SmartATPG: Learning-based Automatic Test Pattern Generation with Graph Convolutional Network and Reinforcement Learning

Wenxing Li[1,2], Hongqin Lyu[1], Shengwen Liang[1], Tiancheng Wang[1,3] and Huawei Li[1,2,3]
[1]*State Key Lab of Processors, Institute of Computing Technology, CAS, Beijing, China*
[2]*University of Chinese Academy of Sciences, Beijing, China*
[3]*CASTEST, Beijing, China*

ABSTRACT

Automatic test pattern generation (ATPG) is a critical technology in integrated circuit testing. It searches for effective test vectors to detect all possible faults in the circuit as entirely as possible, thereby ensuring chip yield and improving chip quality. However, the process of searching for test vectors is NP-complete. At the same time, the large amount of backtracking generated during the search for test vectors can directly affect the performance of ATPG. In this paper, a learning-based ATPG framework, SmartATPG, is proposed to search for high-quality test vectors, reduce the number of backtracking during the search process, and thereby improve the performance of ATPG. SmartATPG utilizes graph convolutional network (GCN) to fully extract circuit feature information and efficiently explore the ATPG search space through reinforcement learning (RL). Experimental results indicate that the proposed SmartATPG outperforms traditional and artificial neural network (ANN)-based heuristic strategies on most benchmark circuits.

1 INTRODUCTION

Automatic test pattern generation (ATPG) has served as an essential procedure dedicated to searching for effective test patterns which detect as many faults as possible in circuits [1]. The pivotal goal of ATPG is to attain a compact test vector set with high fault coverage, a task that becomes more challenging and indispensable with the escalating complexity of chip design [2]. Many ATPG algorithms have been proposed, such as D-algorithm [3], PODEM [4], FAN [5], and other methods. These methods are mainly designed based on traditional heuristic strategies. As the scale and complexity of integrated circuits continue to climb, the inherent weakness of traditional heuristic strategies is amplified, leading to prolonged execution times and unsatisfied fault coverage.

The limitations of traditional heuristic strategies on large-scale integrated circuits have motivated researchers to seek better ways to enhance ATPG's ability. Fortunately, recognizing the capability of deep learning (DL) algorithms to automatically learn hierarchical representations from data, researchers attempt to apply DL to ATPG tasks for high test coverage and computation efficiency [9–12]. Specifically, in DL-based ATPG algorithms, DL models, particularly artificial neural networks (ANNs), are trained on vast datasets of circuit information to guide the decision-making process in structural ATPG algorithms, primarily focusing on PODEM in existing works. This has resulted in significant improvements in reducing backtrack numbers. However, the existing DL-based ATPG algorithms heavily rely on the collected data, and the data's quality directly affects the model's quality. In addition, these solutions treat the search process of test patterns as a classification or regression problem [9–12], which cannot capture changes in circuit state.

In fact, ATPG is more suitable for being treated as an optimization problem, with the objective of finding effective test patterns for fault detection in digital circuits based on the context of the circuit's state. Reinforcement learning (RL) emerges as a promising solution to solve such combinatorial optimization problem [13]. With its capacity to perceive environmental changes through trial-and-error and feedback, RL's efficacy in capturing circuit characteristics has the potential to empower ATPG in navigating intricate search spaces and adapting to dynamic circuit states more flexibly. This adaptability could enable ATPG to discover optimal or near-optimal search strategies effectively.

To harness the full potential of RL and integrate it effectively into ATPG, we primarily tackle two key challenges. On the one hand, we transform the test pattern search process of ATPG into a Markov decision process (MDP) and define the state space, action space, state transition, and reward function corresponding to the RL task. To ensure that the agent obtained through RL training effectively guides the backtrace path selection, we meticulously design the reward function to align RL optimization with the path-searching problem. Another challenge lies in algorithm scalability. With modern designs becoming increasingly complex, representing the state space of a larger-size problem becomes a scalability bottleneck. Recognizing that the circuit netlist is inherently a graph, the ATPG process essentially involves searching for values that meet the specified requirements for designated nodes. However, existing methods have not fully considered the circuit's topology structure, and RL itself doesn't inherently capture topology representation. To bridge this gap, we employ a graph convolutional network (GCN) to capture both the topology of the circuit and additional relevant information, to represent the formation of a large graph into low-dimensional vectors for downstream tasks.

To this end, we propose a learning-based ATPG framework, SmartATPG, equipped with GCN and RL algorithms. The major contributions are listed as follows.

