The Elephant in the Room: Variable Dependency in GNN-based SAT Solving

Zhiyuan Yan*
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, Guangdong, China
zyan760@connect.hkust-gz.edu.cn

Min Li
The Chinese University of Hong Kong
Hong Kong, China
mli@cse.cuhk.edu.hk

Zhengyuan Shi
The Chinese University of Hong Kong
Hong Kong, China
zyshi21@cse.cuhk.edu.hk

Wenjie Zhang
Peking University
Beijing, China
zhang_wen_jie@pku.edu.cn

Yingcong Chen
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, Guangdong, China
yingcong.ian.chen@gmail.com

Hongce Zhang
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, Guangdong, China
hongcezh@ust.hk

Abstract

Boolean satisfiability problem (SAT) is fundamental to many applications. Existing works have used graph neural networks (GNNs) for (approximate) SAT solving. Typical GNN-based end-to-end SAT solvers predict SAT solutions *concurrently*. We show that for a group of symmetric SAT problems, the concurrent prediction is guaranteed to produce a wrong answer because it neglects the dependency among Boolean variables in SAT problems. We propose AsymSAT, a GNN-based architecture which integrates recurrent neural networks to generate dependent predictions for variable assignments. The experiment results show that dependent variable prediction extends the solving capability of the GNN-based method as it improves the number of solved SAT instances on large test sets.

CCS Concepts: • **Hardware** → *Theorem proving and SAT solving*.

Keywords: SAT Solving, Graph Neural Networks, Variable Dependency

1 Introduction

Recently, there has been a growing interest in applying machine learning methods to solve the SAT problem end-to-end. For example, the paper [6] proposed the NeuroSAT architecture to approach SAT solving as a binary classification problem. NeuroSAT treats the input conjunctive norm form (CNF) as a bipartite graph and uses the graph neural network (GNN) to predict whether the set of CNF clauses is satisfiable or not. Amizadeh *et al.* considered the scenario when the problem formulation is in the form of a circuit (namely the Circuit-SAT problem) [1] and applied GNN-based graph learning as well. These methods demonstrated the potential of machine learning in SAT solving.

Although prior works show that it is possible to predict SAT solutions from the graph structure of the problem formulation, we identify an important flaw in these existing GNN-based end-to-end SAT solving methods: there exists a set of satisfiable CNF formulas (correspondingly, a set of satisfiable circuits for the Circuit-SAT problem) whose satisfying assignments cannot be learned by the existing methods in [6] or [1]. These CNF formulas (or circuits) have symmetric formulations but asymmetric solutions. Here, symmetry means swapping a pair of variables results in an equivalent CNF formula or circuit, but in a satisfying assignment, these symmetric variables must take different values. Existing methods predict Boolean assignments for a and b concurrently. The concurrent prediction on each variable solely depends on the graph embedding of the variable node. Meanwhile, a and b share the same embedding because they are symmetric in CNF or circuit form so the predictions for a and b inevitably become the same. To address this problem, we introduce a recurrent neural network (RNN) in the SAT assignment decoding layer. With the help of RNN, later predictions will be able to "remember" prior variable assignments. We call our new model AsymSAT.

Overall, our main contributions in this paper are:

- We identify the need of addressing variable dependency in the existing GNN-based end-to-end SAT solving methods.
- We propose an improvement to the neural network architecture to take dependency among variables into consideration. Our AsymSAT model uses RNN to make sequential predictions of SAT solutions.
- We demonstrate that AsymSAT achieves a higher accuracy in SAT and Circuit-SAT solving compared to prior works.

2 Variable Dependency in SAT Solving

Generally speaking, SAT and Circuit-SAT solving must consider variable dependency. In other words, they must "remember" what predictions have been made so far. A simple example is the 2-input XOR ($x \oplus y$). Here, x and y are symmetric — if we swap them, we will get exactly the same

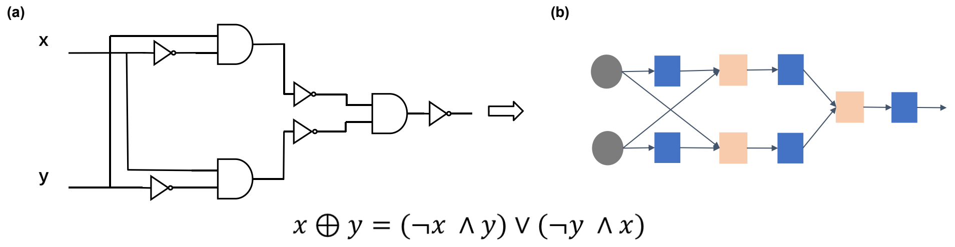

Figure 1. (a) XOR implemented by AND-gates and inverters; (b) the DAG representation of (a).

formula because XOR is commutative. However, we must assign different values to x and y in order to get a 1 as the result. If x has been assigned as 1, then y must be 0. This is the dependency between these two variables.

