

# Adaptive Reconvergence-driven AIG Rewriting via Strategy Learning

Liwei Ni<sup>1,2,6</sup>, Zonglin Yang<sup>3</sup>, Jiayi Zhang<sup>4</sup>, Junfeng Liu<sup>5</sup>, Huawei Li<sup>1,2,6</sup>, Biwei Xie<sup>1,2,6</sup> and Xinqian Li<sup>2,✉</sup>

<sup>1</sup>Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

<sup>2</sup>Peng Cheng Laboratory, Shenzhen, China

<sup>3</sup>Shenzhen University, Shenzhen, China

<sup>4</sup>School of Computer Science, Peking University, Beijing, China

<sup>5</sup>SKLSDE, Beihang University, Beijing, China

<sup>6</sup>University of Chinese Academy of Sciences, Beijing, China

Emails: nlwmode@gmail.com, 2100271085@email.szu.edu.cn, zhangjiayi@pku.edu.cn,

liujunfeng@buaa.edu.cn, lihuawei@ict.ac.cn, xiebiwei@ict.ac.cn, lixq01@pcl.ac.cn

**Abstract**—Rewriting is a common procedure in logic synthesis aimed at improving the performance, power, and area (PPA) of circuits. The traditional reconvergence-driven And-Inverter Graph (AIG) rewriting method focuses solely on optimizing the reconvergence cone through Boolean algebra minimization. However, there exist opportunities to incorporate other node-rewriting algorithms that are better suited for specific cones. In this paper, we propose an adaptive reconvergence-driven AIG rewriting algorithm that combines two key techniques: multi-strategy-based AIG rewriting and strategy learning-based algorithm selection. The multi-strategy-based rewriting method expands upon the traditional approach by incorporating support for multi-node-rewriting algorithms, thus expanding the optimization space. Additionally, the strategy learning-based algorithm selection method determines the most suitable node-rewriting algorithm for a given cone. Experimental results demonstrate that our proposed method yields a significant average improvement of 5.567% in size and 5.327% in depth.

**Index Terms**—Rewriting, Reconvergence, Multi-strategy, Learning

## I. INTRODUCTION

Logic level optimization [1] is a crucial process in reducing the area and depth of a circuit to achieve better performance, power, and area (PPA). Rewriting, a highly flexible and efficient technique, is widely used for logic level optimization, providing a means to reduce circuit costs in many ways. The circuit is typically represented as a Directed Acyclic Graph (DAG), and the general framework for rewriting can be summarized in the following four steps:

- 1) **Subgraph computation.**
- 2) **Possible structures computation.**
- 3) **Cost evaluation.**
- 4) **Equivalent local replacement.**

Numerous works on rewriting have focused on the aforementioned steps, and the following provides a brief introduction: Regarding the *Subgraph computation* step, the emphasis is primarily on multi-inputs and single-output subgraphs. Works such as [2], [3], and [4] have primarily employed  $k$ -feasible cut enumeration [5] to compute subgraphs. However, due to the exponential growth of exhaustive cut enumeration with input size  $k$ , this method is

typically used when  $k$  is less than or equal to 4. On the other hand, works such as [6], [7], and [8] focus on computing the reconvergence cone to obtain large-inputs subgraphs. Once the subgraph is obtained, the next step is *Possible structures computation*, which involves determining the potential lower-cost structure candidates. In [9], possible structures are computed by determining their NPN classes through Negating and Permuting inputs or Negating the output. Similarly, [10] computes possible structures using the irredundant sum-of-product (iSOP) technique based on Boolean algebra. For minimal delay/area structures, [11] employs exact synthesis through Boolean chain encoding and SAT solvers. And [12] focuses on computing balanced structures using the AND-tree-balancing algorithm. The next step is the crucial *Cost evaluation*, which assesses the benefits of substituting the original structure with the new one. The depth can be estimated by the positive slack [13], while the area is estimated by maximum fanout free cone (MFFC) [14]. In the case of the Boolean matching problem [15], a transformation correspondence is established between the inputs and outputs of the original subgraph and the new structure. The *Equivalent local replacement* step ensures the equivalence after substituting the old subgraph with the new one in the original circuit. These four steps are executed for each node, typically applying the rewriting algorithm to all nodes in a topological order.

Since the problem of area and depth minimization in logic synthesis is NP-Complete, there are also some works to apply deep learning in logic synthesis. It is clear that the rewriting algorithm follows greedy approaches based on the local search method, and it can be easily trapped into local minimal that does not allow improving quality of results (QoR) [3]. However, a better result can be obtained through the design space exploration (DSE) method. Works such as [16] [17] [18] [19] focus on obtaining improved area or depth results by exploring the design space of logic optimization sequences through RL and Bayesian Optimization. Additionally, [20] and [21] optimize circuit size and mapping results using RL techniques.

Regarding the above brief introduction, we have three findings: 1) In the *Possible structures computation* step, the node-rewriting algorithms exhibit varying optimization

This work is supported in part by the Major Key Project of PCL (No. PCL2023AS2-3) and NSFC (No. 62090024).

capabilities; 2) Reconvergence-cone-based rewriting primarily optimizes using the iSOP method. However, not all cones are large, and other node-rewriting algorithms can be applied to suitable cones as well; 3) During each iteration of node-rewriting, the previously modified substructures can be overwritten and refreshed by the subsequent cones.

In this paper, we propose an adaptive reconvergence-driven AIG rewriting algorithm that addresses the aforementioned three cases by incorporating a strategy learning approach. Firstly, we enhance the original reconvergence-driven AIG rewriting algorithm to accommodate multiple node-rewriting algorithms. Secondly, as it can be challenging to determine the most suitable node-rewriting algorithm for a given subgraph, we formulate this as a classification problem that involves predicting the node-rewriting algorithm based on the subgraph’s features. To accomplish this, we propose a reinforcement learning (RL) algorithm to generate the training dataset for the classification problem. Our contributions can be summarized as follows:

- We propose an adaptive reconvergence-driven AIG rewriting algorithm against the three identified findings.
- We introduce a strategy learning approach for selecting the most suitable node-rewriting algorithm for a given subgraph by classification method.
- We formulate the multiple node-rewriting selection algorithm as the Markov decision process, and solve it by RL to generate the training dataset.
- We demonstrate the effectiveness of our proposed algorithm through experiments and comparisons with traditional methods.

