

Addressing Variable Dependency in GNN-based SAT Solving

Zhiyuan Yan¹, Min Li², Zhengyuan Shi², Wenjie Zhang³, Yingcong Chen¹ and Hongce Zhang¹

¹Hong Kong University of Science and Technology(Guangzhou)

²The Chinese University of Hong Kong

³Peking University

zyan760@connect.hkust-gz.edu.cn¹, {mli,zyshi21}@cse.cuhk.edu.hk², zhang_wen_jie@pku.edu.cn³, yingcong.ian.chen@gmail.com¹, hongcezh@ust.hk¹

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.

1 Introduction

Boolean satisfiability problem (or the SAT problem) is to decide if there exists a set of 0-1 (false/true) assignments to the Boolean variables in a given Boolean formula such that the formula evaluates to 1 (true). Solving the SAT problem is at the core of many applications, for example, the electronic design automation (EDA), where the growing complexity of hardware demands for more intelligent design and verification tools [Brand, 1993]. One of the classic applications of SAT solving in EDA is combinational logic equivalence checking, where two circuits for comparison are composed into one miter circuit [Gupta *et al.*, 2006]. This miter circuit can be further converted into a conjunctive norm form (CNF) formula. If the formula is satisfiable, then the two circuits are not equivalent. In this paper, we refer to solving the Boolean formula in CNF as the SAT problem. If the problem input is directly in the circuit form, then we call it the Circuit-SAT problem.

Traditionally, SAT and Circuit-SAT problems are solved by heuristic searching algorithms, for example, the Conflict-Driven Clause Learning (CDCL) algorithm [Marques-Silva and Sakallah, 1999], implemented in solvers like MiniSAT [Sorensson and Een, 2005] and Glucose [Audemard

and Simon, 2014]. While SAT problems are NP-complete, heuristics play an important role in speeding up the algorithms. Many effective heuristics have been proposed so far. For example, the Variable State-Independent Decaying Sum (VSIDS) method was brought up to aid variable selection in the search process [Moskewicz *et al.*, 2001]. Those heuristic algorithms have led to a huge improvement in computational efficiency. However, they are also limited by the greedy nature of their strategy and therefore, could be sub-optimal in certain cases [Shi *et al.*, 2021].

Recently, there has been a growing interest in applying machine learning methods to solve combinatorial problems, including the SAT problem [Duan *et al.*, 2022]. With the help of deep neural networks (DNN), end-to-end models can learn the expected outcome from the problem description. For example, [Selsam *et al.*, 2018] proposed the NeuroSAT architecture to approach SAT solving as a binary classification problem. NeuroSAT treats the input CNF as a bipartite graph and uses the graph neural network (GNN) to predict whether the set of CNF clauses is satisfiable or not. Compared to the existing heuristic searching method, NeuroSAT requires no pre-defined heuristics. [Amizadeh *et al.*, 2018] considered the scenario when the problem formulation is in the form of a circuit (namely the Circuit-SAT problem). As the circuit is by nature a graph, the GNN-based architecture also applies here. Unlike NeuroSAT which requires extra steps to decode a satisfying assignment from a SAT prediction, the neural Circuit-SAT method in [Amizadeh *et al.*, 2018] directly predicts the satisfying assignment. These existing GNN-based end-to-end methods have demonstrated the potential of using machine-learning techniques in SAT solving.

Although prior works showed 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 [Selsam *et al.*, 2018] or [Amizadeh *et al.*, 2018]. These CNF formulas (or circuits) have symmetric formulations, but their solutions are asymmetric. Here, symmetry means swapping a pair of variables resulting in exactly the same CNF formula (or circuit). However, a satisfying assignment requires that the two symmetric variables must take different values. A

simple example is the formula (or the circuit) that encodes “ $a \oplus b$ ”, where a and b are two Boolean variables and “ \oplus ” is the XOR operation. Due to the commutativity of XOR, “ $b \oplus a$ ” is equivalent to “ $a \oplus b$ ”. However, a satisfying assignment requires a and b to take different values because only “ $0 \oplus 1$ ” or “ $1 \oplus 0$ ” results in 1. 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. Therefore, the predictions for a and b inevitably become the same, which is clearly not a satisfying variable assignment for the formula.

We attribute this problem to the concurrent prediction in existing model that does not factor in variable dependency in SAT problems. Variable dependency refers to the fact that the prediction of one variable assignment will affect other variables. In the “ $a \oplus b$ ” example, if a is assigned to be 0, then b must be 1, despite that a and b are symmetric. 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, because it can produce asymmetric SAT solutions given symmetric SAT problems. We show by experiments that this small change can significantly improve GNN-based SAT solving.

