Delay-Driven Physical-Aware Logic Synthesis with Reinforcement Learning

Linyu Zhu, Xinfei Guo*
University of Michigan – Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University, Shanghai, China
*Corresponding author's email: xinfei.guo@sjtu.edu.cn

***Abstract*—A typical design flow is separated into front-end and back-end stages, incurring huge number of iteration loops between logic synthesis and place and route to close timing. This has been even worse in advanced technology where wire delay dominates timing. It becomes increasingly important to integrate physical awareness in the logic synthesis optimization processes to achieve better timing correlations. To tackle the physically-aware synthesis challenges, in this paper, we formulate the whole design flow as a multi-stage search problem, where informed search algorithms are utilized to perform efficient search with additional guidance. By incorporating a newly-developed learning-based routing-aware timing prediction model in the inform value function, a delay-driven physically-aware synthesis methodology called DDPAS is proposed, enabling efficient logic optimization with awareness of final routed design. The framework pairs with various search algorithms and learning-based strategies. Evaluation results show that the reinforcement learning (RL) based DDPAS delivers 65.6% improvement in terms of TNS and 12.1% power savings compared to a state of the art RL-based logic synthesis framework. Overall, DDPAS yields to better final QoR after routing and significantly reduces the timing closure cycles.**

***Index Terms*—Logic optimization, Physically-aware synthesis, Timing closure, Informed search, QoR**

I. Introduction

Recently, Electronic Design Automation (EDA) industry has started to push towards a "shift-left" paradigm that mirrors the success of software industry. Shifting left in EDA indicates that the design cycle and time to market is shortened by accounting for later flow steps early in the cycle by removing design bottlenecks in advance and avoiding later "surprises" [1]. A typical design implementation flow can be divided into frontend and backend stages naturally, where the frontend steps synthesize the behavior description of a design into a mapped netlist, and the backend steps transform the netlist into a physical layout. As chip geometries become smaller, wire delays dominate a timing path. It becomes increasingly difficult to close timing without going through multiple iteration loops between place and route and logic synthesis. Miscorrelation in timing and unpredictability occur frequently in design closure. It thus becomes imperative that logic synthesis and place and route are essentially done at the same time.

Physically-aware synthesis appeared to enhance the traditional synthesis flow by making use of the actual design and physical information [2], [3]. The fundamental challenges from developing a robust and effective physically-aware synthesis methodology arise from two aspects. Firstly, the interconnect dominates the delay equation and requires to be disclosed in early phases where the actual wires don't even exist. This indicates that physically-aware timing prediction at the synthesis phase is extremely difficult based on very limited design and technology information [4]. The second challenge is based on the fact that design closure involves a very large search space for convergence. An ideal flow is supposed to be just a one-pass linear process, while much of iterative loops and checks between frontend and backend are required to fix problems, such as validation failure, timing closure problems and bad quality of results (QoR) in general. Thus, it becomes extremely challenging to fuse frontend and backend without incurring even longer iteration cycles. In the last decade, progress has been made to tackle both challenges. For the first challenge, various machine learning techniques have been introduced to resolve the timing prediction problems [4]–[7]. Though these work showed promising results in terms of reducing prediction error when compared to the ground truth labeled values, many of them focused on correlating pre- and post-routing timing, and even for those focused on predicting timing in synthesis, sophisticated feature selection and processing are required, making them less feasible to be adapted in a synthesis engine that in general requires fast and accurate timing responses. While for the second challenge, most of the existing work applied approaches to increasingly integrated the distinct design processes by moving later steps to earlier phases. For example, early placement or routing can be done in synthesis [1]. But results showed that this movement in synthesis can result in about marginal improvement in performance while incurring similar or even more run time [1]. Fundamentally, the search space still remains as large as before with this merge. In this paper, we attempt to push the physically-aware synthesis envelop further by attacking both challenges. We firstly model the whole EDA flow as a multi-stage search process. Inspired from the idea of informed search algorithm which uses additional knowledge to increase the efficiency of the search, we propose a delay driven physically aware synthesis framework named DDPAS to enhance the AIG-based logic synthesis optimization to potentially produce better correlation with place and route

This work is supported by the National Key R&D Program of China (2021ZD0114701) and a CCF-Tencent Open Fund (RAGR20220114).