The remaining sections of the paper are organized as follows: Section II presents the preliminary concepts and background related to the reconvergence-driven rewriting algorithm and reinforcement learning. Section III outlines the problem formulation of our proposed rewriting algorithm and provides a detailed description of the Markov decision process formulation. Section IV presents the framework of our proposed method, explaining the key components and steps involved. Section V presents the evaluation of our implementation through comprehensive experimentation, providing sufficient experimental results and analysis. Finally, section VI concludes this paper.

## II. PRELIMINARIES

### A. Boolean Network

A *Boolean network* is modeled as a DAG, where the node denotes a logic gate, input or output, and an edge denotes a signal wire between two logic gates. A node  $v$  has zero or more *fanins* and *fanouts*, and fanin denotes the number of incoming edges, and fanout denotes the number of outgoing edges. The *primary inputs* (PIs) are the nodes that have zero fanins, and the *primary outputs* (POs) are the nodes that have zero fanouts, we can intuitively think of PIs and POs are the inputs and outs of the circuit. The node  $v$  is *dead* iff it is an internal logic gate without any fanouts. A *transitive fanin/fanout cone* (TFI/TFO) of the node  $v$  is a subset of nodes that are reachable through the fanin/fanout edges of  $v$ . If there is a path from node  $v_i$  to  $v_j$ , then we can say  $v_i$  is in the TFI of  $v_j$ , and  $v_j$  is in the TFO of  $v_i$ . A *max fanout free cone* (MFFC) of the

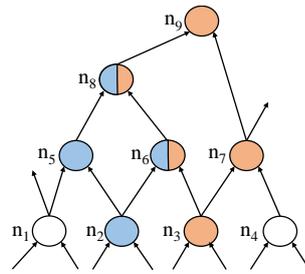


Fig. 1. The subgraph structure with reconvergence in the Boolean network.

node  $v$  is a subset of the TFI of  $v$ , such that each path from a node in this subset to the POs must pass through  $v$ . An And-Inverter Graph (AIG) is a Boolean network with the internal nodes referring to AND gate and the edges referring to the inverter-embed signal or not.

### B. Reconvergence-cone

The *reconvergence* refers to the paths starting at the fanout of a node  $n$  that will meet again before arriving at the POs. As shown in Fig. 1, it is a subgraph from a circuit, node  $n_2$ ’s fanouts meet at  $n_8$  before reaching POs, and as well as the path from  $n_3$  to  $n_9$ . The reconvergence cases are inevitable due to the logic sharing in multi-level Boolean networks. The *reconvergence-cone* technique works by identifying points in the circuit where multiple paths converge and then diverge again. The core idea is that there is a high probability of eliminable points on these paths in the reconvergence cone. These points are known as *reconvergence points*, and they often represent opportunities for optimization.

### C. Reinforcement Learning

RL is a subfield of Artificial Intelligence (AI) that focuses on developing algorithms and techniques for an agent to learn optimal behaviors through interaction with an environment. In RL, an agent learns by taking action in an environment and receiving feedback in the form of rewards or punishments. The agent’s goal is to learn a policy, which is a mapping from states to actions, that maximizes the cumulative rewards over time.

1) *Markov Decision Process (MDP)*: An MDP is a mathematical framework used to model decision-making problems in which an agent interacts with a dynamic environment. And it can be defined as the following:

$$MDP : (S, A, P, R, \gamma) \quad (1)$$

where  $S$  is the set of possible states (state space), and  $A$  is the set of possible actions (action space).  $P$  is the state transition probability matrix,  $P : S \times A \times S \rightarrow \mathbb{R}_{\in[0,1]}$ , where  $P(s, a, s')$  represents the probability of transitioning from state  $s$  to state  $s'$  when taking action  $a$ .  $R$  is the reward function,  $R : S \times A \rightarrow \mathbb{R}$ , where  $R(s, a, s')$  represents the immediate reward corresponding to the  $P(s, a, s')$ . And  $\gamma$  is the discount factor.

The objective in an MDP is to find a policy  $\pi(a|s)$ , which is a mapping from states to actions, that maximizes the expected cumulative reward. This is typically done by defining a value function  $V(s)$ , which represents the expected cumulative reward starting from state  $s$  and following

a particular policy  $\pi$ . The value function can be recursively defined using the Bellman equation:

$$V(s) = R(s, a, s') + \gamma \sum P(s, a, s')V(s') \quad (2)$$

By solving the equation (3), the optimal value function  $V^*(s)$  can be obtained, and the corresponding optimal policy  $\pi^*(s)$  can be derived by selecting the action with the highest expected rewards in each state:

$$\pi^*(s) = \max_{a'} \{R(s, a', s') + \gamma \sum P(s, a', s')V(s')\} \quad (3)$$

2) *Q-learning*: Q-learning [22] is a popular RL algorithm that aims to learn an optimal policy for a Q-agent to make decisions in an environment. In the Q-learning algorithm at any timestamp  $t$ , the Q-agent is at state  $s_t$ , it will enter the next state  $s_{t+1}$  by taking action  $a_t$ . The value function is represented by the matrix of Q-value for each pair of state and action,  $Q : S \times A \rightarrow \mathbb{R}$ . And the corresponding Bellman equation:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_{t-1}, a_{t-1}) + \alpha(R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')) \quad (4)$$

where  $\alpha \in [0, 1]$  is the learning rate. And there is also some proof in [20] [23] to tell that a finite MDP( $S, A, P, R, \gamma$ ) can be solved optimally using Q-learning with the update rule as in equation (4) while the learning rate  $\alpha$  is independent on both the state and action.

### III. PROBLEM DEFINITION

In this section, we present the fundamental problem definition of our proposed multi-strategy rewriting algorithm. Furthermore, we conduct an analysis of the design space exploration for multi-strategy rewriting to demonstrate the significance of solving for the optimal strategy sequence using RL. Lastly, we formulate the aforementioned problem as a Markov decision process (MDP).

#### A. Multi-Strategy Selection Rewriting Problem

The traditional rewriting problem of the Boolean network, as introduced in Section I, performs node-rewriting at each node to reduce the cost of the circuit. And it can be formulated as follows:

$$\begin{aligned} \min \quad & \sum_{i=0}^m cost(n'_i) - cost(n_i) \\ \text{s.t.} \quad & G' = \underbrace{NR(\dots(NR(n_i)))}_m, \quad n_i \in G \end{aligned} \quad (5)$$

where  $G$  refers to the input Boolean network, and  $G'$  refers to the optimized Boolean network and  $n'_i \in G'$ ,  $m$  is the node size of  $G$ , and the nodes generally follow a topological order. The term  $NR$  refers to a certain node-rewriting operation, which remains fixed once selected for all the nodes. And the objective function is to minimize the cost of the circuit.