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 with this small change, AsymSAT achieves a higher accuracy in SAT and Circuit-SAT solving compared to prior works. In addition to the end-to-end machine-learning-based SAT solving, the idea of sequential predictions proposed in this paper could potentially also benefit hybrid SAT solvers that take machine-learning as one searching heuristic.

The paper is organized as follows: the next section provides a background on SAT problems and the existing GNN-based end-to-end SAT solving methods. Section 3 highlights the variable dependency in SAT solving. Section 4 introduces our improvement to the GNN architecture for SAT solving, followed by experiments in Section 5 and the related works in Section 6. Finally, Section 7 concludes the paper.

2 Background

2.1 Boolean Satisfiability Problem

The Boolean satisfiability problem talks about propositional logic with Boolean variables and Boolean operators like “and”(\wedge), “or”(\vee), “not” (\neg). The problem is to decide whether there exist assignments to the Boolean variables so that a given propositional logic formula evaluates to true under such assignment. The expected answer is either satisfi-

able (SAT) or unsatisfiable (UNSAT). In case the formula is satisfiable, we also expect to know the satisfying variable assignment.

In Boolean satisfiability problem, the input is usually a Boolean formula in the conjunctive norm form (CNF). In CNF, the variables and their negations are called the literals, which are first connected with disjunctions (\vee) to form the clauses. The clauses are then connected with conjunctions (\wedge). An arbitrary Boolean formula can always be converted into CNF using Boolean algebra, though the number of clauses may exponentially blow up. Alternatively, Tseitin transformation [Tseitin, 1983] converts a formula into an equi-satisfiable CNF, whose size is linear to the number of operators in the original one. Therefore, it is the preferred solution of CNF conversion in modern SAT solving.

Besides CNF, the input of SAT could also be a circuit, which is essentially a directed acyclic graph (DAG) with each node representing either a circuit input, a logic gate or the circuit output. The goal is to find an input combination that causes the circuit output to become 1. The circuit form is more natural in some applications (for example, EDA). Note that the circuit form can be converted into a CNF formula by introducing new variables, and the number of clauses and variables in the converted CNF is proportional to the number of graph nodes in the circuit DAG [Prestwich, 2009]. On the other hand, a CNF formula can also be converted back into a circuit form, as presented in [Amizadeh *et al.*, 2018, Appendix C].

2.2 Solving SAT Problems by Graph Neural Networks

Selsam *et al.* proposed using a graph neural network to decide whether a Boolean formula in CNF is satisfiable or not [Selsam *et al.*, 2018]. This is an end-to-end method, where the problem input is a bipartite graph with nodes representing either a literal or a clause in CNF. The only output of the neural network is a single bit representing the satisfiability of the formula. An edge between a literal node and a clause node in the bipartite graph indicates that the literal is contained in the clause. Each node has an initial embedding, which is updated by iterations of message-passing between literals and clauses. Finally, the literal embeddings are sent to a multi-layer perceptron (MLP) to generate a vote. The averaged vote from all literals is used to decide satisfiability. The paper also demonstrated it is possible to decode satisfying assignments from the literal embeddings via k-means clustering.

2.3 Solving Circuit-SAT Problems by Graph Neural Networks

When the problem is formulated in the circuit form, we automatically have a graph structure. Graph neural network can be similarly applied in Circuit-SAT problems. Note that any logic function can be implemented with only AND gates and NOT gates. A circuit graph made up of only these two types of gates forms an And-Inverter-Graph (AIG), which is commonly used in the EDA community. Amizadeh *et al.* developed a GNN-based model, named DG-DAGRNN to solve Circuit-SAT problems by machine learning [Amizadeh *et al.*, 2018]. The types of graph nodes (whether a node is a logic

gate or a circuit input) are encoded as one-hot vectors, which are the inputs to the GNN model. Similar to NeuroSAT, message-passing generates node embeddings which are used to predict variable assignments. A message-passing iteration consists of one forward pass from circuit input nodes to the only circuit output node and one backward pass in the reversed order. Compared to NeuroSAT, message-passing in DG-DAGRNN follows the topological order and sequentially update node embeddings, whereas NeuroSAT updates all node embeddings concurrently.

Both NeuroSAT and DG-DAGRNN predict SAT solutions concurrently without considering dependency among variables. This results in a fundamental weakness: they are unable to predict the correct solutions for certain symmetric graphs, as explained in the next section.