Symmetry naturally exists in many SAT problems. Sometimes, it is part of the formula that is symmetric. A symmetric formula like $x \oplus y$ will result in a symmetric circuit implemented by AND-gates and inverters (Figure 1). After transforming such formula to CNF using Tseitin encoding, we will also get a symmetric CNF formula.

It is not hard to see, symmetric nodes have symmetric predecessors and successors. Therefore, when GNN-based SAT solvers use message-passing to encode the graph structure, symmetric nodes will have the same node embeddings, unless they are distinguished by initialization. However, pure random initialization for all nodes provides no extra information for the neural network to distinguish the symmetric ones. On the other hand, a bias in initialization would introduce artefact that does not generalize. Therefore, prior works [1, 6, 9] all used equal initial embeddings and therefore, they would not be able to distinguish symmetric nodes when predicting SAT assignments. We accompany our argument on random initialization with experimental results in the appendix. As a consequence, We argue that a GNN-based SAT solver should *sequentially* predict variable assignments in order to take variable dependency into consideration. This is achieved by a recurrent neural network added in our model, explained in the next section.

3 Our Methods

3.1 Problem formulation

Problem input. We expect the problem input to be a DAG representing the structure of the circuit. CNF formulas may be converted into the circuit form using [2]. There are only AND gates and inverters (NOT gates) as they are sufficient to express arbitrary Boolean functions. Formally, we expect the problem input to be the form of $G =< V_G, N_G, E_G >$, where V_G is a set of circuit nodes, N_G is a function that maps each node to the type of logic gate, and E_G is the set of directed edges of the circuit graph.

Problem output. The machine learning model should predict a 0-1 assignment for each circuit input node. We denote the assignments as $L \in \{0,1\}^i$ and i is the number of circuit input nodes. Each instance in the dataset is in the form of (G, L), where the 0-1 assignments are generated by an external SAT solver that works as the oracle. In this paper, we apply the supervised learning method with Cross-Entropy for the loss function.

3.2 The Proposed GNN Architecture

In the high-level, we would like to build a machine-learning model that learns the mapping from a circuit graph to the 0-1 assignment on input nodes: $f : G \rightarrow L$.

Graph embedding layers. To better explain our GNN architecture, we introduce the following notations. Each graph node $v \in V_G$ is associated with a d-dimensional hidden state vector x_v, which is iteratively updated based on the messages from neighboring nodes. During message-passing, we distinguish the nodes that reach v following a directed edge (in other words, the predecessors) from those that leaves v (the successors). We only use the messages from predecessors in the forward pass, and likewise, the successors in the backward pass. The incoming messages are aggregated by an aggregator function $\mathcal{A}$, which is invariant to permutation of the elements in the input set. Finally, the aggregated message is used to update the hidden state of v by a standard GRU function $GRU(\cdot)$ [3].

In AsymSAT, message passing follows the topological order. In the forward pass, messages flow from circuit input nodes (which have no predecessors) to the only circuit output node (which has no successors). The hidden state vectors are updated sequentially. In the backward pass, messages flow from the circuit output node to the circuit input nodes. In each pass, the hidden state vectors are updated according to the following rule:

$$x_v^{(k+1)} := GRU\left(p_v, \mathcal{A}\left(\left\{m_n^{(k)} | n \in \mathcal{N}(v)\right\}\right)\right) \quad (1)$$

Initially, $p_v = N_G(v)$, which is the node type vector of node v. So in the first forward pass, the type of a node is encoded into the hidden state vector. In all remaining passes, $p_v = x_v^{(k)}$, which is the hidden state vector resulted from the previous pass. In Equation 1, $\mathcal{N}(v)$ is either the predecessors or the successors of v. Their hidden state vectors are encoded into messages $m_n^{(k)}$ by a learnable function $\mathcal{M} : x_n^{(k)} \rightarrow m_n^{(k)}$.