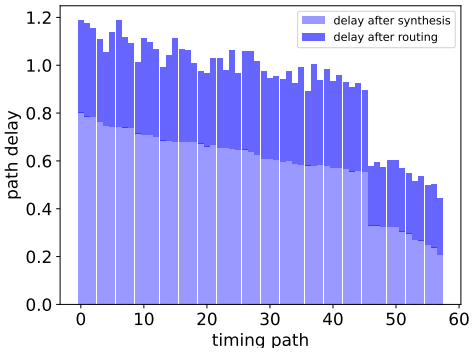

Figure 1: Path delay miscorrelation between synthesis and routing in an AES design. The X-axis shows path IDs, and the path delay is given based on static timing analysis. In general, timing after routing tends to be worse. Critical path can switch after place and route.

steps. The inform value function is given by a newly proposed learning based timing prediction model to guide the searching. This model achieves up to 99.5% of prediction accuracy, which gives 4% improvement over prior work. The proposed DDPAS framework is compatible with multiple informed searching algorithms, and is evaluated by reinforcement learning (RL) as an example.

II. Background and Related Work

Timing closure problems occur mainly due to the unawareness about physical information (e.g. interconnect delay) in earlier stages. For a conventional logic synthesis process, netlist is generated with limited consideration of routing information, many possible timing optimization techniques such as cell resizing are thus missed. Fig. 1 shows an example where an AES design ran through the full OpenRoad implementation flow [8] based on the 45nm open-source technology. It demonstrates how timing mis-correlation can happen with respect to the same timing path before and after place and route. It should be also noted that critical paths can switch, and this leads to completely different optimization targets between synthesis and place and route. Accurate interconnect delay information are thus necessary during the synthesis to close this gap before entering the physical implementation stages.

Wireload models are widely used for interconnect delay estimation, but they are inaccurate due to that advanced technology nodes involve very sophisticated parasitics and coupling. Recently, machine learning (ML) techniques started to play an important role in EDA industry. Many recent work leveraged the power of ML to predict timing in early design stages with pure RTLs or synthesized netlist. [9] formalized logic paths as sentences, with the gates being a bag of words, word embedding can be leveraged to represent generic paths and predict whether a given path is likely to be critical after P&R. The focus of this work was to train a conventional neural network (CNN) to be able to identify the critical path from a synthesized netlist. [6] proposed to estimate the delay of every individual cell arc and net arc with random forest (RF) models. This solution required prediction of the net size first and incurred long run time. In [5], an end-to-end graphic neural network was proposed to estimate predict delay and slew. The input graphs and their features came directly from the earlier RTL representation. They mapped the slew and delay prediction as an edge-level regression task. However, such model is expensive to run. In addition, many work purely focused on the prediction itself without considering full timing picture such as the whole path delay or slack after routing which is essential for timing optimization. We argue that prediction error is only one measure of the timing prediction, another key aspect is how to use these prediction models efficiently so they can be used to guide the optimizations to achieve real physical awareness in a synthesis process.

The idea of physically-aware synthesis was not new, while it has just been introduced as part of products in recent years. Logic synthesis integrated with virtual P&R is now a well-known methodology to handle timing closure in commercial tools [10], [11]. Virtual P&R is a simplified version of actual P&R. For instance, placement is done through a coarse placement without legalization [12], and routing is performed through shortest connection without consideration on routing resources and layer assignment. This kind of integration can only partially addressed the separation of the frontend and backend problem, it is still time-consuming and relies on physical information which can change significantly in later stages. [3] proposed a novel design flow by integrating a high level synthesis (HLS) tool with physically aware logic synthesis technology to deal with congestion problems. This allows designers to resolve the congestion problems before going to the layout design phase. Similarly, [13] introduced physical awareness in HLS for power optimization. [14] proposed an algorithm that allowed the power of logic restructuring using Boolean methods to be unleashed without the fear of long wires introduced in later steps. These work either focused on different optimization target (instead of timing) or failed to provide a physically-aware optimization technique that is adaptive and portable. It is well known that logic synthesis optimization space exploration is an intractable problem, due to the large number of possible permutations. Several existing work offered promising solutions to narrow down the search space. [15] developed a framework called DRiLLS, which transferred the process of logic optimization to a deterministic Markov decision process (MDP) and employed Advantage Actor Critic (A2C) method to perform the logic synthesis optimization. In a similar category, [16]–[18] also used policy gradient algorithms as the learning strategy, whose policy function was modeled based on Graph Convolution Networks (GCN). Besides, Bayesian Optimization in BOiLS [19] and proximal policy optimization with long short-term memory networks (LSTM) in [20] were applied to explore the logic synthesis optimization space. However, all these prior works were physically-unaware and only explored local search, without considering physical information such as wire length or wire delays. Our proposed work differentiates from prior work in several aspects. *Firstly, the proposed physically-aware timing prediction model is developed with the goal of being easily integrated to guide the search. Thus it outputs the path delay and slack directly and it is developed to be light yet*