As also mentioned earlier in Section I, our findings reveal that the reconvergence-driven rewriting algorithm has untapped optimization potential. To address this, we propose a multi-strategy selection rewriting algorithm, which can be formulated as follows:

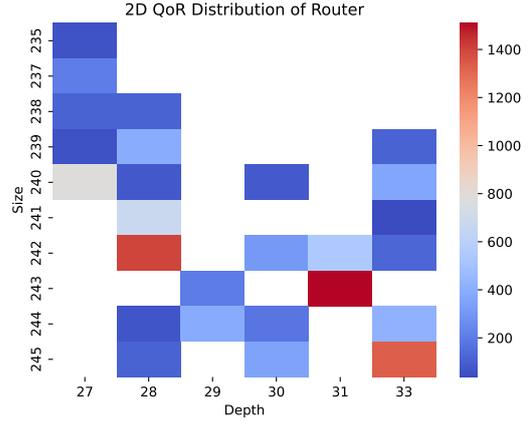


Fig. 2. The 2D QoR Distribution for the case “router”.

$$\begin{aligned} \min \quad & \sum_{i=0}^m cost(n'_i) - cost(n_i) \\ \text{s.t.} \quad & G' = \underbrace{NR_k(\dots(NR_j(n_i)))}_m, \quad n_i \in G, \end{aligned} \quad (6)$$

$NR_j, \dots, NR_k \in NRP$

where the  $NR_j$  and  $NR_k$  mean that we can use different node-rewriting algorithms at different nodes. the  $NRP$  represents the available rewriting algorithm pool, and the remaining definitions and objectives remain the same as in equation (5).

#### Opportunities:

- It offers a fusion approach that combines multiple node-rewriting methods;
- It has the potential to optimize the circuit further by leveraging the additional optimization space provided by different node-rewriting algorithms;

#### Challenges:

- How to decide the most suitable node-rewriting algorithm for a given cone?
- How to improve the generalization ability for the decision-making process for different AIG and reconvergence cones?

#### B. Design Space Exploration of the Rewriting

To demonstrate the effects of the multi-strategy rewriting method, we implemented a reconvergence-driven AIG rewriting algorithm using random strategies. These random strategies involve randomly selecting a node-rewriting algorithm from the algorithm pool for each rewritable node, taking into account the specific input size limitations of certain algorithms.

Using this novel optimization operator, we conducted an exploration of the design space by randomly shuffling the node-rewriting algorithms at each node. Specifically, we generated 10,000 rewriting results for the “router” case in the EPFL [24] benchmark using our reconvergence-driven AIG rewriting method based on random strategies.

The 2-D QoR distribution is presented in Fig. 2 as a heatmap. The x-axis represents the depth of the resulting AIG, the y-axis represents the size, and the color indicates

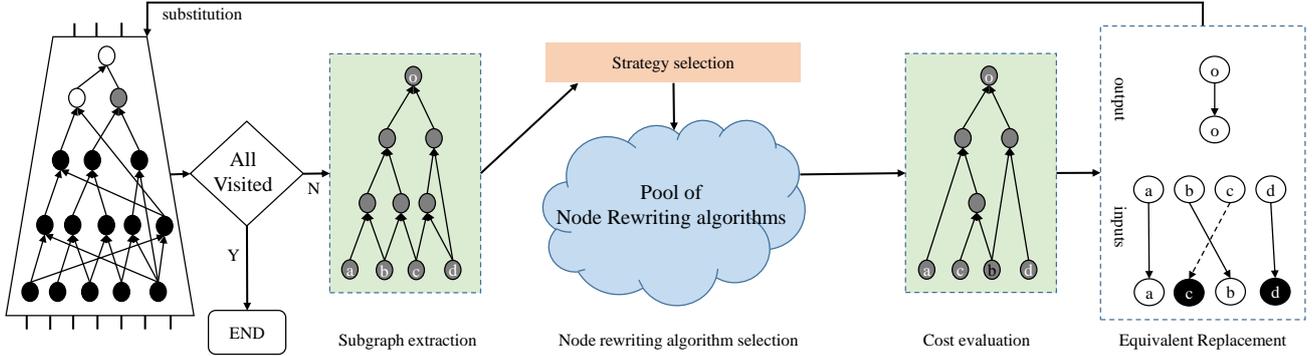


Fig. 3. The framework of the multi-strategies selection-driven node-rewriting method. These steps are corresponding to the traditional rewriting method: (1) Cone(subgraph) extraction; (2) Possible structures computation; (3) Cost evaluation; (4) Equivalent replacement. The main difference here is the multi-strategies selection at step (2) corresponding to the equation (6) in subsection III-A.

the count size. It is evident from the heatmap that the quality of results (QoR) is influenced by the node-rewriting algorithms, highlighting the importance of algorithm selection in achieving desired outcomes.

### C. Formulating Rewriting Selection as MDP

As mentioned in III-A, it tells that the node-rewriting algorithm is hard to be selected for the corresponding subgraph structure for the proposed multi-strategies-based problem. In this section, we model the multi-strategies selection rewriting-based optimization into an MDP. It should also be noticed that the outcome of a deterministic strategy sequence for the nodes is also deterministic, which simplifies the common probabilistic setting in typical MDP formulations. And we formulate this MDP problem in the rest of this subsection.

1) *State and its Space S*: As the next node's rewriting follows the result circuit graph of the previous node's rewriting, we define the state as the extracted features of the computed reconvergence cone, and they are as follows:

- *is\_critical*. It means the node is on the critical path or not, its range is Boolean{0,1};
- *input\_size*. It means the input size of the cone, and is sensitive to the node-rewriting algorithm, its range is [2,10];
- *node\_size*. It is the total size of this cone, the more node size with less input\_size may have more optimization space, its range is [2,16];
- *fanout\_size*. The total fanout size of the nodes in this cone. The MFFC is related to the fanout size;
- *positive\_edges*. It means the number of edges in two AND nodes without NOT gate;
- *negative\_edges*. It means the number of edges in two AND nodes with NOT gate;
- *max\_depth*. The max difference in depth between the root and leaves.