Table 1. Solution rate for the $SR(n)$ problems

	$SR(3)$	$SR(4)$	$SR(5)$	$SR(6)$	$SR(7)$	$SR(8)$	$SR(9)$	$SR(10)$
AsymSAT	98.30%	100.00%	93.23%	94.51%	81.56%	82.90%	88.95%	85.45%
NeuroSAT	87.70%	74.47%	63.10%	59.57%	52.94%	48.40%	49.73%	43.82%
DG-DAGRNN	10.21%	15.23%	5.21%	1.83%	8.38%	5.70%	4.07%	4.24%

SAT assignment decoding layers. In our AsymSAT, we use a recurrent neural network (referred to as the $\mathcal{R}$ layer) to generate sequential predictions on variable assignments, so that the model output on a certain circuit input node depends on the predictions of other nodes. We make this $\mathcal{R}$ layer bi-directional to account for dependencies from both sides. A subsequent MLP will work as a selector to decide which direction is more preferred. Sequential prediction mimics classic (non-machine-learning-based) SAT solvers. These classic SAT solvers like GRASP [5] or MiniSAT [8] pick decision variables one after another. Regarding the aforementioned XOR example, we expect this RNN layer will be able to learn to predict different variable assignments for the two symmetric variables after training with such examples.

4 Experimental Evaluation

4.1 Data preparation

Overall, we prepare the following datasets:

Small-scale symmetric AIG with asymmetric solutions. We manually construct 10 circuits with no more than 3 inputs. Within each circuit, there are at least two input nodes that are symmetric but require distinct assignments. We intentionally keep this training set small. If NeuroSAT and DG-DAGRNN are capable of handling symmetric circuits with asymmetric SAT solutions, they should easily reach a high training accuracy on this small dataset.

Medium-size randomly generated CNF formulas. We generate random CNF formulas in the same way as described by [6]. We refer to this dataset as the $SR(n)$ problem, where n is the number of variables. CNF formula for $SR(n)$ problems can be converted into the circuit form using the principle of Shannon's Decomposition as suggested by [1].

4.2 Experimental setup and result

The dimension on the outcome of the $\mathcal{R}$ layer is 10 and we use the Adam optimizer during training process. For NeuroSAT, and DG-DAGRNN, we follow the same configurations as described in [6] and [1]. To our best knowledge, the source code for the original DG-DAGRNN model is not publicly available. We build this model following the instructions in [1]. We train and test all three models on a server with a NVIDIA GeForce RTX 3090 GPU.

Effectiveness of the RNN decoding layer. In this experiment, we train our AsymSAT model, the NeuroSAT model and the DG-DAGRNN model on the same 10 symmetric circuits and measure the training accuracy. We use LSTM as the bi-directional RNN layer in AsymSAT. We also add one case of removing the $\mathcal{R}$ layer in AsymSAT as comparison. Table 2 illustrates the result for the symmetric circuits on five different models. Just as we discussed in Section 2, DG-DAGRNN and NeuroSAT cannot break the tie in symmetric circuits or symmetric CNF formulas. And there is no way to train these two models on this dataset. Thanks to the $\mathcal{R}$ layer we introduced, our AsymSAT model can reach a solution rate of 100.00% with either LSTM or GRU in the $\mathcal{R}$ layer.

Table 2. Percentage of the symmetric circuit problem solved

Asym w. L	**Asym w.o. L**	**NeuroSAT**	**DG-DAGRNN**
100.00%	0.00%	0.00%	0.00%

Comparison on the $SR(n)$ problems. We use $8K$ $SR(n)$ problems sampled uniformly from $SR(U(3,8))$ to train the three models. The test set contains $1.5K$ $SR(n)$ problems (n is from 3 to 10). For AsymSAT and DG-DAGRNN, CNF formulas are first converted into circuits to serve as the model input. Table 1 summarizes the performance measured on the $SR(n)$ problem. The result shows that AsymSAT model has a better performance compared to NeuroSAT and DG-DAGRNN on this dataset. Overall, AsymSAT can reach more than 90% solution rate (averaged across $SR(3)$ to $SR(10)$), while NeuroSAT can only reach 60%. In our experiment, the performance of DG-DAGRNN is non-competitive to the other two and we provide a detailed analysis of DG-DAGRNN in Appendix A.

We also conduct the experiment on our own dataset — large random circuits with more than 1K logic gates, which is presented in Appendix B.

5 Conclusion

This paper addresses the need of considering variable dependency when designing a machine-learning model for SAT solving. This paper proposes using RNNs to make sequential predictions for SAT solving. Our experiments show that this improvement extends the solving capability on symmetric Circuit-SAT problems and achieves a higher solution rate on randomly generated SAT and Circuit-SAT instances compared to concurrent GNN-based SAT solving methods.

