Circuit Learning for Boolean Satisfiability Problem

Zhengyuan Shi
The Chinese University of Hong Kong
Qiang Xu [1]
The Chinese University of Hong Kong

Abstract

Boolean Satisfiability (SAT) is a fundamental problem with wide-ranging applications. Over the past decades, the SAT community has standardized the use of Conjunctive Normal Form (CNF) and built powerful CNF-based solvers. However, key challenges remain: the inherent structural information is lost during the CNF transformation. To address these issues, we propose using circuits as a complementary representation. Circuits preserve rich topological information and can be efficiently translated to and from CNF. By developing preprocessing and inprocessing heuristics directly in the circuit domain, we aim to accelerate SAT solving.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Circuit learning, Boolean Satisfiability, representation learning, heuristics

1 Introduction

The Boolean Satisfiability (SAT) problem serving as a foundational problem holds a significant position in various domains, such as software verification [7], circuit design [5] and AI planning [8]. The prevalent modern solvers opt for the Conjunctive Normal Form (CNF) and apply searching algorithm to explore the satisfiability of the given CNF instance. In the past decades, there are numerous CNF-based heuristics have been proposed to accelerate the searching process, such as clause learning and conflict analysis [1, 14, 12], to guide the search for a satisfying assignment.

Although the CNF-based algorithms show promise in solving SAT problems, there are still several challenges that need to be addressed. One challenge is the diversity of SAT problem instances. CNFs derived from different types of problems (such as graph coloring, random k-SAT, and circuit problems) may have different underlying distributional properties, which can make it difficult to generalize heuristic across a wide range of problem instances. Another challenge is the loss of topological structure information when converting the original problem instance into CNF. This can lead to a loss of important information that is relevant for solving the SAT problem, and may limit the effectiveness of topological-based heuristic.

To overcome the limitations of the CNF format and enable more effective SAT solving, we propose circuit as the complementary representation. Circuit satisfies three key criteria: (1) it can be convertible to and from CNF in polynomial time via Tseytin Transformation [20], ensuring compatibility with existing SAT solvers; (2) it allows efficient processing to generate compact structure and normalized data distributions using Electronic Design Automation (EDA) tools; (3) it retains richer structural information than the bipartite CNF graph, while closely preserving the original problem's topology.

In this paper, we propose a novel paradigm, termed circuit learning for SAT solving. As illustrated in Figure 1, our framework departs from the conventional flow, where the problem is translated into Conjunctive Normal Form (CNF) and them solved directly by a SAT solver, our proposed pipeline leverages the intermediate circuit representation as the basic for learning-based heuristic designs. To be specific, a circuit is extracted from the

[1] This is the summary of Ph.D. topic for CP/SAT Doctoral Program 2025, Prof. Xu is Ph.D. Advisor