- We innovatively use RL algorithms to tackle the ATPG problem, which utilizes the trained agent to select the optimal path during backtracing.
- We represent the circuit netlist as a graph and utilize GCN to generate node embeddings, enabling the transfer of knowledge between different netlist topologies.
- In order to derive the RL agent to comprehend the circuit environment fully, we introduce the random network distillation (RND) mechanism into the framework to enhance the agent's exploration ability.
- Experimental results demonstrate the proposed SmartATPG outperforms traditional and ANN-based heuristic strategies on the majority of benchmark circuits.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656526

2 BACKGROUND

2.1 Automatic Test Pattern Generation

Mainstream ATPG methods include structural ATPG [3–6] and Boolean satisfiability (SAT)-based ATPG [7, 8]. Boolean satisfiability (SAT)-based ATPG algorithms formulate the test pattern generation problem as a Boolean satisfiability problem, where the goal is to find a satisfying assignment to the Boolean variables that represent the circuit's inputs and outputs [8]. However, the effectiveness of transforming the ATPG problem into a SAT instance significantly relies on the quality of the encoding. Different from SAT-based ATPG algorithms, structural-based ATPG algorithms are performed on the circuit directly. Common structural-based ATPG algorithms include methods like path-oriented decision making (PODEM) [4] and its variants [5, 6]. Taking the classical structural-based ATPG algorithm PODEM as an example, its process is as follows:

First, the fault is activated by modifying the status of the fault location to the opposite of the fault value. Then, the backtracing step starts from the first objective composed of the fault location and its state. A predecessor node with an unknown status is selected for backtracing toward the input direction until a primary input (PI) is reached. The PI is assigned a value to satisfy the objective's state. A conflict check determines if there is a discrepancy between this assignment and the current circuit state. In case of a conflict, the assignment is revoked, triggering the exploration of alternative assignment schemes and initiating the backtracking process. If there is no conflict, the next objective is chosen for further backtracing, identifying a new PI and assigning a value. It is essential to note that, to propagate the fault effect to a primary output for fault detection, new objectives are consistently generated. This iterative process continues until a test vector capable of detecting the current fault is found, or the search space is exhausted, ultimately concluding that the fault under test is undetectable.

The selection of an optimal path during backtracing significantly impacts ATPG performance. Instead of randomly selecting the backtrace path, it is preferable to employ heuristic strategies for choosing backtrace paths based on available choices. For instance, PODEM utilizes Distance [4], SCOAP [16], and COP [17] to guide the backtracing process. Nevertheless, when dealing with very large-scale circuits, ATPG based on these traditional heuristic strategies requires excessively long execution times.

2.2 Deep Learning-based ATPG

Although DL algorithms have been extensively explored to address various problems encountered in chip design, such as high-level synthesis (HLS), floorplanning, placement, and routing [18], the application in ATPG is still in its early stages. Existing DL-based ATPG methods are predominantly built upon structural-based ATPG algorithms. These approaches often leverage ANNs to replace traditional heuristic strategies in guiding the backtracing decision-making process. Roy et al. integrated ANN with one hidden layer into PODEM to guide backtracing by prioritizing the available choice. This approach has shown promise in reducing the number of backtracks, thereby enhancing the performance of ATPG [10]. The same team also introduced a structured training methodology incorporating multiple heuristics [11]. Subsequently, to further enhance the ATPG efficiency, principal component analysis (PCA) is leveraged to pre-process the training data to reduce the ANN complexity [12]. Despite the benefits gained from using ANN, current approaches tend to view ATPG exclusively as a classification or regression problem, thereby imposing limitations on unlocking the full potential of DL.

2.3 Reinforcement Learning

Different from DL, reinforcement learning (RL) is a machine learning paradigm that involves an agent learning to make decisions through interactions with an environment, guided by principles from behavioral psychology [13]. In the context of RL, a formalization commonly used is the Markov decision process (MDP) as illustrated in Figure 1. At each time step t, the agent initiates the process in state s_t, takes an action a_t, transitions to a new state s_{t+1}, and receives a reward r_t from the environment. By repeating episodes, comprising sequences of states, actions, and rewards, the agent's trajectory is fully characterized. The primary goal in an MDP is to determine a policy π, represented as a mapping from the state space to the action space ($\pi : S \rightarrow A$). The objective of this policy is to maximize the cumulative rewards. Different algorithms have been proposed for training policies in RL, such as Q-learning [14], policy gradient [13], proximal policy optimization (PPO) [15], and so on.