References

[1] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. 2018. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *International Conference on Learning Representations*.
[2] Armin Biere. 2007. CNF2AIG. https://fmv.jku.at/cnf2aig/.
[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
[4] Swen Jacobs and Mouhammad Sakr. 2021. AIGEN: Random Generation of Symbolic Transition Systems. In *International Conference on Computer Aided Verification*. Springer, 435–446.
[5] Joao P. Marques-Silva and Karem A. Sakallah. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.* 48, 5 (1999), 506–521.
[6] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2018. Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685* (2018).
[7] Claude E. Shannon. 1949. The Synthesis of Two-Terminal Switching Circuits. *The Bell System Technical Journal* 28, 1 (1949), 59–98.
[8] Niklas Sorensson and Niklas Een. 2005. Minisat v1. 13-a SAT solver with conflict-clause minimization. *SAT* 2005, 53 (2005), 1–2.
[9] Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. 2020. NLocalSAT: Boosting local search with solution prediction. *arXiv preprint arXiv:2001.09398* (2020).

A Appendix: Unsupervised Learning in DG-DAGRNN

In our experiment, DG-DAGRNN performs the worst on the circuits converted from CNF. We argue that (a) an unsupervised learning approach in the prior work [1] suffers from the absence of a tie breaker among symmetric nodes, and (b) the `smooth-min` and `smooth-max` functions used to replace AND gates and OR gates bring in the vanishing gradient problem for some circuits. We justify these two arguments below.

For the first argument, it is not hard to see, if we convert a symmetric circuit into its differentiable counterpart by replacing AND-gates and NOT-gates with `smooth-min` and $1 - z$ functions, we will end up with a symmetric function. Therefore, the gradient on the two symmetric input variables will be the same. A gradient descent procedure will keep the same variable assignment if the two input nodes initially have the same assignment.

For the second argument, we know that the `smooth-min` and `smooth-max` functions are defined as Equation 2. They are referred to as the S_{min} and S_{max} functions in the following text. They have a tunable parameter τ (the temperature). As can be seen from Equation 2, when $\tau = +\infty$, both S_{min} and S_{max} functions become the arithmetic mean function. As $\tau \longrightarrow 0$, $S_{max}(\cdot) \longrightarrow \max(\cdot)$, and $S_{min}(\cdot) \longrightarrow \min(\cdot)$.

$$\begin{aligned} S_{min}\left(x_1, \ldots, x_n\right) &= \frac{\Sigma_{i=1}^{n} x_i e^{-x_i/\tau}}{\Sigma_{i=1}^{n} e^{-x_i/\tau}}, \\ S_{max}\left(x_1, \ldots, x_n\right) &= \frac{\Sigma_{i=1}^{n} x_i e^{x_i/\tau}}{\Sigma_{i=1}^{n} e^{x_i/\tau}} \end{aligned} \tag{2}$$

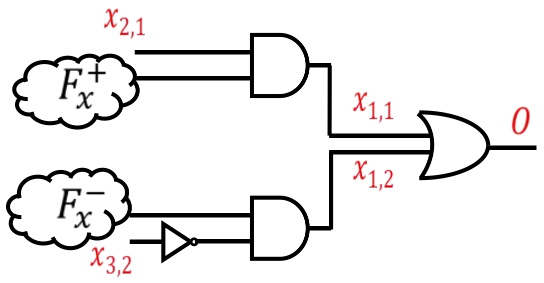

Figure 2. Circuit structure when converting CNFs into circuit forms

In the training phase, τ gradually decreases so that S_{max} and S_{min} allow the gradients to flow back through all paths in the beginning, while approximating the max and min functions in the end. The problem lies in the fact that S_{max} and S_{min} functions are very similar to the arithmetic mean functions when τ is large in the beginning. The gradients from different back propagation paths can easily cancel each other. Let's consider a circuit graph generated by converting problems in CNF format to circuit forms. The conversion method was suggested in [1], which is based on Shannon's Decomposition [7]. The structure of the circuit is shown in Figure 2, where for every input variable (e.g., v in the figure), there are an even number of paths to the output node and one half of the paths will go through an additional NOT-gate. If we consider the gradient on the loss function with respect to a circuit input variable v, we have:

$$\begin{aligned} \frac{\partial loss}{\partial v} = \frac{\partial loss}{\partial o} \cdot \frac{\partial S_{max,1}}{\partial x_{1,1}} \cdot \frac{\partial S_{min,2}}{\partial x_{2,1}} + \\ \frac{\partial loss}{\partial o} \cdot \frac{\partial S_{max,1}}{\partial x_{1,2}} \cdot \frac{\partial S_{min,3}}{\partial x_{3,2}} \cdot (-1) \end{aligned} \tag{3}$$