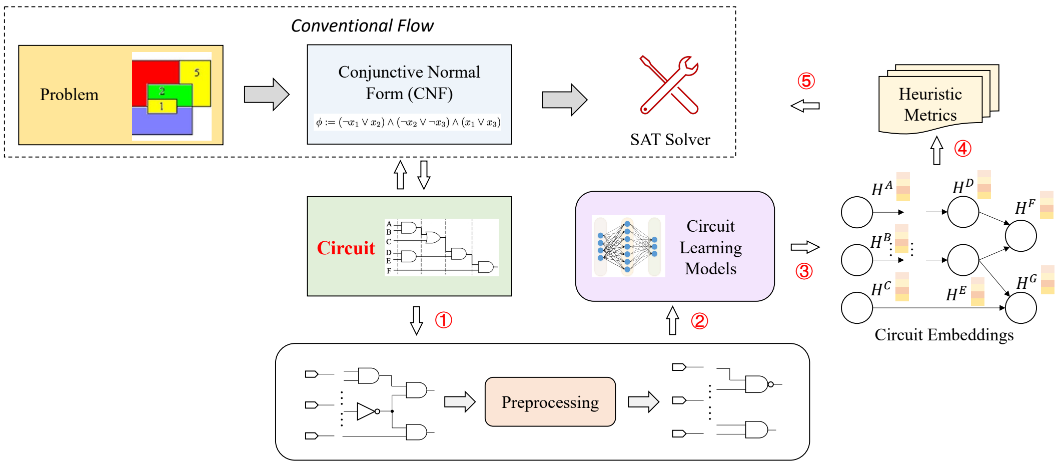


Figure 1 Overview of circuit learning for Boolean satisfiability problem

original problem formulation, which preserves the functional and structural properties of the instance. Then, we propose *circuit-based preprocessing* to simplify the problem instances in Section 3. A graph-based neural model is then applied to learn embeddings that capture rich circuit information and predict heuristic metrics relevant to SAT solving. Finally, we incorporate the circuit embeddings into SAT solvers to achieve the *circuit-based inprocessing heuristics* (see Section 4), such as variable decision prioritizatio or variable phasing. By embedding domain-specific knowledge into the SAT solving pipeline via circuit learning, our approach aims to bridge symbolic reasoning with the circuit-based heuristics, offering a more scalable and adaptable framework for solving Boolean satisfiability problems.

2 Related Works

2.1 Boolean Satisfiability Problems

SAT problem holds a pivotal position in hardware design, software verification, planning and scheduling, which seeks to identify if there exists at least one assignment that makes a given Boolean formula True. The modern SAT solvers, central in addressing SAT problems, predominantly rely on the Conflict-Driven Clause Learning (CDCL) algorithm [12]. Within the backbone CDCL algorithm, CNF has become a standard format in SAT solving, capable of representing arbitrary Boolean formulas. It is structured as a conjunction of clauses, denoted by $\phi = (C_1 \wedge C_2 \wedge \ldots)$, with each clause being a disjunction of variables or their negations, e.g., $C_i = (x_1 \vee \neg x_2 \vee \ldots)$. Over recent decades, numerous CNF-based heuristics have been developed to enhance SAT solving efficiency, including advanced clause management strategies [1] and innovative branching heuristics [14, 17].

Unfortunately, converting a natural problem instance into CNF involves a lot redundant constraints and losses much topological information [15]. Circuit is a instance format closer to the front-end, especially for EDA applications. A few circuit-based SAT solvers, such as QuteSAT [21] and CirSAT [6], have been proposed. However, these circuit-based SAT solvers are still not adopted widely due to the inferior performance. More importantly, although both the circuit-based and CNF-based SAT solvers have their own advantages, we must choose between the two rather than fully leveraging the strengths of each. In this direction, we aim to accelerate the SAT solving process by incorporating circuit as a complementary format based on the CNF solvers.

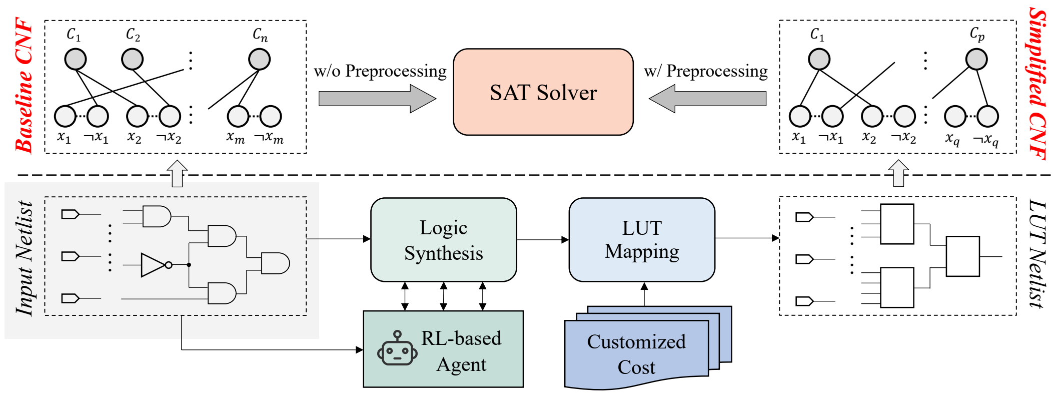


Figure 2 Overview of circuit-based preprocessing flow

2.2 Circuit Representation Learning

A prominent trend in the deep learning community is to learn a general representation from data first and then apply it to various downstream tasks, for example, BERT [3] learn representations of natural language text that can be fine-tuned for a wide range of natural language processing tasks. Circuit representation learning has also emerged as an attractive research direction, which embeds the functional and structural information into vectors. DeepGate family [10, 18, 19] uses logic simulation and circuit structural analysis results as training supervisions to learn the rich circuit embeddings. DeepGate has been fine-tuned for many downstream tasks, including testability analysis [16] and power estimation [9]. In the following work, we will take the DeepGate2 as the backbone model to represent rich circuit information and transfer into SAT solving.