According to equation (6), we can make a simple assumption that each node-rewriting algorithm will output different results/states for a cone, and the state space is  $k^m$  in theory, where  $k$  is the number of node-rewriting algorithms,  $m$  is the number of times for node-rewriting which approximately equal to the number of internal nodes in AIG. In this paper, the state space's size is  $3^m$  in theory.

2) *Action Space A*: The action space is the pool of the following three node-rewriting algorithms. Since different node-rewriting algorithms have their own suitable input ranges, we also set some restrictions on them.

- NPN-based node-rewriting [2]. As the search space for the NPN problem is  $n!2^{n+1}$  ( $n$  is the input size), it is suitable when  $n \in [2, 4]$ ;
- Exact synthesis-based node-rewriting [11]. Exact synthesis can compute the optimal area/depth for the given cone with the price of time overhead. Cache-based exact synthesis [4] can significantly reduce the time overhead, and we set the input size  $n$  with the range of [2, 5].
- iSOP-based node-rewriting [10]. The iSOP method is the node-rewriting algorithm in the open source Logic Synthesis tool berkeley-abc [25] as the command "refactor", and we set the input size  $n$  with the same range of [2, 10].

3) *Rewards R*: The reward function is directly corresponding to the QoR improvement, and it includes two parts:

- *local\_reward*. It is the reward corresponding to the local gain after taking an action, and the local gain can be set to area, depth, or and-delay-product (ADP).
- *global\_reward*. It is the total local reward after the train of one episode, and we set it by the total area gain here.

## IV. PROPOSED FRAMEWORK

In this section, we provide the framework of our proposed adaptive reconvergence-driven AIG rewriting method via the strategy learning method. The core idea is to support the multi-node-rewriting algorithm for a given cone. And the implementation scheme is to classify the features we defined by a classifier, before that, the labeled dataset is generated by a Q-learning procedure.

### A. Framework Overview

The framework depicted in Figure 3 presents an overview of our proposed method. The input Boolean network is situated on the left side of the framework. It is processing at the gray node, subsequently, we apply the multi-node-rewriting selection algorithm to the given subgraph, and the

---

**Algorithm 1** Reconvergence-driven AIG rewriting via Multi-Strategy Selection
 

---

**Input:** Original AIG  $G$   
**Output:** Optimized AIG  $G'$

```

1:  $V \leftarrow \text{topological\_sorting}(G)$ 
2:  $\text{critical\_path\_computation}(G)$ 
3: for  $n$  in  $V$  do
4:   if  $\text{is\_logic\_node}(n)$  then
5:      $sg \leftarrow \text{reconvergence\_cone\_computation}(n)$ 
6:      $fe \leftarrow \text{feature\_extraction}(sg)$ 
7:      $op \leftarrow \text{strategy\_selection}(fe)$ 
8:      $psg \leftarrow \text{possible\_structure\_computation}(op, sg)$ 
9:      $price \leftarrow \text{cost\_evaluation}(sg, psg)$ 
10:    if  $price$  is acceptable: then
11:       $mp \leftarrow \text{match\_ports}(sg, psg)$ 
12:       $\text{equivalent\_replacement}(mp, sg, psg)$ 
13:    end if
14:  end if
15: end for
16:  $G' \leftarrow \text{remove\_dead\_nodes}(G)$ 

```

---

specific details of this process will be elaborated upon in Algorithm 1.

As depicted in Algorithm 1, the rewriting algorithm follows a topological order (line 1), and the critical path also should be marked for the influence of depth (line 2). Then we perform the multi-node-rewriting algorithm at each logic node (lines 3-15). For a given processing node  $n$ , the reconvergence cone  $sg$  will be computed first (line 5); Following this, the previously defined features  $fe$  are extracted to enable the evaluator to determine the appropriate node-rewriting algorithm  $op$  to be selected (lines 6-7). The possible structure candidates  $psg$  will be computed by  $op$ , and the cost  $price$  also will be evaluated whether it will result in the QoR improvement (lines 8-9). If the  $price$  is deemed acceptable (positive gains), the “equivalent local replacement” is executed to insert  $psg$  into the original circuit by ensuring correct port matching (lines 11-12).

We highlight two key aspects in our framework: (1) The “multi-strategy-based” rewriting method can expand the search space of possible structures by the algorithm pool; (2) The “strategy selection” process can be implemented using a learning approach. In the following two subsections, we will elaborate on how RL is employed to generate the training dataset and how the classification method is employed for “strategy selection”.

### B. Training dataset generation through Q-learning

We formulate the optimal “strategy selection” problem as an MDP as mentioned in subsection III-C. In this subsection, we use the Q-learning [22] method as the “agent” to learn a good policy of the circuits. The following provides a detailed description of the implementation:

1) *Strategy sequence*: To facilitate the calculation of the current state and next state during the topological node-rewriting procedure, we introduce the “strategy sequence” method, which fixes the strategy at each rewritable node. First, we create a mapping for each node-rewriting algorithm as follows:  $\{\{\text{“iSOP”}, 0\}, \{\text{“Exact synthesis”}, 1\}, \{\text{“NPN”}, 2\}\}$ . Each algorithm is assigned a value that corresponds to the index of the Actions  $\{a_0, a_1, a_2\}$ . The following Example 1 illustrates how the strategy sequence helps in computing the current state and next state.

---

**Algorithm 2** Q-learning based training procedure
 

---

**Input:** Input AIG  $G$   
**Output:** Policy matrix: Q-value table

```

1: Init Q-learning agent:  $q\_agent$  with  $\gamma, \alpha$ 
2: Init train parameter:  $st, Eps, decay, \epsilon, done \leftarrow \text{false}$ 
3: for  $ep$  in  $\text{range}(Eps)$  do
4:   while not  $done$  do
5:      $st \leftarrow \text{compute\_current\_state}(G)$ 
6:      $pas \leftarrow \text{get\_possible\_actions}(st)$ 
7:      $a \leftarrow \text{choose\_action}(q\_agent, st, pas)$ 
8:      $nst, r \leftarrow \text{apply}(st, a, \epsilon)$ 
9:     if  $nst$  is valid then
10:       $\text{update\_q\_table}(q\_agent, a, r)$ 
11:       $st \leftarrow nst$ 
12:     else
13:        $done \leftarrow \text{true}$ 
14:     end if
15:      $\epsilon \leftarrow \epsilon * decay$ 
16:   end while
17: end for

```

---

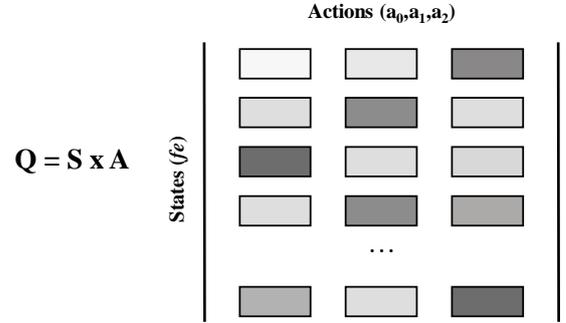


Fig. 4. The Q-table for the MDP solving by the Q-learning method.