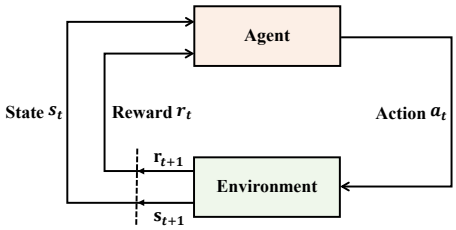

Figure 1: The Agent-Environment interaction in an MDP.

2.4 Graph Convolutional Network

Graph-based learning is an emerging paradigm in machine learning with diverse applications [19]. Before undertaking a specific task, it is essential to obtain representations of the nodes. These representations not only are based on the nodes' individual attributes but also incorporate the structural information of the local neighborhood. The resulting embeddings can then be inputted into downstream tasks for further processing. Graph convolutional networks (GCNs) are a type of neural network architecture specifically designed for processing and analyzing graph-structured data.

GraphSAGE [20] leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data. Instead of training individual embeddings for each node, this method learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood.

3 PROPOSED APPROACH

3.1 Task Formulation

In this paper, we target the ATPG problem, aiming to search for high-quality test vectors in a vast search space. The primary objective of ATPG is to efficiently generate compact test vector sets with high fault coverage. In addition, it should ensure scalability to handle large-scale circuit designs without sacrificing performance.

To align the objectives of ATPG with RL's goals of maximizing cumulative rewards, we meticulously designed the following MDP tailored to the needs of solving the ATPG problem.

1) **State Space**: In ATPG algorithms like PODEM, any decision on a primary input is made by choosing a sequence of lines to backtrace from an objective line. When detecting all faults in the circuit, this process potentially encounters all possible lines in the netlist. Therefore, for the RL algorithm to fully understand and simulate the circuit environment, all possible lines should be included in the state space.

2) **Action Space**: During the searching for backtrace paths, when the agent encounters a node with multiple fan-in paths, its choice of fan-in path crucially affects the state entered and the reward received. Please note that the fan-in sizes of different nodes may not be consistent. Therefore, the action space of each node is determined by its fan-in size.

3) **State Transition**: During each backtracing step, given the current line (current state), choosing any fan-in (action) will cause the agent to move to another line (next state).

4) **Reward Mechanism**: Backtracking during the test vector search process will have a great impact on the performance of ATPG, which will not only increase test time but also reduce fault coverage because of the waste of runtime. We meticulously design the reward to ensure that, while maximizing the reward, there is a reduction in backtracking.

3.2 SmartATPG Framework

Before diving into the design details, we introduce the overall flow of our learning-based ATPG with GCN and RL, SmartATPG, as illustrated in Figure 2. Unlike traditional or ANN-based heuristic strategies, SmartATPG extracts features from the netlist graph before utilizing RL policies to guide the path selection during backtrace, taking the circuit's topologies into account. Considering PODEM is well-suited for modeling as an MDP, and most existing ML-based ATPG methods are implemented based on this method, we instantiate our approach in PODEM [4].

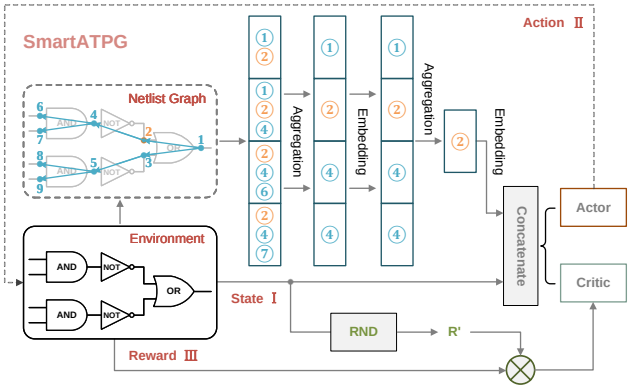

Figure 2: Overview of the SmartATPG framework.

The proposed end-to-end network architecture comprises three essential components: First, circuit structure learning (Section 3.2.1): This involves applying GCN to generate node embeddings from the netlist graph, facilitating subsequent RL in acquiring more comprehensive state information. Second, search strategy learning (Section 3.2.2): This component employs an online RL algorithm to solve the path-searching problem in ATPG. Third, the agent boosting (Section 3.2.3): To enable the RL agent to comprehend the circuit environment fully, the RND is incorporated to enhance the agent's capacity for exploration. Further details will be provided in the following sections respectively.