3 Circuit-based Preprocessing Flow

3.1 Overview

Figure 2 shows the overview of the proposed circuit-based preprocessing flow. We start from a circuit-based problem instance, which can be directly collected from the original problems with circuit representations or converted from the conjunctive or disjunctive functions of CNF. The circuit instance is represented in And-Inverter Graphs (AIGs), which only contain three gate types (primary input, AND and NOT) by [2].

First, we utilize logic synthesis (LS) technique to reformulate SAT instance (see Section 3.3). Since the logic synthesis process can be considered as a sequential decision-making task, we train an agent to minimize solving complexity by reinforcement learning (RL), especially employing Deep Q-learning algorithm. Second, we design a cost-customized Look-up Table (LUT) mapping operation with the objective of minimizing the branching times during SAT solving in Section 3.2. Finally, to ensure compatibility with modern SAT solvers, we transform the LUT netlist back into a simplified CNF.

3.2 RL-based Logic Synthesis Recipe Exploration

At each step t (where $t = 0, 1, ..., T-1$), the features of the current netlist $\mathcal{G}^t$ are extracted to form the state s^t. Subsequently, the Q-learning agent utilizes s^t as input and selects an action a^t. Then, the environment including the LS tool performs the synthesis operation based on a^t and transforms the netlist $\mathcal{G}^t$ into a new netlist $\mathcal{G}^{t+1}$. At the end of this step, the environment provides RL agent with a reward r^t. The above process is repeated for subsequent steps until $t = T - 1$. During the training process, the objective of RL agent is to maximize the cumulative sum of the reward values. As a result, the trained RL agent is able to select LS operations step by step to achieve the optimal solving time reduction.

State: We extract the representative features $\mathcal{E}(\mathcal{G}^t)$ of the netlist $\mathcal{G}^t$ into state s^t, including the area, depth, wire counts, proportion of AND gates and proportion of NOT gates of netlist. Meanwhile, the state s^t is incorporated with the primary output embeddings of the initial netlist obtained by pretrained DeepGate2 [18] (denoted as $\mathcal{D}(\mathcal{G}^0)$), which contains rich structural and functional information of the initial problem instance.

$$s^t = \text{Concatenate}(\mathcal{E}(\mathcal{G}^t), \mathcal{D}(\mathcal{G}^0)) \tag{1}$$

Action: The action space $\mathcal{A}$ of agent is discrete and encompasses LS operations. In this paper, we set the available operations as follows: *rewrite*, *refactor*, *balance*, *resub* and *end* (marking the end of the LS process).

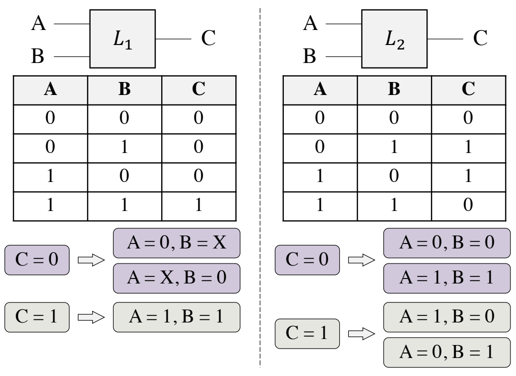
State Transition: The state transition function is implemented by the LS tool. We call the tool with a specific LS operation determined by RL agent to transform the current netlist $\mathcal{G}^t$ to another functional-equivalent but simplified netlist. More formally, we note the given netlist as $\mathcal{G}^t$ and RL action as a^t at step t. The state transition function is denoted as: $\mathcal{G}^{t+1} = \mathcal{F}(\mathcal{G}^t, a^t)$, where $\mathcal{G}^{t+1}$ is the updated netlist by tool. The updated netlist, $\mathcal{G}^{t+1}$, is then considered as the input for the subsequent decision step at time $t + 1$.

