K. Karandikar and et al. Technology mapping using logical effort for solving the load-distribution problem. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 27(1):45–58, 2007.
[20] X. Li, L. Chen, F. Yang, M. Yuan, H. Yan, and Y. Wan. Himap: a heuristic and iterative logic synthesis approach. In *Proc. ACM/IEEE Design Automation (DAC)*, pages 415–420, 2022.
[21] X. Li and et. al. iEDA: an open-source infrastructure of eda. In *Proc. Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 77–82. IEEE, 2024.
[22] X. Li and et. al. iPD: an open-source intelligent physical design toolchain. In *Proc. Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 83–88. IEEE, 2024.
[23] J. Liu, L. Ni, X. Li, M. Zhou, L. Chen, X. Li, Q. Zhao, and S. Ma. Aimap: Learning to improve technology mapping for asics via delay prediction. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 344–347. IEEE, 2023.
[24] A. Mishchenko, R. Brayton, and M. Fujita. Mapping and retiming revisited. In *Proc. International Workshop on Logic Synthesis(IWLS)*, number CONF, 2023.
[25] A. Mishchenko, S. Chatterjee, R. Brayton, X. Wang, and T. Kam. Technology mapping with boolean matching, supergates and choices. *Berkeley Logic Synthesis and Verification Group*, 2005.
[26] R. Murgai. On the complexity of minimum-delay gate resizing/technology mapping under load-dependent delay model. In *Proc. International Workshop on Logic Synthesis(IWLS)*, pages 209–211, 1999.
[27] W. L. Neto, M. T. Moreira, Y. Li, L. Amarù, C. Yu, and P.-E. Gaillardon. Slap: a supervised learning approach for priority cuts technology mapping. In *Proc. ACM/IEEE Design Automation (DAC)*, pages 859–864. IEEE, 2021.
[28] A. L. Oliveira and R. Murgai. On the problem of gate assignment under different rise and fall delays. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 22(6):807–814, 2003.
[29] Z. Pei and et al. Alphasyn: Logic synthesis optimization with efficient monte carlo tree search. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.
[30] R. L. Rudell. *Logic synthesis for VLSI design*. University of California, Berkeley, 1989.
[31] M. Sentovich. Sis: A system for sequential circuit synthesis. *Memorandom no. UCB/ERL M92/41*, 1992.
[32] S. Shirobokov, V. Belavin, M. Kagan, A. Ustyuzhanin, and A. G. Baydin. Black-box optimization with local generative surrogates. *Proc. Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:14650–14662, 2020.
[33] I. Sutherland, B. Sproull, and D. Harris. *Logical effort: designing fast CMOS circuits*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
[34] R. Murgai. Technology-dependent logic optimization. *Proceedings of the IEEE*, 103(11):2004–2020, 2015.
[35] P. Wang and et al. Easymap: Improving technology mapping via exploration-enhanced heuristics and adaptive sequencing. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pages 01–09. IEEE, 2023.
[36] Z. Wang and et al. A circuit domain generalization framework for efficient logic synthesis in chip design. *arXiv preprint arXiv:2309.03208*, 2023.
[37] M.-C. Wu, A. Q. Dao, and M. P.-H. Lin. A novel technology mapper for complex universal gates. In *Proc. Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 475–480, 2021.
[38] Y. Ye and et al. Timing-driven technology mapping approximation based on reinforcement learning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2024.

Junfeng Liu received the PhD degree in computer science from Beihang University, China, in 2024. He is currently a postdoctoral researcher at Pengcheng Laboratory, China. His research interests include EDA logic synthesis and graph data management.

Liwei Ni received the B.S. degree in computer science from the Anhui University of Finance and Economics, Bengbu, China, in 2018, and the M.S. degree in Software Engineering from the Beihang University, Beijing, China in 2021. He is pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, and is jointly trained with Pengcheng Laboratory. His research focuses on the logic synthesis.

Lei Chen received the PhD degree in computer science from Hong Kong Baptist University, Hong Kong, in 2016. He is currently a senior researcher of Huawei Noah's Ark Lab, Hong Kong, China. His research interests include EDA logic synthesis, circuit data modeling, data-driven optimization algorithms and big data management.

Xing Li is a senior researcher at Huawei Noah's Ark Lab in Hong Kong, specializing in AI and logic synthesis. He received the master degree from RWTH Aachen University in Germany in 2020.

Qinghua Zhao received the PhD degree in computer science from Beihang University, China, in 2024. She is currently a lecturer at the School of Artificial Intelligence and Big Data, Hefei University, Hefei, China. Her research interests include data-driven AI and NLP.

Xingquan Li received the Ph.D degree from Fuzhou University in 2018. He is currently an Associate Researcher at Pengcheng Laboratory (PCL). His research interests include EDA and AI for EDA. His team has developed an open-source infrastructure of EDA and toolchain (iEDA). He has published over 50 papers in journals and conferences such as TCAD, TC, TVLSI, TODAES, DAC, ICCAD, DATE, ICCD, ASP-DAC, ISPD. He received the Best Paper Award from ISEDA 2023.

Shuai Ma received the Ph.D. degrees in computer science from Peking University, China, in 2004, and from The University of Edinburgh, England, in 2010, respectively. He is a professor with the School of Computer Science and Engineering, Beihang University, China. He was a postdoctoral research fellow with the Database Group, University of Edinburgh, a summer intern at Bell Labs, Murray Hill, NJ, and a visiting researcher of MSRA. His current research interests include big data, database theory and systems, and graph data analysis.