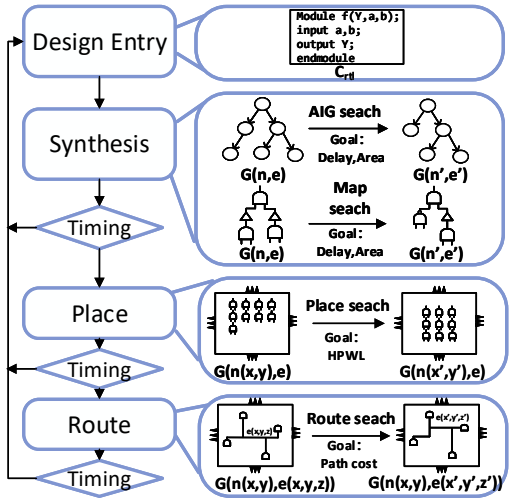

Figure 2: A design flow is essentially an in-stage search model. Each flow stage can be modeled as an individual in-stage search problem.

accurate. Secondly, our physically-aware synthesis framework extended the prior and-inverter-graph (AIG) representation based search to account for the whole design process by modeling the EDA flow as an integrated multi-stage search problem, where AIG-based optimization is only the beginning of a chain of optimizations. Lastly, the proposed flow is developed based on open-source tool suites and is made compatible with multiple searching strategies. Thus it is more portable compared to others.

III. DDPAS Framework

The whole EDA flow could be formulated as a NP-hard problem, given design and constraint, searching for the best QoR. However, the search space is divided into several hierarchies or stages, from upstream abstract representation to downstream detailed representation. The biggest problem faced by EDA algorithms is how to perform efficient search from exploding search space. For example, only in the placement stage, the state space of placing 1000 clusters of nodes on a grid with 1000 cells is of the order of 1000 (greater than 10^{2500}) [21]. The whole flow search space is then exponential of the state space in each stage. Thus efficient search is especially critical to enable physically-aware synthesis. In this section, we will discuss details about the proposed search framework.

A. In-stage Search Model

As shown in Fig. 2, a typical EDA flow from synthesis to route can be formulated as several in-stage search processes. The synthesis stage takes the RTL input C_{rtl}. Under the constrain of $Cons$, C_{rtl} is transformed into $C_{syn} = G(n, e)$ with synthesis recipe SR, where netlist is represented by graph $G(n, e)$. n (nodes) and e (edges) represent gates and wires. Synthesis $Syn : C_{rtl} \rightarrow G(n, e)$ not only executes logic equivalent transformation, but also searches for the best graph topology, node types and connections. Without losing generality, we use and-inverter-graph (AIG) to implement $G(n, e)$ for technology independent logic optimization. And Synthesis recipe SR is given from transformation and optimization steps within synthesis suite $Yosys$ [22] and ABC [23], including $rewrite$, $re-substitute$, $refactor$, $rewrite-z$, $resub-z$, $refactor-z$, $balance$. As an example, this AIG based state space could be represented as a search tree where details of tree components are listed in Table I. It should be noted that any search strategies can be paired with this search tree.

Table I: Technology independent logic synthesis search tree

Search Tree	Description
State	All logic satisfied AIG $G(n, e)$
Action	Seven AIG-based transformations
Successor function	Transformation by action algorithm
Initial state	Input RTL-level design C_{rtl}
Goal	Delay (an/or area)
Search tree root	Initial state
Tree node	State
Tree leaf	Last state in search path
Branching factor	Number of actions (seven)

In a similar way, each flow stage can be formulated as an in-stage search and represented by search tree. For example, technology mapping transforms $G(n, e)$ into $G(n', e')$, where $G(n', e')$ is a technology-dependent gate that maps the technology independent gate $G(n, e)$. The goal is to minimize delay and area. And the action is mapping. For placement and routing, each node and edge incorporate the physical location information. It transforms $G(n, e)$ into $G\big(n(x, y), e(x, y, z)\big)$. The goals are wire length and path cost. The actions are changing gates and wires locations.

B. Multi-stage Search Model

Based on the in-stage search model discussed in prior section, if we consider timing closure as an ultimate design goal, a design C with design constraint $Cons$, ran through synthesis and PnR, can be formulated as a continuous multi-stage search problem to find the most optimal C:

$$C = \underset{Cons}{argmin}\Big(Metric\Big(G\big(n(x, y), e(x, y, z)\big)\Big)\Big)$$

where $G\big(n(x, y), e(x, y, z)\big)$ is graph representation of c in any stage. $Metric$ is the evaluation function such as power, area, timing or congestions. In this work, we limit our scope to timing optimization only. Isolated local in-stage search could now be combined as a sequential multi-stage search.