3.2.1 ***Circuit Structure Learning*** To capture additional information, such as the topology structure of the circuit under test (CUT), we represent the circuit netlist as a graph and extract features through a GCN. Specifically, this representation is realized by modeling the circuit netlist as a directed graph, where the circuit's lines are mapped as nodes of the graph, and the interconnections between these lines are delineated as the edges of the graph. In this graph-based representation, the attributes of a node, which represent the corresponding line in the circuit netlist, are characterized by logic gate type, logical level, fan-out count, and SCOAP measurements ($C0, C1, O$). Given the netlist graph, the GCN is implemented similarly to GraphSAGE to capture the intricate relationships and properties within the circuit. The node embedding is generated through a multiple-layer network involving aggregation and encoding, which can be formulated as follows.

$$h^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{h^{k-1}_u, \forall u \in \mathcal{N}(v)\}) \quad (1)$$

$$h^k_v \leftarrow \sigma(W^k \cdot \text{CONCAT}(h^{k-1}_v, h^k_{\mathcal{N}(v)})) \quad (2)$$

Where in Equation (1), h^{k-1}_u denotes embeddings of node u at layer $k-1$, AGGREGATE_k represents the aggregation function applied to node v and its neighborhood $\mathcal{N}(v)$. Equation (2) is the encoding operation, comprising an embedding projection parameterized by the weight matrix W^k and a non-linear activation σ.

Through these processes, the GCN module captures and fuses both the local and neighboring features at each layer. In the end, the embedding for the entire graph is calculated as the mean of all node embeddings at the final graph convolution layer, which is then fed into the subsequent search strategy learning module.

3.2.2 ***Search Strategy Learning*** Utilizing the extracted graph information and state details, the RL agent is trained to guide the path selection during backtracing, informed by the learned search strategy. We first define the states, actions, and rewards within circuits to empower the RL agent. Then, we outline the learning process of the RL agent.

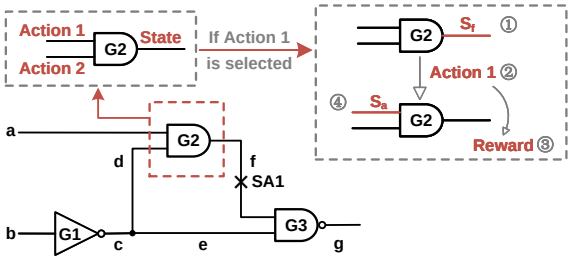

Figure 3: MDP state transitions in the CUT.

State design. The line traversed during PODEM backtracing is defined as the current state, encompassing information about the line's logic gate type g, logical level l, fan-out count n, SCOAP measurements s, and a mask vector m. Each state can be represented by a vector $[g, l, n, s, m]$. The mask vector enables the agent to learn to take into account the constraints imposed by the mask determined by the PODEM algorithm when making decisions. The length of the mask is determined by the maximum fan-in of the nodes in the CUT. The initial value of the mask is set based on the fan-in count of each node, with 1 for feasible operations and 0 for infeasible operations. Furthermore, for the fan-in whose previous set state has already met the expected logical state of the target node, its mask bit will transition from 1 to 0 to block the corresponding backtrace path. For example, consider a node with 3 fan-ins in a circuit where the maximum number of fan-ins is 4; the initial mask for this node would be set as [1, 1, 1, 0]. If the first fan-in satisfies the required logic state of the target node, the initial mask transitions from [1, 1, 1, 0] to [0, 1, 1, 0].

Action design. During backtracing, when the agent encounters a node with multiple fan-ins, the selected fan-in corresponds to the action it performs. In Figure 3, line f has two fan-in paths, which

means the agent has two possible actions in state s_f (Fig. 3 ①). For example, the agent can select action 1 (Fig. 3 ②) to evaluate the reward and make the agent enter the next state s_a (Fig. 3 ④). However, in practical circuit scenarios, the number of fan-in paths can differ among nodes, leading to a dynamically changing action space for the agent across various states. To address this, we use the initial mask vector to assist the agent in making more accurate action choices. This strategy ensures that the agent makes decisions only within the range of valid actions.