Reward: We consider the reduction of variable branching times during SAT solving as the reward. The reward function is shown as Eq. (2), where $\Delta\#Branching$ is the difference in branching times between the final instance and the initial instance during solving. We opt for the terminated reward, which has only one non-zero reward value at the terminated step, i.e. after performing the entire sequence of synthesis operations.

$$r^t = \begin{cases} 0, t = 0, \cdots, T-2, \\ \text{-}\Delta\#Branching,\ \ t = T - 1\ or\ a^t = end \end{cases} \tag{2}$$

3.3 Cost-Customized LUT Mapping

We present a cost-customized LUT mapping approach that not only hides the internal nodes during solving, but also forms an easy-to-solve instance. Unlike conventional mapping tools that focus on constructing circuits with low area or delay, our approach aims to create circuits that are easier to solve. Specifically, we consider two 2-fanin LUTs, denoted as L_1 (representing an AND gate) and L_2 (representing an XOR gate), and list the truth tables for these LUTs in Figure 3. We observe that both L_1 and L_2 have two possible combinations when the fanout pin (C) is logic-0. However, when the fanout pin is logic-1, L_2 has two possible combinations while L_1 has only one unique branch. We define the *branching complexity* C as the total number of possible fanin combinations. In this example, the LUT L_1 has $C_{L_1} = 3$ and LUT L_2 has $C_{L_2} = 4$. Therefore, L_2 is more hard to be solved than L_1. Our approach is motivated by the observations that instances



A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3 LUTs with branching complexity

containing a significant proportion of XOR gates often require more time to be solved, where the branching complexity of XOR gate (L_2 in Figure 3) is higher than other gates.

To prioritize the LUTs during mapping process, we employ a strategy to enumerate all 4-LUTs and integrate their branching complexity into the cost function. We implement the cost-customized mapper by modifying the area cost to reflect the branching complexity of each LUT and fixing the delay cost as a constraint. By executing a sequence of operations aimed at minimizing the total cost, the resulting post-mapping netlists exhibit minimal overall branching complexity. Our preprocessing strategy ends with a LUT to CNF transformation for compatibility.

4 Circuit-based Inprocessing Heuristics

4.1 Overview

We integrate our DeepGate2 [18], a circuit representation learning model, into a modern SAT solver. Firstly, we obtain gate-level embeddings of the original circuit and predict the pairwise functional similarity between these gates. Secondly, we incorporate the learnt knowledge into the SAT solver, where assign the reverse value to the similar variable, promptly causing conflict for joint decision.

4.2 Learning Circuit Functionality by DeepGate2

DeepGate2 [18] is a graph-based neural model designed to learn gate-level representations of digital circuits by explicitly disentangling structural and functional characteristics. As illustrated in Figure 4, the model processes circuits through an And-Inverter Graph (AIG) transformation followed by a Directed Acyclic Graph (DAG)-based Graph Neural Network (GNN) architecture. Given input circuits with potentially different structures, DeepGate2 encodes each circuit into two separate embedding spaces: the structural embedding space and the functional embedding space. For example, Circuit 1 and Circuit 2 exhibit distinct topologies but share identical functionality, while Circuit 3 shows the similar topological structure with Circuit 1. To capture these relationships, the model generates embeddings $\{hs_1, hs_2, hs_3\}$ for structure and $\{hf_1, hf_2, hf_3\}$ for function. These embeddings are learned such that functionally equivalent circuits are mapped closer in the functional space (e.g. $hf_1 \approx hf_2$) and structurally similar circuits are grouped in the structural space (e.g. $hs_1 \approx hs_3$).

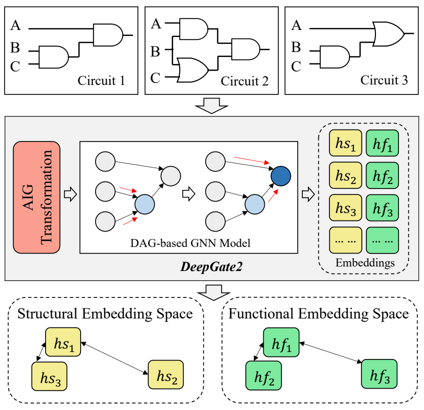


Figure 4 Overview of DeepGate2

4.3 Circuit-based Heuristic for Variable Decision

Our heuristic is motivated by [11], which proposes to utilize the correlation of logic gate functionality to enforce variable decision for solving circuit-based SAT instances. Although the solution achieves remarkable speedup over SAT solvers, it still relies on the time-consuming logic simulation to obtain the functionality. Based on [11], we demonstrate how the DeepGate2 models functional correlation efficiently and accelerates SAT solving.