**Example 1.** For an AIG network under the given strategy sequence “01002011”, the first eight rewritable nodes will use the corresponding node-rewriting algorithm to optimize its reconvergence cone. And the current state is the features of the 8th node’s reconvergence cone, the next state will be computed by the 9th (8+1→9) computed reconvergence cone.

2) *Q-learning-based policy learning*: Fig. 4 displays the Q-table used in our method, which illustrates the process of updating the rewards for the defined states and actions. Each state is represented by a tuple of the features  $fe$ , and the action space consists of three node-rewriting actions. Each rectangular node in the Q-table represents the reward  $Q(s_t, a_t)$  for a state under its corresponding action. The brightness of the rectangular node indicates the value of the reward, and the darker the brightness, the greater the value.

Algorithm 2 outlines the training procedure based on Q-learning. The  $q\_agent$  is initialized with the learning rate  $\alpha$  and discount factor  $\gamma$  according to the equation (4) (line 1). We also initialize several training parameters:  $st$  represents the current state,  $Eps$  denotes the batch size of the training,  $decay$  and  $\epsilon$  indicate the discount factor when selecting actions (line 2). For each training episode, the Q-table is updated according to the current state  $st$ , its possible actions  $pas$ , and the computed reward  $r$ , and then proceeds to the next state  $nst$  (lines 5-15). It should be noted that the possible actions  $pas$  are selected based on the restrictions

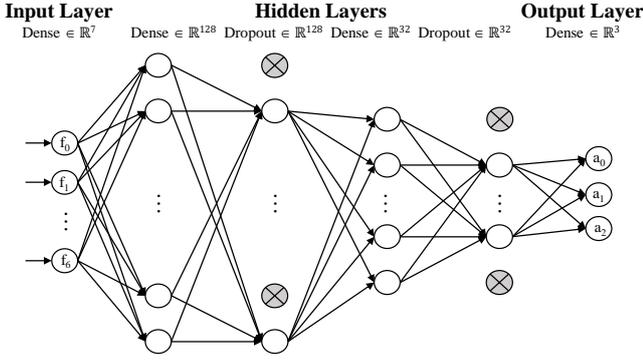


Fig. 5. The architecture of MLP for classification.

set in subsection III-C (line 6). If the next state is not valid, indicating that all rewritable nodes have been visited and rewritten, the procedure proceeds to the next episode (line 13).

### C. Classification

Due to the dimension of the features is only 7, and the Q-learning procedure will provide the data of the best action for its corresponding state, also as the feature, all of this undoubtedly motivates us to use classification methods to solve the problem of generalization. Hence, we employ a multi-layer Perceptron (MLP) [26] with dropout as a classification method to classify the features with the corresponding node-rewriting algorithm labels.

The architecture of the MLP is depicted in Fig. 5, consisting of the input layer, hidden layers, and output layer. The input layer has 7 dimensions, corresponding to the 7 features. Following the input layer, there are two hidden layers with dimensions of 128 and 32, respectively, each followed by a dropout layer. The dropout layer helps enhance the generalization ability of the neural network and mitigates overfitting. Finally, the output layer of the MLP is fully connected to the hidden layer, and a softmax activation function is applied to obtain the probabilities of the output classes.

## V. EVALUATION

The implementation of our proposed method “refactor-plus” is using the open source Boolean network library “mockturtle” [27] to reproduce the reconvergence-driven AIG rewriting method, also as the command “refactor”, of the berkeley-abc project in C++ language. Then we make it supports the multi-node-rewriting algorithm. And the Q-learning and MLP classification codes are implemented in Python language. All procedures run on an Intel(R) Xeon(R) Platinum 8260 CPU with 2.40GHz, 24 cores, and 128GB RAM. As for the benchmark, we perform the evaluation on the well-known EPFL combinational [24] benchmark, which consists of 10 arithmetic circuits and 10 control circuits, and the circuit sizes range from 174 to 214335.

### A. Convergence of Strategy Learning

For the convergence of strategy learning, there are two aspects to consider: the convergence of Q-learning for the training dataset and the convergence of the classification model for the features.

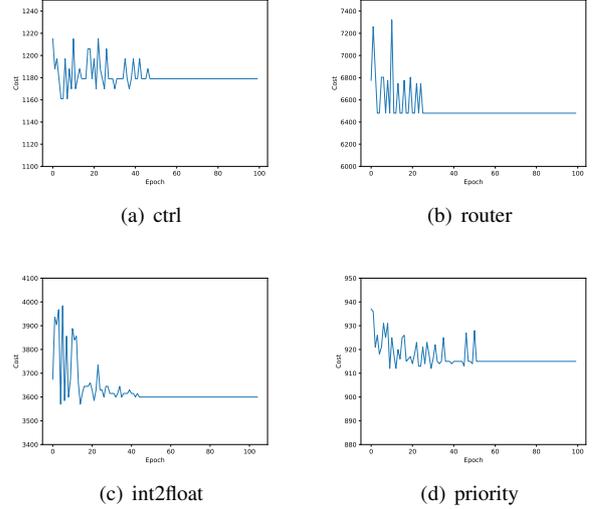


Fig. 6. The convergence of the Q-learning strategy selection method, and the y-axis represents the cost value which is negatively related to the reward value of the learning procedure, and the smaller, the better.

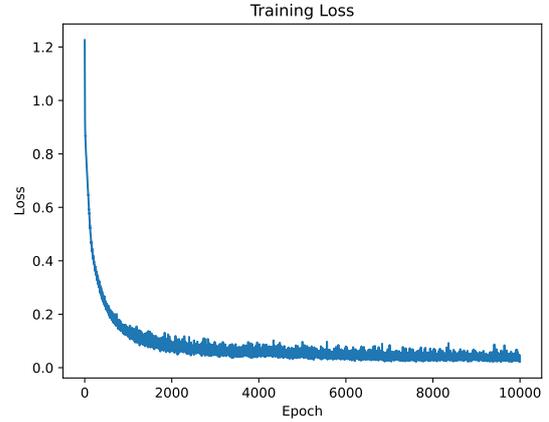


Fig. 7. Train Loss of the classification problem on the features through MLP method with 87% test accuracy.