Reward design. In order to enable the agent to choose the optimal path during backtracing and minimize the number of backtracks, we design the following reward function:

$$Reward = \begin{cases} -0.1, \text{ if } not\ PI \\ 10 - \alpha * e^{\beta(backtrack_{num}+visit_{num})}, \text{ if } PI\ but\ not\ done \\ 100, \text{ if } done\ and\ detected\ fault \\ -100, \text{ if } done\ but\ undetected\ fault \end{cases} \tag{3}$$

In Equation (3), the negative reward (-0.1) before reaching a PI represents the potential gain for the agent before reaching a PI. This design strategy aims to encourage the agent to choose shorter paths and quickly reach a PI. Once the agent reaches a PI, the reward mechanism involves three factors (lines 2-4 of Equation (3)). The agent first receives an immediate reward (+10), indicating a successful location of a PI. However, if the located PI leads to backtracking, a penalty is imposed by $backtrack_{num}$ to encourage the agent to bypass paths that lead to backtracking. Additionally, to avoid the agent repeatedly visiting PIs that do not lead to backtracking, artificially increasing the total reward per round, we consider the number of times an agent visits a PI, denoted as $visit_{num}$, in the penalty mechanism to ensure that the total reward for a "game" round remains consistent with the number of backtracks.

In this reward mechanism, two crucial hyperparameters, α and β, play a vital role. A larger α value ensures that the agent receives clear feedback once a penalty is applied. A smaller β value ensures that the penalty's exponential growth rate is relatively slow, adapting to the characteristics of the PODEM algorithm, where producing a low value of $backtrack_{num}$ and $visit_{num}$ at a PI point is common.

Ultimately, when the agent successfully acquires a set of valid test vectors, thereby detecting the current fault, it receives a substantial reward (+100). Similarly, if the agent fails to acquire a test vector set for detecting the fault after reaching a preset limit of backtracks, it incurs a significant penalty (-100). The high rewards and penalties aid the agent in comprehending the human-defined concepts of "task completion" and "incomplete task."

Learning process. As shown in Figure 2, we first concentrate the node embeddings and the designed state representations into a 1-D vector. This consolidated vector is then used as the input for both the Actor, which facilitates action decisions, and the Critic, which estimates the strategic value at each step. The Actor parameterized with θ outputs the probability distribution over actions $\pi_\theta(a|s)$. While the Critic C_ϕ calculates the estimated value based on the current state s_t and the next state s_{t+1}. The Actor decides on actions, and the Critic assesses their outcomes as feedback. At the same time, the reward r_t is generated by the environment according to Equation (3). Following this, an estimation of the advantage is calculated by:

$$A_t(s_t, a_t) = r_t + \gamma C_\phi(s_{t+1}) - C_\phi(s_t) \tag{4}$$

The Actor's loss is determined by the expectation of the cumulative reward, as indicated in Equation (5), whereas the Critic's loss, derived from the mean squared error, is depicted by Equation (6):

$$L^{CLIP}_{\pi_\theta} = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\left[\rho_t(\pi_\theta, \pi_{\theta_k})A_t, clip(\rho_t(\pi_\theta, \pi_{\theta_k}), 1-\epsilon, 1+\epsilon)A_t\right]\right] \tag{5}$$

$$L(\phi) = \mathrm{E}_{s_t, s_{t+1}, r_t}\left[A_t(s_t, a_t)^2\right] \tag{6}$$

Then, the Actor and Critic are updated utilizing the gradient algorithm to minimize the loss, with the PPO algorithm applied to control the model update amplitude, thereby safeguarding the model's stability [15].

3.2.3 ***The Agent Boosting*** While the interaction between the agent and the environment is crucial for RL algorithms, in circuit environments with high-dimensional state spaces, the vastness of the state space adds complexity to this interaction. Furthermore, through reward design, we find that the reward signals during the search for test vectors are relatively sparse. Therefore, we incorporate random network distillation (RND) as an auxiliary task within the PPO algorithm, with the aim of enhancing the agent's efficiency in exploring the circuit environment.

RND employs two pivotal neural networks: a fixed and randomly initialized target network, which sets the prediction problem, and a predictor network, which is trained on data collected by the agent. The target network takes an observation to an embedding $f : O \rightarrow \mathbb{R}^k$ and the predictor network $\hat{f} : O \rightarrow \mathbb{R}^k$ is trained by gradient descent to minimize the expected MSE $\|\hat{f}(x;\theta) - f(x)\|^2$ with respect to its parameters $\theta_{\hat{f}}$. This procedure refines a randomly initialized neural network into one that is trained. A higher prediction error is anticipated for new states that differ from those on which the predictor has been trained [21].