As shown in Algorithm 1, when the current variable s is assigned a value v, we identify all unassigned variables s' in the set $\mathcal{S}$ that contains correlated variables with s. As modern

Algorithm 1 Variable Decision Function with DeepGate2

```
1 Current variable $s$ just being assigned a value $v$; Set $\mathcal{S}$ containing the correlated
   variables with $s$;
2 Function Sim$(s_i, s_j)$ to calculate the behaviour similarity of two variables;
3 $\delta$ is the threshold for the joint decision
4 Function $V(s)$ to get the assigned value of variable $s$;
5 Function Decision$(s, v)$ to assign decision value $v$ to the current decision variable $s$.

1: Decision($s$, $v$)
2: for $s'$ in $\mathcal{S}$ do
3:   if $s' \neq s$ and $V(s') =$ None then
4:     if Sim$(s', s) > 1 - \delta$ then
5:       Decision($s'$, $\bar{v}$)
6:     end if
7:   end if
8: end for
```

SAT solvers reduce searching space by detecting conflicts as much as possible [12], we assign the reverse value $\bar{v}$ to s' to promptly cause conflict for joint decision. Besides, the threshold δ in Algorithm 1 is set to $1e-5$.

5 Transformation from CNF to Circuit (On-going)

To support a broad class of SAT problems, we propose a novel approach, CNF2LUT: Construct LUT netlist to reformulate CNF, which seeks an inverse transformation from clauses to logic gates. Our approach is inspired by the Tseytin transformation that converts circuits to CNF. To be specific, it can be considered as reversed Tseytin transformation, where we replace the relationships among variables described by certain clauses with LUTs that can model any Boolean logic.

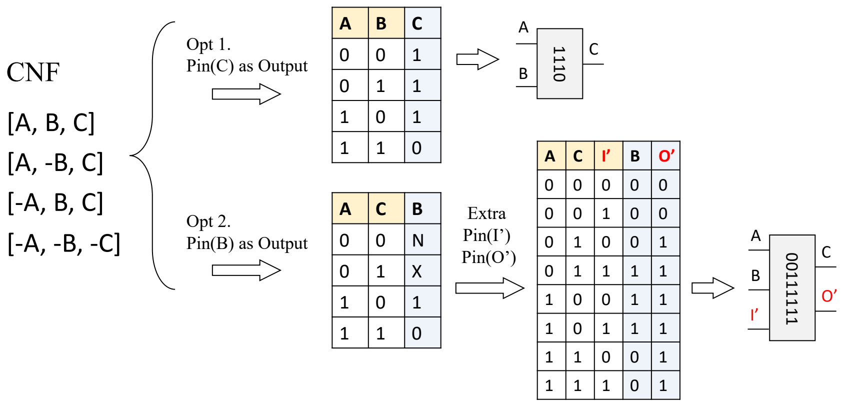
Figure 5 illustrates how CNF clauses can be mapped to LUTs by selecting one variable as the output and others as inputs. In Option 1, choosing C as the output allows direct construction of a complete truth table using inputs A and B, which maps cleanly to a 2-input-1-output LUT. In contrast, Option 2 selects B as the output, resulting in ambiguous or undefined output values (denoted as X and N) for some input combinations. To resolve this, two auxiliary variables I' and O' are introduced, enabling construction of a consistent 3-input-2-output LUT. This conversion is applied iteratively, adding LUTs and their interconnections according to the variable fan-in directions until all clauses are covered.

If a cycle is encountered, i.e., a LUT indirectly drives itself through part of the circuit, we resolve it by inserting an additional primary input (PI) at the fan-out of the cycled LUT to break the feedback loop. This new PI is constrained to exhibit the same behavior as the original LUT, which we enforce by introducing an XOR gate to ensure functional equivalence.

6 Experiments

6.1 Experimental Settings

We construct 300 circuit instances from logic equivalence checking (LEC) and automatic test pattern generation (ATPG) problems. For LEC instances, we modify original industrial