1) *Convergence of Q-learning*: Fig. 6 demonstrates the convergence curve of the Q-learning training procedure over 100 episodes on 4 combinational circuits. The curve indicates that the Q-learning algorithm achieves convergence within 50 episodes, it suggests that the training process is effective in learning the optimal policy for selecting node-rewriting algorithms. Along with the Q-learning procedure converging, the classification procedure can utilize the well-trained dataset generated by Q-learning. Because the RL-based training dataset generation procedure is time-consuming, here we did not give the program running time, but the reader can according to the TABLE-I and SECTION-IV-B to estimate the runtime roughly.

2) *Convergence of classification*: Fig. 7 depicts the convergence of the classification procedure of features. All the trained dataset is obtained from the Q-learning step on the small cases. Since this is a multi-classification problem, the loss function employed is the “cross-entropy loss”. The dataset is split into an 80% training set and a 20% test set for model training and evaluation, respectively. After 10,000 training episodes, it shows that the loss function is

TABLE I. Comparison of the results on “refactor” and “refactor-plus”.

Circuits	origin		refactor				refactor-plus					
	size	depth	size	improv%	depth	improv%	time(s)	size	improv%	depth	improv%	time(s)
log2	32060	444	31521	1.681	444	0	2.536	<b>30814</b>	3.886	<b>423</b>	4.730	9.677
square	18484	250	18302	0.985	250	0	1.38	<b>18275</b>	1.131	<b>248</b>	0.800	2.051
adder	1020	255	1019	0.098	255	0	0.066	1019	0.098	255	0	0.486
sin	5416	225	5321	1.754	224	0.444	0.286	<b>5279</b>	2.530	<b>214</b>	4.889	0.95
div	57247	4372	56745	0.877	4372	0	2.718	<b>56550</b>	1.218	4372	0	23.265
hyp	214335	24801	<b>212341</b>	0.930	24801	0	23.872	212431	0.888	<b>24794</b>	0.028	301.767
max	2865	287	2865	0	287	0	0.154	2865	0	<b>226</b>	21.254	0.629
sqrt	24618	5058	23685	3.790	5058	0	0.379	<b>20155</b>	18.129	<b>5056</b>	0.040	8.107
multiplier	27062	274	26814	0.916	274	0	2.171	<b>25509</b>	5.739	<b>272</b>	0.730	6.112
bar	3336	12	3141	5.845	12	0	0.144	3141	5.845	12	0	0.545
priority	978	250	<b>854</b>	12.679	<b>227</b>	9.200	0.089	888	9.202	242	3.200	0.489
cavlc	693	16	690	0.433	16	0	0.116	<b>688</b>	0.722	16	0	0.517
arbiter	11839	87	11839	0	87	0	0.42	11839	0	87	0	2.886
i2c	1342	20	1338	0.298	20	0	0.09	<b>1308</b>	2.534	<b>18</b>	10	0.49
voter	13758	70	12681	7.828	63	10	0.677	<b>11363</b>	17.408	<b>70</b>	0	1.455
int2float	260	16	251	3.462	16	0	0.065	<b>237</b>	8.846	16	0	0.485
ctrl	174	10	143	17.816	9	10	0.065	<b>132</b>	24.138	<b>9</b>	10	0.394
dec	304	3	304	0	3	0	0.063	304	0	<b>3</b>	0	0.345
mem_ctrl	46836	114	46582	0.542	114	0	0.875	<b>46071</b>	1.633	<b>113</b>	0.877	13.729
router	257	54	246	4.280	54	0	0.075	<b>238</b>	7.393	<b>27</b>	50	0.455
AVE.	-	-	-	3.211	-	1.482	-	-	<b>5.567</b>	-	<b>5.327</b>	-

converging. And the overhead for inference time is low, and it is about 10 microseconds according to the statistical results. Furthermore, the test accuracy of 87% suggests that the model is able to generalize well to the unseen cone’s feature. The following experiments will demonstrate the effectiveness of our proposed algorithm.

### B. Comparison with traditional “refactor”

In this subsection, the experiments will depict that the multi-strategy selection rewriting through strategy learning is working well. And our solution for strategy learning is also successful. To showcase the performance of our proposed method “refactor-plus” compared to the traditional “refactor” method, we conducted experiments from the following four perspectives:

1) *Comparison by individual operator*: The performance of individual operator optimizations is presented in Table I. The results demonstrate that the “refactor” method achieves an average improvement<sup>1</sup> of 3.211% in size and 1.482% in depth. While the “refactor-plus” method achieves a higher improvement of 5.567% in size and 5.327% in depth. In most cases, “refactor-plus” outperforms “refactor” and effectively reduces both the size and depth of the circuits. In the best case scenario, the improvement of “refactor-plus” can reach 24.138% in size and 50% in depth. It also shows that “refactor-plus” can effectively reduce the depth while reducing the size than “refactor”. The time of “refactor-plus” is mainly caused by the three parts: (1)the selected algorithm’s time; (2) the data structure we implemented; and (3) the decision inference time. As the inference is low overhead, the main causes are (1) and (2).

2) *Comparison by “pre-balance”*: The “balance” command in the berkeley-abc tool is a powerful technique for reducing the depth of AIG circuits. Table II presents the results of applying the “refactor” and “refactor-plus” operators followed by the “balance” command. Despite the effectiveness of the “balance” operation, the “refactor-plus” still achieves an average improvement of 2.436% in size and 0.939% in depth compared to “refactor”.

3) *Comparison by “resyn2”*: The “resyn2” is a logic optimization flow in the berkeley-abc tool, and it is composed of the sequence of “balance, rewrite, refactor, balance, rewrite, rewrite -z, balance, refactor -z, rewrite -z,

<sup>1</sup>For improv% computation, Table I is computed by (origin - refactor/refactor-plus)/origin\*100; Table II, III, and IV are computed by (refactor - refactor-plus)/refactor\*100.

TABLE II. Comparison of the results of “pre-balance”.