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

The benchmark circuits used in the experiments were sourced from the ISCAS'85 and ISCAS'89 benchmarks. We selected 100 hard-to-detect faults from each of the c6288 and s38417 circuits for training, respectively, and all (testable and redundant) faults of the other circuits in ISCAS'85 and ISCAS'89 are used for testing. For the hyperparameter of our proposed SmartATPG, we set α and β of the objective function as 7.5 and 0.07, respectively. The Actor and Critic learning rates are configured as 0.001 and 0.01, respectively. The experiments were conducted on NVIDIA A100 GPU and Intel Xeon Silver 4216 CPU @ 2.10GHz.

4.2 Performance Comparison

To evaluate the proposed SmartATPG, we compare it with heuristic strategies including Distance [4], SCOAP [16], and ANN [11]. To ensure a fair comparison, the ANN is configured with the same settings as described in [11]. Figure 4 presents the comparative performance results in terms of four key metrics in test generation, including the number of backtracks and backtraces, run time, and fault coverage. It can be seen that SmartATPG outperforms other methods in these four aspects.

Under the SmartATPG framework, backtracking is substantially reduced compared to Distance, SCOAP, and ANN-based heuristics, with an average reduction of 2.17x, 2.24x, and 1.54x, respectively. This suggests that the agent effectively learns an optimal backtracing strategy from the circuit environment. The ANN-based heuristic strategy shows a lower average number of backtracks compared to traditional