As shown in Algorithm 1, search strategy in each stage S_i is the applied search algorithm, such as heuristic, simulated annealing and so on. Search goal of each stage $Goal_i$ is the actual design goal and converging condition. Total stages N_c equals to the number of whole EDA flow stages. Based on this model, it shows that cross-stage search requires a Depth First Search (DFS), which indicates that searched state in synthesis must be evaluated for timing closure after that the final node in the final stage is searched. However, a typical in-stage search is Breadth First Search (BFS) to guarantee that enough states are searched before entering next stages. When in-stage transformation or a state meets $Goal_i$, the searched state will transfer to downstream stage state (iterate loops in line 3). If the best node in the final stage is not found, it will iterate the outer loop (line 2). Although this approach is complete, the

Algorithm 1: EDA flow modeled as multi-stage search

Input: initial design C_{rtl}; search strategy (EDA algorithm) in each stage S_i; goal of each stage $Goal_i$; total stages N_c;
Output: searched design C_{output}; whether stage i is failure

current design $C_{current}$ =initial design C_{rtl};
while *not satisfy timing closure* **do**
  for $i = 1 \rightarrow N_c$ **do**
    initialize search tree from root $C_{current}$;
    while *there are candidates for expansion* **do**
      choose a leaf state node for expansion according to search strategy S_i;
      if *state node satisfy* $Goal_i$ **then**
        $C_{current}$ = state node corresponding design;
      else
        expand state node and add resulting nodes to search tree
    if *no candidate for expansion* **then**
      return stage i failure;
      $i = 0$;
  return $C_{current}$

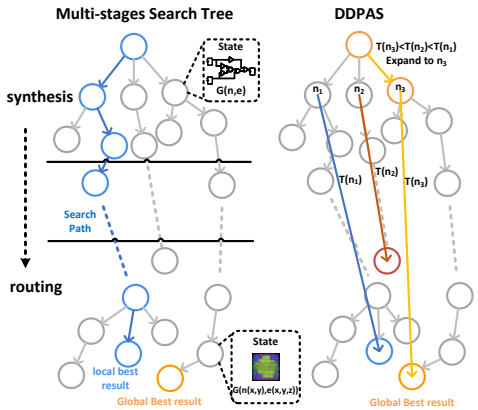

Figure 3: Delay driven physically aware synthesis (DDPAS) with informed search A* algorithm, offering efficient cross-stage informed search.

cost is huge as many computation resources and timing are wasted in later search path if the prior stage (e.g. synthesis) search path is not well performed. The worst situation occurs when there is a wrong search path in the first stage, DFS returns until it finishes searching for the final stage. This will incur long iteration loops.

C. Delay Driven Physically Aware Synthesis with Informed Search

According to the previous discussed multi-stage search model, whatever search strategy is applied, the final searched result is only local best of searched path, while the global best results might exist in other paths. This is illustrated in Fig. 3 (left). To perform more effective and efficient cross-stage search, we leverage the usage of A* algorithm, an informed search strategy that is widely used in computer science to perform efficient graph traversal and path search. It avoids expanding paths that are already expensive searches by expanding node that appears to be promising with respect to its informed value function $f(n) = g(n) + h(n)$, where $g(n)$ is cost-so-far to reach node n, and $h(n)$ is the estimated cost-to-go from node n.

Inspired by this, we propose a delay-driven physically-aware synthesis (DDPAS) algorithm that adapts the A* search algorithm. It is illustrated in Fig. 3 (right). Here we define our informed value function as $T(n) = S_{sta}(n) + S_{phy}(n)$, where $S_{sta}(n)$ is the total timing slack given by the timing analysis engine for current node n, mirroring the cost-so-far to reach node. $S_{phy}(n)$ is the total slack estimated in the following search path for node n, like estimated cost-to-go from current node. This function establishes the link from the first stage the last, which can potentially give oracle about future timing of best node of current searched path during synthesis, so that final timing could be estimated in synthesis. For example, $T(n)$ is given to candidate nodes n_1, n_2, n_3. n_3 is on the global best result search path, so n_3 with minimum $T(n)$ value is selected to expand. However, different from a typically graph traversal problem, here an accurate timing prediction model is required to provide a valuable informed guidance to the search.

Table II: Feature list for NLP-based timing prediction

Gate Features		
Feature	**Source**	**Size (Type)**
Gate delay	STA timing report	1 (float)
Gate load	STA timing report	1 (float)
Slew	STA timing report	1 (float)
Arrival time	STA timing report	1 (float)
Fanout	STA timing report	1 (int)
Global Features		
Feature	**Source**	**Size (Type)**
Max path delay between 2 reg	STA timing report	1 (float)
Max path depth between 2 reg	STA timing report	1 (float)
Die area	SDC file	1 (float)
Clock period	SDC file	1 (float)
Clock period × die area	SDC file	1 (float)

D. Inform Value Function with NLP-based Timing Prediction

Inspired by prior ML-assisted timing prediction work in the community, we propose a Natural Language Processing (NLP)-based slack prediction model to feed the inform value function. The model is designed to be light yet accurate so that it is able to provide fast timing response during the search. The features are listed in Table II, where gate features are based on attributes that are highly related to wire length after routing. Furthermore, global features are picked to reflect the optimizations that can potentially impact timing. It should be noted that we also incorporate die area and clock period as part of the feature lists as they also have direct effects on timing.