Circuits	refactor+(balance)		refactor-plus+(balance)			
	size	depth	size	improv%	depth	improv%
log2	31400	410	<b>30706</b>	2.210	<b>409</b>	0.244
square	18086	250	<b>18067</b>	0.105	<b>248</b>	0.800
adder	1019	255	1019	0	255	0
sin	5297	186	<b>5258</b>	0.736	<b>184</b>	1.075
div	56729	4372	<b>56533</b>	0.346	4372	0
hyp	<b>211669</b>	24801	212115	-0.211	<b>24792</b>	0.036
max	2865	287	2865	0	<b>200</b>	30.314
sqrt	23681	5058	<b>20139</b>	14.957	<b>5056</b>	0.040
multiplier	26705	266	<b>25399</b>	4.890	<b>264</b>	0.752
bar	3141	12	3141	0	12	0
priority	<b>768</b>	<b>227</b>	805	-4.818	241	-6.167
cavlc	686	16	<b>684</b>	0.292	16	0
arbiter	11839	87	11839	0	87	0
i2c	1275	16	<b>1247</b>	2.196	16	0
voter	12474	<b>62</b>	<b>11244</b>	9.861	70	-12.903
int2float	233	15	<b>220</b>	5.579	15	0
ctrl	142	9	<b>130</b>	8.451	9	0
dec	304	3	304	0	3	0
mem_ctrl	46549	114	<b>45953</b>	1.280	<b>113</b>	0.877
router	246	27	<b>239</b>	2.846	<b>26</b>	3.704
AVE.	-	-	-	<b>2.436</b>	-	<b>0.939</b>

TABLE III. Comparison of the results of “resyn2”.

Circuits	resyn2 with refactor		resyn2 with refactor-plus			
	size	depth	size	improv%	depth	improv%
log2	29370	376	<b>29257</b>	0.385	<b>368</b>	2.128
square	<b>16623</b>	248	16636	-0.078	<b>247</b>	0.403
adder	1019	255	1019	0	255	0
sin	5039	177	<b>5034</b>	0.099	<b>172</b>	2.825
div	40772	<b>4361</b>	<b>40758</b>	0.034	4362	-0.023
hyp	<b>211330</b>	24794	211408	-0.037	<b>24786</b>	0.032
max	<b>2834</b>	204	2858	-0.847	<b>179</b>	12.255
sqrt	19437	<b>4968</b>	19437	0	4975	-0.141
multiplier	24556	262	<b>24379</b>	0.721	262	0
bar	3141	12	3141	0	12	0
priority	<b>676</b>	<b>203</b>	707	-4.586	217	-6.897
cavlc	662	16	<b>656</b>	0.906	16	0
arbiter	11839	87	11839	0	87	0
i2c	1162	15	<b>1143</b>	1.635	<b>14</b>	6.667
voter	9756	57	<b>8798</b>	9.820	57	0
int2float	214	15	<b>210</b>	1.869	15	0
ctrl	108	8	<b>107</b>	0.926	8	0
dec	304	3	304	0	3	0
mem_ctrl	45614	110	<b>44949</b>	1.458	<b>107</b>	2.727
router	<b>177</b>	<b>19</b>	196	-10.734	21	-10.526
AVE.	-	-	-	<b>0.079</b>	-	<b>0.999</b>

balance”. We use “refactor-plus” to replace the optimization of “refactor” and “refactor -z”. Most of the cases as shown in Table III show that the “resyn2” with refactor can get improvement, in general, we have average improvements for the “resyn2” with 0.079% in size and 0.999% in depth.

It should be noted that the “resyn2” is a heuristic approach, the integration of refactor-plus to “resyn2” is not necessarily the most appropriate. However, this experiment result still shows that refactor-plus brings the difference of optimization space.

4) *Comparison by “pre-mapping”*: We also evaluated the optimization capabilities of “refactor-plus” from the perspective of technology mapping. Table IV presents the results of applying the “refactor” and “refactor-plus” operators followed by the FPGA technology mapping command “if -K 6” in the berkeley-abc tool. It shows that the “refactor-plus” achieves an average improvement of 1.087% in edge, 2.617% in area, and 2.388% in delay compared to “refactor”.

Overall, these experiments demonstrate that “refactor-plus” is a powerful approach that effectively reduces the size and depth of the circuits, making it a valuable approach for Boolean network optimization.

TABLE IV. Comparison of the results of “pre-mapping”.

Circuits	refactor+(fpga-mapping)			refactor-plus+(fpga-mapping)					
	edge	area	delay	edge	improv%	area	improv%	delay	improv%
log2	<b>36397</b>	52225	76	36572	-0.481	<b>50600</b>	3.112	<b>73</b>	3.947
square	16518	34848	50	<b>16437</b>	0.490	<b>34451</b>	1.139	50	0
adder	1036	1666	51	<b>1022</b>	1.351	<b>1654</b>	0.720	51	0
sin	6491	9786	42	<b>6359</b>	2.034	<b>9146</b>	6.540	<b>41</b>	2.381
div	<b>76998</b>	84553	864	77016	-0.023	<b>84298</b>	0.302	864	0
hyp	197758	342434	4194	<b>197728</b>	0.015	<b>342162</b>	0.079	<b>4193</b>	0.024
max	3350	3365	56	<b>3141</b>	6.239	<b>3130</b>	6.984	<b>46</b>	17.857
sqrt	<b>21874</b>	34280	1023	21900	-0.119	<b>34116</b>	0.478	<b>1022</b>	0.098
multiplier	<b>27704</b>	45326	53	27818	-0.411	<b>43502</b>	4.024	53	0
bar	2688	<b>4476</b>	4	2688	0	4539	-1.408	4	0
priority	<b>1004</b>	<b>1279</b>	<b>30</b>	1064	-5.976	1393	-8.913	31	-3.333
cavlc	629	966	4	<b>619</b>	1.590	<b>952</b>	1.449	4	0
arbiter	15137	12415	18	<b>15135</b>	0.013	12415	0	18	0
i2c	1662	1565	4	<b>1613</b>	2.948	<b>1513</b>	3.323	4	0
voter	12493	38111	<b>15</b>	<b>12308</b>	1.481	<b>32187</b>	15.544	16	-6.667
int2float	250	289	3	<b>241</b>	3.600	<b>276</b>	4.498	3	0
ctrl	133	232	2	<b>131</b>	1.504	<b>194</b>	16.379	2	0
dec	684	397	2	684	0	397	0	2	0
mem_ctrl	54008	61254	<b>25</b>	<b>51361</b>	4.901	<b>60039</b>	1.984	28	-12
router	310	<b>386</b>	11	<b>302</b>	2.581	401	-3.886	<b>6</b>	45.455
AVE.	-	-	-	-	<b>1.087</b>	-	<b>2.617</b>	-	<b>2.388</b>

## REFERENCES

[1] G. D. Micheli., *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

## C. Discussion

This paper introduces a reconvergence-driven AIG rewriting method called “refactor-plus” and demonstrates its effectiveness through a series of experiments. The multi-node-rewriting algorithm can expand the searching space, while strategy learning can help to make a good decision for a given cone.