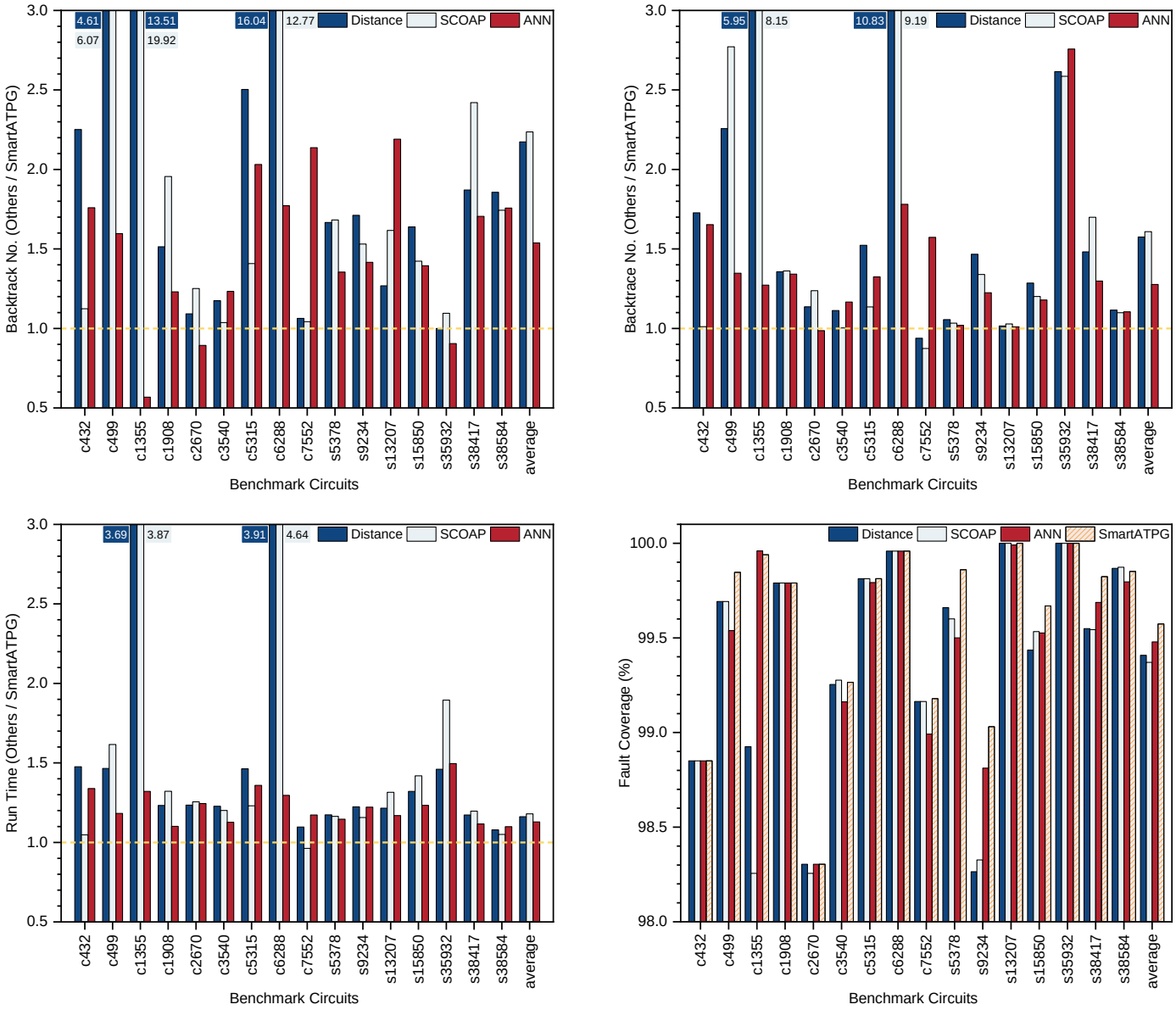

Figure 4: Evaluation on the benchmark circuits of ISCAS'85 and ISCAS'89, with different strategies.

heuristic approaches, indicating that deep learning methods are capable of extracting knowledge from circuit data to optimize the test vector search process.

However, we observed that the ANN-based heuristic strategy underperforms in circuits such as c7552 and s38584, among others, which could be attributed to the more complex environment of these circuits. In comparison, SmartATPG showed better performance on these circuits. This is due to the ability of RL to develop an effective backtracing search strategy through continuous interaction with the circuit environment. Furthermore, compared to other heuristic algorithms, the GCN architecture in SmartATPG enables it to acquire more circuit information, meanwhile enhancing the model's scalability.

4.3 Ablation Study

To investigate the contributions of GCN and RND to the proposed SmartATPG framework, we conducted a series of ablation experiments. The experimental setup is categorized into three distinct configurations: first, employing the RL algorithm in isolation to establish a baseline (RL-only); second, augmenting the RL framework with GCN to assess the impact of graph-based learning (w/o RND); and third, the full SmartATPG model, which encompasses both GCN and RL, further enhanced with RND.

Table 1 shows the performance of the above three configurations on benchmark circuits. Firstly, we assess the impact of GCN on our framework's performance. Secondly, we delve into investigating the influence of RND on the framework's effectiveness. The experimental results comparing w/o RND and RL-only show that GCN is beneficial for reducing the number of backtracks, decreasing runtime, and enhancing fault coverage. This may be attributed to the graph representation capturing the topology of the circuit, with GCN extracting more valuable information through embeddings. SmartATPG outperforms w/o RND in four aspects, which proves the role of RND in enhancing the ability of the agent to find the optimal backtracing strategy.

In summary, the results from Table 1 clearly demonstrate that SmartATPG surpasses frameworks devoid of either GCN or RND.

Table 1: Evaluation of RL, GCN, and RND on performance

Bench	Backtrack (times)			Backtrace (times)			Run Time (ms)			Fault Coverage (%)		
	RL-only	w/o. RND	Smart ATPG	RL-only	w/o. RND	Smart ATPG	RL-only	w/o. RND	Smart ATPG	RL only	w/o. RND	Smart ATPG
c432	5296	3497	3444	5184	3807	3674	65	49	46	98.736	98.851	98.851
c499	1878	974	925	3384	2551	2505	32	26	25	99.692	99.846	99.846
c1355	3994	2350	1678	6300	4797	4098	155	141	123	99.959	99.919	99.939
c1908	2627	1516	1367	4422	3510	3288	148	122	120	99.790	99.790	99.790
c2670	9285	8420	7674	13545	10083	9895	587	479	475	98.304	98.288	98.304
c3540	9676	9634	9480	9572	9547	9592	701	654	683	99.265	99.254	99.265
c5315	2032	2085	1494	4387	4259	3863	940	951	930	99.813	99.813	99.813
c6288	2419	1476	2033	2828	4363	2413	966	1479	897	99.958	99.958	99.958
c7552	14981	14845	12137	21999	21296	21278	2182	2064	2393	99.128	99.123	99.179
s5378	430	331	327	3780	3682	3462	706	674	617	99.780	99.840	99.860
s9234	22921	17436	16753	24107	20030	19600	1706	1574	1568	98.796	99.015	99.031
s13207	421	355	347	5686	5376	5483	6815	6449	5890	100	100	100
s15850	10082	7605	7028	16238	14235	13736	7956	7493	6916	99.489	99.707	99.669
s35932	22	22	21	238	620	239	6069	6824	5679	100	100	100
s38417	21654	18979	17900	41279	38180	36768	94318	93000	91204	99.753	99.796	99.823
s38584	2695	2223	2178	11200	10951	10993	88257	87337	86047	99.822	99.814	99.851
average	6901	5734	**5299**	10884	9830	**9430**	13225	13082	**12726**	99.518	99.563	**99.574**

This highlights the GCN's superior capability in extracting more efficient feature expressions compared to the original state features. Furthermore, RND enhances the RL agent's comprehension of the circuit environment.