Gate-level netlist has no direct representation in vector space, which makes it less applicable in ML model. One way is to embed a whole logical paths in a low-dimensional space using NLP-based embedding. As shown in Fig. 4, a timing path could be treated as a chain of words, where each word represents a gate. We embed each gate with gate features listed in Table II. Embedded path is then concatenated with global features. To concentrate our efforts on flop to flop timing, the input and output paths that include IO ports are not considered here. We collect the critical path for each pair of start point and

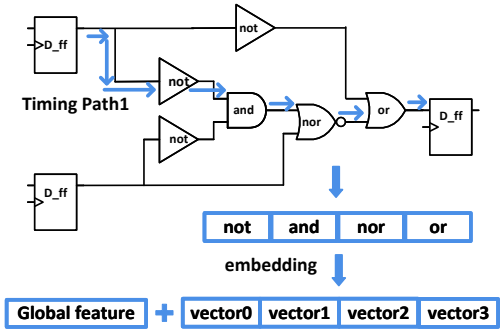

Figure 4: NLP-based timing prediction model.

end point. The increased delay and slacks of the paths from the same start point and end point after routing are extracted as labels. Regression models are trained to predict path delay and slack. Algorithm 2 shows how $T(n)$ is obtained with the trained NLP model. The accuracy of the model will be discussed in Section IV.

Algorithm 2: $T(n)$ **calculation**

Input: search state n corresponding STA timing report after synthesis and SDC file, trained NLP model $\mathbb{M}$
Output: inform value $T(n)$

```
parse and extract global feature from SDC file;
parse STA timing report;
for each critical timing path do
    if start point and end point are all registers then
        build gate feature ;
        input feature = global feature + gate feature;
        critical path slack after routing = M(input
          feature) ;
T(n)=0;
for each predicted timing path do
    if critical path slack after routing<0 then
        T(n) += critical path slack after routing
return T(n)
```

E. DDPAS-RL Algorithm

To verify the proposed DDPAS methodology , we combine the framework with reinforcement learning, which expands node by learning based sampling probability, rather than always choosing better node. Reinforcement Learning (RL) was introduced in logic synthesis recently [15], [21], [24]. We adapt a similar method to build a Markov process as DRiLLS [15]. The RL-enriched DDPAS framework (named DDPAS-RL) is shown in Fig. 5. In the dotted line box, similar to DRiLLS [15], RL agent give selected node and state feature is extracted from synthesis tool. But DRiLLS uses delay and area as reward to train a RL agent, while we replace it with physically-aware inform value function $T(n)$. For Actor-Critic Methods used in DDPAS-RL agent, critic network $\hat{q}_w(s,a)$ measure the taken action, actor network $\pi_\theta(s,a)$ selects the actions for current state. s and a represent the state and the action in Markov process, while θ and w represent the

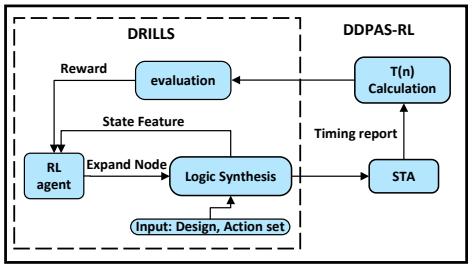

Figure 5: DDPAS-RL framework.

Table III: Benchmark statistics

Design	Design Information			Die Size (mm^2)	Clock Period (ns)
	#cell	#wire	#ff		
TinyRocket	23877	28956	3826	$\{0.525 \times 0.5 \rightarrow 0.925 \times 0.8\}$	$\{1.2 \rightarrow 2.03\}$
AES	16758	17172	530	$\{0.45 \times 0.45 \rightarrow 1.02 \times 0.92\}$	$\{0.8 \rightarrow 1.2\}$
gcd	362	387	35	$\{0.006 \times 0.006 \rightarrow 0.1 \times 0.1\}$	$\{0.285 \rightarrow 0.485\}$
ibex	15700	18226	1931	$\{0.35 \times 0.34 \rightarrow 0.55 \times 0.54\}$	$\{2.0 \rightarrow 2.4\}$
jpeg	56800	73514	4380	$\{0.7 \times 0.7 \rightarrow 1.1 \times 1.1\}$	$\{1.4 \rightarrow 1.8\}$
dynamic node	13445	13095	2303	(unseen design)	

trainable weights of each network. Gradient optimization for critic network is

$$\Delta w = \beta \delta \nabla_w \hat{q}_w(s_k, a_k)$$

$$\delta = R(s,a) + \gamma \hat{q}_w(s_{k+1}, a_{k+1}) - \hat{q}_w(s_k, a_k)$$

where β sets learning rate, γ is discount factor. Similarly for actor network:

$$\Delta\theta = \alpha \nabla_\theta (\log \pi_\theta(s,a)) \hat{q}_w(s,a)$$

where α sets learning rate.