One of the notable strengths of “refactor-plus” is its ability to reduce both the size and depth of the optimized circuits. The expanded search space leads to better possible structure candidates, and the *is\_critical* and *max\_depth* is also considered as part of features when training the classifier. What this does is it allows “refactor-plus” to achieve an average improvement of 5.567% in size and 5.327% in depth.

The convergence of both the Q-learning-based RL and MLP-based classification procedures is essential for the overall effectiveness of the proposed method. First, the problem of “refactor-plus” can be formulated as an MDP, theoretically, there is an optimal solution. Therefore, we can get a good training dataset from the Q-learning-based RL to label each cone’s feature. With this good dataset, a good classification result is obtained with a test accuracy of 87%.

## VI. CONCLUSION

This paper proposes a new approach to AIG rewriting that utilizes two key techniques: multi-strategy-based rewriting and strategy learning. The adaptive reconvergence-driven approach expands the search space for possible structures by allowing for a multi-node-rewriting algorithms selection, which is in contrast to the traditional method. The strategy learning process involves two parts: generating a training dataset through RL and then classifying it using MLP.

Our evaluation results demonstrate that this new approach significantly improves the QoR of reconvergence-driven AIG rewriting when compared to the traditional method. However, there are still many areas for exploration, such as finding the optimal operator sequence for circuits and applying for multi-outputs-window-based rewriting algorithms. In the future, we plan to continue exploring these areas and refining our approach.

- [2] A. Mishchenko, S. Chatterjee, and R. Brayton, “DAG-aware AIG rewriting: a fresh look at combinational logic synthesis,” in *ACM/IEEE Design Automation Conference*, 2006, pp. 532–535.
- [3] N. Li and E. Dubrova, “AIG rewriting using 5-input cuts,” in *IEEE International Conference on Computer Design*, 2011, pp. 429–430.
- [4] H. Riener, A. Mishchenko, and M. Soeken, “Exact DAG-Aware Rewriting,” in *IEEE/ACM Design, Automation and Test in Europe Conference and Exhibition*, 2020, pp. 732–737.
- [5] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, “Combinational and sequential mapping with priority cuts,” in *IEEE/ACM International Conference on Computer-Aided Design*, 2007.
- [6] B. R. Alan Mishchenko, “Scalable logic synthesis using a simple circuit structure,” in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.
- [7] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, “Boolean Rewriting Strikes Back: Reconvergence-Driven Windowing Meets Resynthesis,” in *27th Asia and South Pacific Design Automation Conference*, 2022, pp. 395–402.
- [8] H. Riener, W. Haaswijk, and et al., “On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis,” in *IEEE/ACM Design, Automation and Test in Europe Conference and Exhibition*, 2019, pp. 1649–1654.
- [9] J. Zhang, L. Ni, and et al., “Enhanced Fast Boolean Matching Based on Sensitivity Signatures Pruning,” in *IEEE/ACM International Conference on Computer Aided Design*, 2021, p. 1–9.
- [10] T. Sasao and J. Butler, “Worst and best irredundant sum-of-products expressions,” *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 935–948, 2001.
- [11] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, “SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 871–884, 2020.
- [12] A. Mishchenko, R. Brayton, S. Jang, and V. Kravets, “Delay optimization using SOP balancing,” in *IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 375–382.
- [13] J. Vygen, “Slack in static timing analysis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 9, pp. 1876–1885, 2006.
- [14] J. Cong, C. Wu, and Y. Ding, “Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution,” in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999, pp. 29–35.
- [15] K.-C. Chen and J.-Y. Yang, “Boolean matching algorithms,” in *1993 International Symposium on VLSI Technology, Systems, and Applications Proceedings of Technical Papers*, 1993, pp. 44–48.
- [16] W. Haaswijk, E. Collins, and et al., “Deep Learning for Logic Optimization Algorithms,” in *IEEE International Symposium on Circuits and Systems*, 2018, pp. 1–4.
- [17] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, “DRiLLS: Deep reinforcement learning for logic synthesis,” in *ACM/IEEE Asia and South Pacific Design Automation Conference*, 2020, pp. 581–586.
- [18] C. Wang, C. Chen, D. Li, and B. Wang, “Rethinking Reinforcement Learning based Logic Synthesis,” 2022.
- [19] A. Grosnit, C. Malherbe, and et al., “BOiLS: Bayesian Optimisation for Logic Synthesis,” in *IEEE/ACM Design, Automation and Test in Europe Conference and Exhibition*, 2022, pp. 1193–1196.
- [20] G. Pasandi, S. Pratty, and J. Forsyth, “AISYN: AI-driven Reinforcement Learning-Based Logic Synthesis Framework,” 2023, arXiv:2302.06415.
- [21] G. Pasandi, S. Nazarian, and M. Pedram, “Approximate logic synthesis: A reinforcement learning-based technology mapping approach,” in *IEEE International Symposium on Quality Electronic Design*, 2019, pp. 26–32.
- [22] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, pp. 279–292, 1992.
- [23] F. S. Melo, “Convergence of Q-learning: A simple proof,” *Institute Of Systems and Robotics, Tech. Rep.*, pp. 1–4, 2001.
- [24] L. Amari, P.-E. Gaillardon, and G. De Micheli, “The EPFL combinational benchmark suite,” in *24th International Workshop on Logic & Synthesis*, no. CONF, 2015.
- [25] Berkeley Logic Synthesis and Verification Group, “ABC: A System for Sequential Synthesis and Verification,” <https://people.eecs.berkeley.edu/~alanmi/abc/>.
- [26] S. Pal and S. Mitra, “Multilayer perceptron, fuzzy sets, and classification,” *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.
- [27] M. Soeken, H. Riener, and et al., “The EPFL logic synthesis libraries,” Jun. 2022, arXiv:1805.05121v3.