When $\tau \longrightarrow +\infty$, all of $\frac{\partial S_{max,1}}{\partial x_{1,1}}$, $\frac{\partial S_{max,1}}{\partial x_{1,2}}$, $\frac{\partial S_{min,2}}{\partial x_{2,1}}$, and $\frac{\partial S_{min,3}}{\partial x_{3,2}}$ are very close to $\frac{1}{2}$, because the S_{max} and S_{min} functions are similar to the arithmetic mean function. Note that there is a sign on the second term introduced by the NOT-gate. So the two terms can easily cancel each other, resulting in a near-0 gradient on variable v. In a circuit generated according to Shannon's Decomposition, almost all circuit inputs will go through such even number of paths to reach the output and therefore, there is almost no gradient in the back propagation when τ is large, and it is harder to train the network in the early training iterations. If this is used with learning rate decay, the later training iterations may not be able to fully optimize the circuit input assignments as the learning rate decreases. This possibly explains the experiment result of DG-DAGRNN in Table 1.

Table 3. Solution rate for the $V(n)$ problems

	$V(3)$	$V(4)$	$V(5)$	$V(6)$	$V(7)$	$V(8)$	$V(9)$	$V(10)$
AsymSAT	81.58%	67.50%	72.50%	55.50%	52.50%	60.00%	45.00%	47.50%
NeuroSAT	0.025%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
DG-DAGRNN	35.00%	47.50%	47.50%	45.00%	30.00%	37.50%	37.50%	32.50%

Table 4. Solution rate for the larger $V(n)$ problems

	$V(11)$	$V(12)$	$V(13)$	$V(14)$	$V(15)$
AsymSAT trained on SR(3..10)	45.00%	60.00%	45.00%	45.00%	52.50%
AsymSAT trained on SR(3..10) + V(3..8)	47.50%	47.50%	45.00%	60.00%	57.50%

Table 5. The performance of random initialization vs. having an additional $\mathcal{R}$ layer

	Symmetric Circuit	$SR(10)$
AsymSAT w. LSTM	100.00%	72.75%
AsymSAT w.o. $\mathcal{R}$ (Random Initialization)	00.00%	62.12%
AsymSAT w.o. $\mathcal{R}$	00.00%	60.50%

B Appendix: Experiments on large randomly generated AIGs.

Data preparation. We generate random AIGs using the AIGEN tool [4], which was designed to create random test circuits to check and profile the EDA tools. By default, AIGEN generates sequential logic circuits (those with storage elements). We extract the combinational logic circuits from the sequential logic circuits. We refer to this dataset as the $V(n)$ problem, where n stands for the number of circuit input nodes. $V(n)$ problems can be converted into CNF using Tseitin transformation. Compared to $SR(n)$ problems, $V(n)$ is a nontrivial dataset even when n is relatively small. For example, each $V(10)$ problem has more than 1K logic gates on average. The corresponding CNF formulas contain more than 1K variables, which is much larger than the largest dataset $SR(40)$ used in the prior work [6].

Comparison on the $V(n)$ problems. The training data is a mixture of $8K$ $SR(n)$ problems, (n ranges from 3 to 10), and $1.2K$ $V(n)$ problems (n ranges from 3 to 8). The test set is 320 $V(n)$ problems (n ranges from 3 to 10). Note that $V(n)$ is a nontrivial dataset. On average, each $V(10)$ problem has around 1K AND gates, more than those in circuits converted from the $SR(10)$ problems (which each contain just about 200 AND gates). Even for the $SR(40)$ problems, there are only approximately 600 - 800 AND gates per input. Therefore, $V(n)$ problems can also demonstrate the generalization capability of the tested models. Although the indexing number n is relatively smaller in the $V(n)$ dataset, there are plenty of logic gates in each circuit. These logic gates will add up to the number of variables and clauses after Tseitin transformation. Therefore, the converted CNF inputs are challenging for NeuroSAT. This explains the poor performance of NeuroSAT in Table 3. We also show the generalizability of AsymSAT on the $V(n)$ dataset. On larger $V(n)$ problems, for example, $V(15)$, which is about 128x the size of $V(8)$, AsymSAT still maintains a solution rate around 50%. It is not significantly affected by reducing the training set to only the $SR(n)$ problems as shown by Table 4.

C Appendix: Random Initial Node Embeddings in GNN-based SAT Solving

Random initialization seems to be a workaround that can break the tie between two symmetric nodes. However, our experiment shows it is not as effective as directly addressing variable dependency through RNN. We use random initialization in AsymSAT (without $\mathcal{R}$ layer) to illustrate this. Table 5 summarizes the performance on symmetric circuits and the $SR(10)$ problem. Accuracy on symmetry circuits can hardly get to 100% and it is not as good as AsymSAT with LSTM on the medium-size $SR(10)$ problem.