5 CONCLUSION

This paper introduces SmartATPG, an end-to-end learning-based ATPG framework incorporating GCN and RL. Unlike existing DL-based ATPG methods that treat backtracing decision-making as a classification or regression problem, we transform the ATPG process into an MDP and employ RL to train a powerful agent to make decisions regarding path selection during backtrace. To capture the topological information of the circuit and improve scalability, we leverage GCN to generate node embeddings, ingeniously combining them with RL to create a scalable end-to-end learning scheme. Additionally, we introduce the RND into the framework to enhance the agent's exploration ability. Experimental results demonstrate that SmartATPG reduces the number of backtracks and backtraces in comparison with previous methods, leading to improved ATPG performance.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (NSFC) under grant No. (92373206, 62090024, 62202453, U20A20202). The corresponding author is Huawei Li.

REFERENCES

[1] M. H. Schulz and E. Auth, "Advanced automatic test pattern generation and redundancy identification techniques," in 1988 International Symposium on Fault-Tolerant Computing. Digest of Papers, 1988, pp. 30-35.
[2] O. H. Ibarra and S. K. Sahni, "Polynomially complete fault detection problems," IEEE Transactions on Computers, vol. 100, no. 3, pp. 242-249, 1975.
[3] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," IBM Journal of Research and Development, vol. 10, no. 4, pp. 278-291, 1966.
[4] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE Transactions on Computers, vol. 30, no. 03, pp. 215-222, 1981.
[5] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," IEEE Transactions on Computers, vol. 32, no. 12, pp. 1137-1144, 1983.
[6] T. Niermann and J. H. Patel, "HITEC: a test generation package for sequential circuits," in 1991 European Design Automation Conference (EURO-DAC), 1991, pp. 214–218.
[7] T. Larrabee, "Test pattern generation using Boolean satisfiability," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 11, no. 1, pp. 4-15, 1992.
[8] J. Huang, H. L. Zhen, N. Wang, et al., "Neural Fault Analysis for SAT-based ATPG," in 2022 International Test Conference (ITC), 2022, pp. 36-45.
[9] A. Zorian, B. Shanyour, and M. Vaseekar, "Machine learning-based DFT recommendation system for ATPG QOR," in 2019 International Test Conference (ITC), 2019, pp. 1-7.
[10] S. Roy, S. K. Millican, and V. D. Agrawal, "Machine intelligence for efficient test pattern generation," in 2020 International Test Conference (ITC), 2020, pp. 1-5.
[11] S. Roy, S. K. Millican, and V. D. Agrawal, "Training neural network for machine intelligence in automatic test pattern generator", in 2021 International Conference on VLSI Design (VLSID), 2021, pp. 316-321.
[12] S. Roy, S. K. Millican, and V. D. Agrawal, "Principal component analysis in machine intelligence-based test generation," in 2021 Microelectronics Design and Test Symposium (MDTS), 2021, pp. 1-6.
[13] R. S. Sutton and A. G. Barto, "Reinforcement learning," A Bradford Book, vol. 15, no. 7, pp. 665-685, 1998.
[14] C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, vol. 8, pp. 279-292, 1992.
[15] J. Schulman, F. Wolski, P. Dhariwal, et al., "Proximal Policy Optimization Algorithms," in arXiv preprint arXiv:1707.06347, 2017.
[16] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia controllability/observability analysis program," in 1980 Design Automation Conference (DAC), 1980, pp. 190-196.
[17] F. Brglez, "On testability analysis of combinational circuits," in 1984 International Symposium on Circuits and Systems (ISCAS), 1984, pp. 221-225.
[18] W. He, X. Li, X. Song, et al., "Chip design with machine learning: A survey from algorithm perspective," Science China Information Sciences, vol. 66, no. 11, pp. 1-31, 2023.
[19] H. Cai, V. W. Zheng, and K. C. C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," IEEE Transactions on Knowledge and Data Engineering, vol. 30, no. 9, pp. 1616-1637, 2018.
[20] Hamilton W, Ying Z, and Leskovec J, "Inductive representation learning on large graphs," in 2017 Neural Information Processing Systems (NIPS), 2017, pp. 1025-1035.
[21] Y. Burda, H. Edwards, A. Storkey, et al., "Exploration by Random Network Distillation," in arXiv preprint arXiv:1810.12894, 2018.