IV. Evaluation Results

We implement the proposed framework in Python. The synthesis engine employed is the open-source Yosys suite [22], and the physical design is conducted with the latest OpenRoad flow [8] in 45nm open-source technology node. Our algorithms are trained and tested on a linux machine with 3.7GHz AMD 5900X CPU of 128GB RAMs. We choose widely-used benchmarks that offer diverse design behaviors to train and test the timing prediction model, and they are summarized in Table III. Benchmarks are divided into seen and unseen designs, where unseen designs include unique designs that are excluded from the training dataset or included designs in the training set but with complete different design constraints (such as area constraints and clock periods).

A. Timing Prediction Results for the Inform Value Function

To train the proposed NL-based timing prediction model, we constructed a comprehensive dataset by including 125 design cases with diverse design features. These design cases include five unique open-source designs (accounting for more than 2837k cells in total), each with 25 sets of unique design constraints as listed in Table III. The labels are obtained based on the STA reports after routing, while the features are

Table IV: Predicted slack and delay on benchmark designs with the proposed NLP-based model

Design	Slack (ns)						Delay (ns)					
	MAE	R^2	**MSE**	**ME**	**EVS**	**MedianAE**	**MAE**	R^2	**MSE**	**ME**	**EVS**	**MedianAE**
AES	0.016	0.992	0.0005	0.078	0.9923	0.0118	0.012	0.994	0.0003	0.0627	0.994	0.0089
gcd	0.0025	0.9988	1.157e-5	0.0117	0.9989	0.0021	0.0018	0.9986	5.09e-6	0.0046	0.9986	0.0014
TinyRocket	0.0264	0.9959	0.0013	0.169	0.9959	0.0189	0.023	0.9954	0.0012	0.1616	0.9954	0.0135
ibex	0.018	0.9977	0.0009	0.2699	0.9977	0.01	0.0156	0.9976	0.0007	0.1467	0.9976	0.0067
jpeg	0.0319	0.979	0.0024	0.211	0.9794	0.0163	0.0284	0.9751	0.0022	0.2349	0.9752	0.0118
Average	0.03	0.9947	0.0017	0.241	0.9947	0.0213	0.0262	0.996	0.0015	0.244	0.996	0.0165

Table V: Proposed timing prediction model vs. state of the art ML-based models

		Proposed	**ICCAD22 [25]**	**MICPRO22 [7]** (TabMLP Model)	**MICPRO22 [7]** (MPNN Model)	**ASPDAC21 [26]**	**TCAD22 [6]**
MAE	delay	**0.026**	2.64	9.49	6.76	Not reported, but accuracy<90%	0.11
	slack	**0.03**					
MSE	delay	**0.002**	0.16	1.39	0.96		not reported
	slack	**0.002**					
R^2	delay	**0.996**	0.956	0.79	0.87		0.91
	slack	**0.995**					

Table VI: Comparison of prediction accuracy (best R^2 from each work)

		Proposed	**ICCAD22 [5]**	**MICPRO22 [7]** (TabMLP Model)	**MICPRO22 [7]** (MPNN Model)	**TCAD22 [6]**
R^2	delay	**0.996**	0.956	0.79	0.87	0.91
	slack	**0.995**				

collected based on the STA report after the synthesis step. The data for training and testing is split based on a ratio of $8:2$.

In Table IV, we summarize the accuracy of the timing prediction (for feeding $T(n)$) on five designs for path slack and path delay. Various error metrics are checked, including Mean Absolute Error (MAE), R^2, Max Error (ME), Explained Variance Score (EVS), Median Absolute Error (MedianAE) and Mean Square Error (MSE). Results show that the proposed model is able to achieve decent prediction accuracy for path slack and delay on all tested designs. On average, it delivers close to 99.5% of R^2, which indicates great correlation. It is challenging to make fair comparisons against other timing prediction models based on the fact that their detailed algorithms and benchmark designs are proprietary. In addition, different work uses different metrics to measure errors. Thus, we pick the best reported R^2 value from recently published work [5]–[7] and summarized in Table VI. It shows that the proposed model from this work outperforms others in terms of the prediction accuracy. This model lays solid foundation for guiding the search by feeding the inform values.