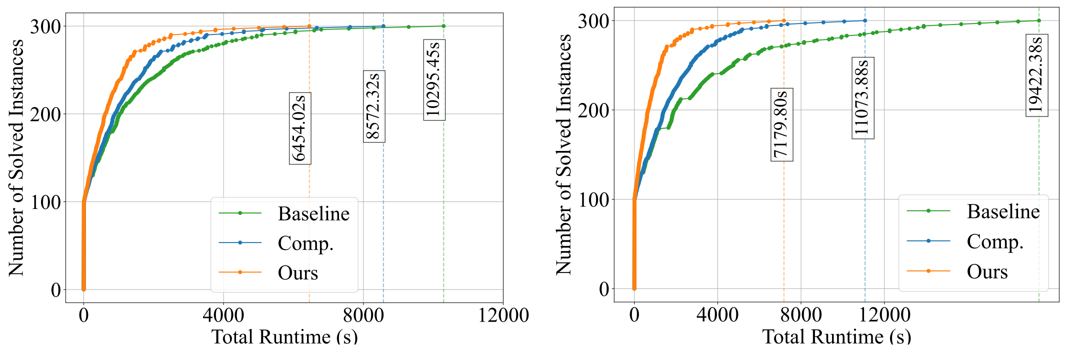


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

A	C	B
0	0	N
0	1	X
1	0	1
1	1	0

A	C	I'	B	O'
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

Figure 5 An example of converting a set of clauses to LUT



(a) Kissat **(c)** Cadical

Figure 6 Runtime comparison with Baseline and Comp.

datapath circuits and connect their primary outputs (POs) through XOR gates to form circuit SAT instances, where satisfiability indicates non-equivalent functionality between circuits. For ATPG instances, we introduce stuck-at faults into industrial circuits and connect the POs of faulty and fault-free circuits through XOR gates, where satisfiable assignments serve as test patterns for fault detection.

We opt for the ABC tool [13] to synthesize the circuit and build a cost-customized mapper based on mockturtle[2] to map the simplified AIG circuit into LUT netlists. The solver runtime limitation is $1,000$s. Any cases that exceed this time limit are marked with TO (Time-out). Each instance solving is conducted on a single core of Intel Xeon CPU E5-2630.

6.2 Effectiveness of Circuit-based Preprocessing

Figure 6 illustrates the relationship between the number of instances solved and overall runtime (including RL model inference time, transformation time and solving time). The Baseline setting represents the conventional solving pipeline, encoding the circuit-based instances directly into CNFs. Our experiment evaluates the effectiveness of our proposed framework using both Kissat solver[3] and CaDiCaL solver[4], with our specific settings distinguished by the orange line (denoted as Ours).

[2] Mockturtle, https://github.com/lsils/mockturtle
[3] Kissat Version 4.0.0, https://github.com/arminbiere/kissat
[4] CaDiCaL Version 2.0.0, https://github.com/arminbiere/cadical

Table 1 Comparing the Runtime between Baseline and Our Solvers

Instance	Size	Baseline(s)	Our(s)			Reduction
			Model	Solver	Overall	
I1	17,495	88.01	1.77	30.25	32.02	63.62%
I2	21,952	29.36	2.85	6.01	8.86	69.82%
I3	23,810	61.24	3.25	32.88	36.13	41.00%
I4	27,606	158.04	4.36	137.77	142.13	10.07%
I5	28,672	89.89	4.78	70.95	75.73	15.75%
Avg.						**40.05%**

To showcase the efficiency of our preprocessing framework, we compare the solving time with another circuit-based approach [4], denoted as Comp. . To the best of our knowledge, although such approach has been around for over a decade, this is the only circuit-based SAT preprocessing technique. According to Figure 6, our proposed circuit-based proprocessing framework can significantly reduce runtime. To be specific, solving these 300 instances necessitates 19,422.38 seconds for the CaDiCaL Baseline pipeline and 11,073.88 seconds for the comparative pipeline (Comp.). In contrast, our method completes the task in a total of 7,179.80 seconds, marking a reduction of 63.03% and 35.16% in solving time compared to Baseline and Comp., respectively.

6.3 Effectiveness of Circuit-based Inprocessing Heuristics

We randomly select 5 instances from the testing dataset for further analysis. The runtime comparison between Baseline and Our are listed in Table 1. To ensure a fair comparison, we aggregate the DeepGate2 model inference time (Model) and SAT solver runtime (Solver) as the Overall runtime. We have the following observations. First, our method achieves a substantial reduction in total runtime for all test cases, with an average runtime reduction of 40.05%. Take Instance I1 as an example, the plain solver requires 88.01s to solve the problem, but by combining with our model, the new solver produces results in only 32.02s, reducing runtime by 63.62%. Second, our model only takes a few seconds to obtain embeddings, occupying less than 10% of overall runtime on average. It should be noted that our DeepGate2 is able to infer within polynomial time that is only proportional to the size of instance. Third, while the two largest instances I4 and I5 show less reduction than the others, it does not necessarily mean that our model is unable to generalize to larger instances. As evidenced by the results for I2, an instance with a similar size to I4 and I5 also demonstrates a significant reduction. The reduction caused by our model should be determined by the characteristics of instance. In summary, our model is effective in speeding up downstream SAT solving.