B. DDPAS-RL Results

We compare our proposed DDPAS-RL algorithm with the state of the art Non-physical aware RL-assisted logic synthesis framework DRiLLS [15]. DRiLLS [15] has been open-sourced, thus we are able to test the same designs to obtain the baseline results. Similar to the evaluation methodology used in the previous section, we only change the synthesis methodology here and keep the physical implementation the same. Results are summarized in TableVII. It shows that DDPAS-RL achieves better timing and power in all design cases. On average, the proposed DDPAS-RL framework delivers 65.6% improvement in terms of TNS and 12.1% power saving after routing when compared against DRiLLS [15].

Table VII: Comparison of the proposed DDPAS-RL framework vs. state of the art RL-based framework DRiLLS [15]

Type	**Design**	**Method**	**TNS (ns)**	**WNS (ns)**	**Design Area** (um^2)	**Utilization (%)**	**Power (mW)**
Seen	AES (clk=0.8)	**DDPAS-RL**	**0**	**0.047**	**24475**	8%	**63.6**
		DRiLLS	-0.865	-0.037	24727	8%	90.8
Unseen	AES (clk=0.75)	**DDPAS-RL**	**-4.255**	**-0.067**	24971	8%	**84.5**
		DRiLLS	-7.076	-0.166	**23429**	8%	88.7
	AES (clk=0.7)	**DDPAS-RL**	**-9.54**	**-0.094**	**24670**	**8%**	**90**
		DRiLLS	-31.823	-0.241	32777	11%	136
	AES (clk=0.85)	**DDPAS-RL**	**-0.544**	**-0.047**	**22500**	**7%**	**75.6**
		DRiLLS	-4.395	-0.142	23737	8%	83.1
	gcd (clk=0.5)	**DDPAS-RL**	**-0.815**	**-0.056**	767	12%	**1.97**
		DRiLLS	-1.296	-0.085	**724**	**11%**	2.01
	gcd (clk=0.55)	**DDPAS-RL**	**-0.05**	**-0.006**	767	12%	**1.79**
		DRiLLS	-0.512	-0.036	**724**	**11%**	1.83
	gcd (clk=0.6)	**DDPAS-RL**	0	**0.019**	**690**	11%	**1.5**
		DRiLLS	0	0.015	700	11%	1.63
	gcd (clk=0.575)	**DDPAS-RL**	**0**	**0.01**	**719**	11%	**1.63**
		DRiLLS	-0.112	-0.011	724	11%	1.75

C. Runtime Improvement

By incorporating the physical awareness, the proposed DDPAS framework is able to efficiently find the optimal solution in the synthesis stage, which otherwise requires significant amount of backend runtime to check the results before knowing if the solution is optimal or not. This yields to the fact that DDPAS is able to close timing faster than a typical flow. By applying the same search method (whether informed greedy search or RL), DDPAS run time is the sum of synthesis search time and $T(n)$ calculation time. However, for a flow with no physical awareness, the run time is the sum of synthesis search time and the whole physical implementation time. For example, if we assume an extreme case when all synthesis results are needed to be verified before obtaining the optimal one in an AES design ($\sim$17k gates only), DDPAS improves the runtime efficiency by over 21983$\times$ compared to DRiLLS [15]. The difference here is few seconds vesus thousands of seconds for a simple design. For larger designs and tighter design constraints, this benefit will be scaled up.

V. Conclusions

In this work, we formulated the design flow as a multi-stage search problem, where informed search was applied to perform efficient yet effective search for finding the optimal solution during synthesis. A NLP-based routing-aware timing prediction model was developed to be incorporated into the inform value function to guide the search. The proposed framework was further paired with reinforcement learning. Evaluation results based on physical implementation demonstrated that the proposed synthesis framework was able to deliver better final QoR in terms of PPA when compared to its counterpart. The efficient search, coupled with physical awareness, was able to

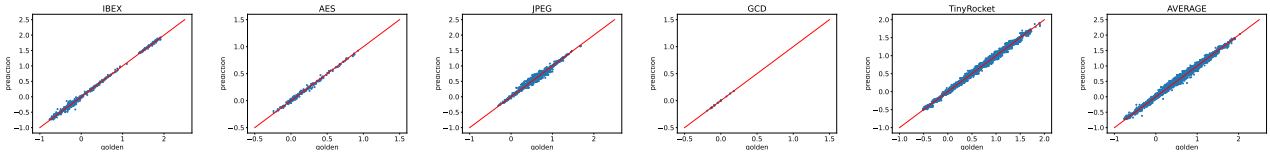

Figure 6: Slack correlation of $T(n)$. Predicted slack versus golden data on different benchmark designs.

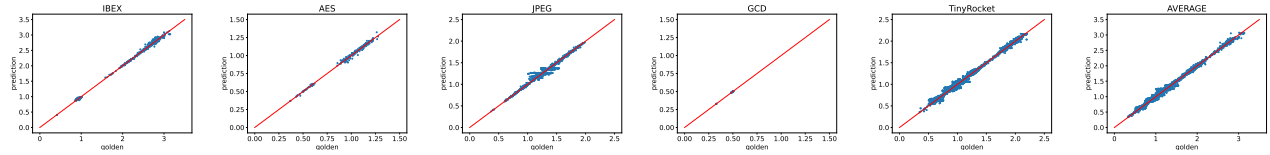

Figure 7: Delay correlation of $T(n)$.Predicted delay versus golden data on different benchmark designs.

reduce timing closure time by triggering more optimizations during the synthesis.

REFERENCES

[1] V. Bhardwaj, "Shift left trends for design convergence in soc: An eda perspective," *Vivek Bhardwaj. Shift Left Trends for Design Convergence in SOC: An EDA Perspective. International Journal of Computer Applications*, vol. 174, no. 16, pp. 22–27, 2021.

[2] L. E. Geralla, M. Guzman, and J. A. Hora, "Optimization of physically-aware synthesis for digital implementation flow," *International Journal of Engineering & Technology*, vol. 7, no. 2.11, pp. 31–34, 2018.

[3] M. Tatsuoka *et al.*, "Physically aware High Level Synthesis design flow," *Proceedings - Design Automation Conference*, vol. 2015-July, 2015.

[4] R. Kirby *et al.*, "Congestionnet: Routing congestion prediction using deep graph neural networks," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2019, pp. 217–222.

[5] D. S. Lopera and W. Ecker, "Applying GNNs to timing estimation at RTL," *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, 2022.

[6] Z. Xie *et al.*, "Preplacement Net Length and Timing Estimation by Customized Graph Neural Network," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4667–4680, 2022.

[7] D. Sánchez Lopera *et al.*, "Early RTL delay prediction using neural networks," *Microprocessors and Microsystems*, vol. 94, no. August, p. 104671, 2022. [Online]. Available: https://doi.org/10.1016/j.micpro.2022.104671

[8] "Openroad-flow-script," https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts.

[9] W. Lau Neto *et al.*, "Read your Circuit: LeveragingWord Embedding to Guide Logic Optimization," *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 530–535, 2021.

[10] "Synopsys fusion compiler," https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html.

[11] "Cadence genus ispatial," https://community.cadence.com/cadence_blogs_8/b/di/posts/ispatial-next-gen-flow.

[12] W. Gosti, S. P. Khatri, and A. L. Sangiovanni-Vincentelli, "Addressing the timing closure problem by integrating logic optimization and placement," *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*, vol. 2, pp. 224–231, 2001.

[13] L. Zhong and N. K. Jha, "Interconnect-aware high-level synthesis for low power," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, 2002, pp. 110–117.

[14] A. I. Reis and J. M. Matos, "Physical awareness starting at technology-independent logic synthesis," *Advanced Logic Synthesis*, pp. 69–101, 2018.

[15] A. Hosny *et al.*, "DRiLLS: Deep Reinforcement Learning for Logic Synthesis," *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, vol. 2020-Janua, pp. 581–586, 2020.

[16] W. Haaswijk *et al.*, "Deep learning for logic optimization algorithms," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–4.

[17] K. Zhu *et al.*, "Exploring logic optimizations with reinforcement learning and graph convolutional network," *MLCAD 2020 - Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, pp. 145–150, 2020.

[18] Y. V. Peruvemba *et al.*, "RL-Guided Runtime-Constrained Heuristic Exploration for Logic Synthesis," *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, vol. 2021-Novem, 2021.

[19] A. Grosnit *et al.*, "Boils: Bayesian optimisation for logic synthesis," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1193–1196.

[20] C. Yang *et al.*, "Logic synthesis optimization sequence tuning using rl-based lstm and graph isomorphism network," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 8, pp. 3600–3604, 2022.

[21] A. Mirhoseini *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[22] "Yosys," https://github.com/YosysHQ/yosys.

[23] "Abc:system for sequential logic synthesis and formal verification," https://github.com/berkeley-abc/abc.

[24] G. Pasandi, S. Pratty, and J. Forsyth, "Aisyn: Ai-driven reinforcement learning-based logic synthesis framework," *arXiv preprint arXiv:2302.06415*, 2023.

[25] D. S. Lopera and W. Ecker, "Applying GNNs to timing estimation at RTL," *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, 2022.

[26] W. Lau Neto, M. T. Moreira, L. Amaru, C. Yu, and P. E. Gaillardon, "Read your Circuit: LeveragingWord Embedding to Guide Logic Optimization," *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 530–535, 2021.