7 Conclusion

This work presents a novel circuit-based learning paradigm for SAT solving that overcomes limitations of traditional CNF-based frameworks. By leveraging circuits as an intermediate representation, the proposed approach retains information typically lost in CNF, enabling the development of more effective heuristics. Our framework includes a preprocessing pipeline that simplifies circuit instances and an inprocessing strategy that integrates circuit-aware knowledge, learned through a GNN-based model, into modern SAT solvers. This paradigm enhances solver efficiency across diverse problem domains and highlights the promise of combining symbolic reasoning with machine learning for scalable SAT solving.

References

1 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009.

2 Armin Biere. Aiger (aiger is a format, library and set of utilities for and-inverter graphs (aigs)). `https://fmv.jku.at/aiger/`, 2006.

3 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

4 Niklas Eén, Alan Mishchenko, and Niklas Sörensson. Applying logic synthesis for speeding up sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 272–286. Springer, 2007.

5 Evguenii I Goldberg, Mukul R Prasad, and Robert K Brayton. Using sat for combinational equivalence checking. In *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, pages 114–121. IEEE, 2001.

6 Kunmei Hu and Zhufei Chu. Cirsat: An efficient circuit-based sat solver via fanout-driven decision heuristic. In *2023 China Semiconductor Technology International Conference (CSTIC)*, pages 1–3. IEEE, 2023.

7 Daniel Jackson and Mandana Vaziri. Finding bugs with a constraint solver. *ACM SIGSOFT Software Engineering Notes*, 25(5):14–25, 2000.

8 Henry A Kautz, Bart Selman, et al. Planning as satisfiability. In *ECAI*, volume 92, pages 359–363. Citeseer, 1992.

9 Sadaf Khan, Zhengyuan Shi, Min Li, and Qiang Xu. Deepseq: Deep sequential circuit learning. *arXiv preprint arXiv:2302.13608*, 2023.

10 Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. Deepgate: Learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 667–672, 2022.

11 Feng Lu, L-C Wang, Kwang-Ting Cheng, and RC-Y Huang. A circuit sat solver with signal correlation guided learning. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 892–897. IEEE, 2003.

12 Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. In *Handbook of satisfiability*, pages 133–182. ios Press, 2021.

13 Alan Mishchenko et al. Abc: A system for sequential synthesis and verification. *URL http://www. eecs. berkeley. edu/alanmi/abc*, 17, 2007.

14 Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Design Automation Conference*, 2001.

15 Mukul R Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in sat-based formal verification. *International Journal on Software Tools for Technology Transfer*, 2005.

16 Zhengyuan Shi, Min Li, Sadaf Khan, Liuzheng Wang, Naixing Wang, Yu Huang, and Qiang Xu. Deeptpi: Test point insertion with deep reinforcement learning. In *2022 IEEE International Test Conference (ITC)*, pages 194–203. IEEE, 2022.

17 Zhengyuan Shi, Min Li, Sadaf Khan, Hui-Ling Zhen, Mingxuan Yuan, and Qiang Xu. Satformer: Transformers for sat solving. *arXiv preprint arXiv:2209.00953*, 2022.

18 Zhengyuan Shi, Hongyang Pan, Sadaf Khan, Min Li, Yi Liu, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, Zhufei Chu, and Qiang Xu. Deepgate2: Functionality-aware circuit representation learning. *arXiv preprint arXiv:2305.16373*, 2023.

19 Zhengyuan Shi, Ziyang Zheng, Sadaf Khan, Jianyuan Zhong, Min Li, and Qiang Xu. Deepgate3: Towards scalable circuit representation learning. In *Proceedings of the 2024 IEEE/ACM international conference on Computer-aided design*, 2024.

20 Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.

21 Chi-An Wu, Ting-Hao Lin, Chih-Chun Lee, and Chung-Yang Huang. Qutesat: a robust circuit-based sat solver for complex circuit structure. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6. IEEE, 2007.