3 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 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 — for example, $(x \oplus y) \wedge z$. When converted to AIG or CNF, a symmetric formula like $x \oplus y$ will result in a symmetric AIG or CNF, as shown by Figure 1. 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 [Selsam *et al.*, 2018; Amizadeh *et al.*, 2018; Zhang *et al.*, 2020] 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.

When individual node embeddings are directly used to predict variable assignments without considering the dependency among them, the inferred assignments will always be the same for the pair of symmetric nodes. As we have shown by the 2-input XOR example, some symmetric formulas reject equal variable assignments as their satisfying solutions. Therefore, NeuroSAT and DG-DAGRNN in [Selsam *et al.*, 2018] and [Amizadeh *et al.*, 2018] are unable to deal with these symmetric SAT or Circuit-SAT problems. 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.

4 Our Methods

In this section, we explain our approach where RNN is used for dependent predictions. Specifically, we focus on the Circuit-SAT problem because a CNF formula for SAT problems can be converted into a circuit structure. We formulate solving Circuit-SAT problems as a supervised learning process as the following.

4.1 Problem formulation

Problem input. We expect the problem input to be a DAG representing the structure of the circuit. As discussed in Section 2, we only need to consider circuits made from AND gates and NOT gates. Each node in the DAG has an one-hot input feature vector that indicates the type of the node. There are in total three types: the primary inputs, AND gates and NOT gates. Formally, we expect the problem input to be the form of $G = \langle V_G, N_G, E_G \rangle$, where V_G is a set of circuit nodes, N_G is a function that maps each node to its type, and E_G is the set of directed edges of the circuit graph. An edge between two nodes means that there is a wire connection from a logic gate or a circuit input to another gate.

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.

4.2 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: $f : G \rightarrow L$. There are plenty of existing GNN models that are designed to handle input data organized as a graph [Yolcu and Póczos, 2019; Selsam *et al.*, 2018; Selsam and Bjørner, 2019]. There, each graph node is associated with a vector (the hidden state vector) which eventually represents some structural information around the node. Nodes exchange their knowledge of the graph structure by sending messages to their neighbors, and the hidden states will be gradually updated. The propagation of information is referred to as the message-passing mechanism, which essentially embeds the information about the graph structure into the hidden states.

Graph embedding layers. When it comes to the implementation of message passing, there are various choices, for example, which direction the message flows towards, how to aggregate messages from several nodes, what is the order of hidden state updates. Therefore, different variants of message-passing can be implemented. In this work, we build upon the DAG-RNN framework [Shuai *et al.*, 2016] to create a GNN architecture for sequential variable assignment prediction.

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

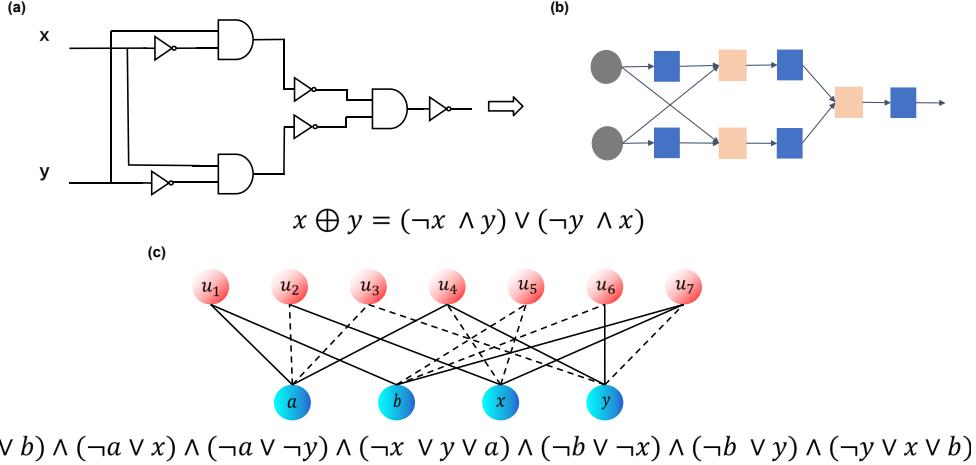


Figure 1: (a) XOR implemented by AIG; (b) the DAG representation of (a); (c) the equi-satisfiable CNF with additional variable a and b , and the corresponding bipartite graph of XOR (here dotted line means the variable is negated in the clause).

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 $GRU(\cdot)$ [Cho *et al.*, 2014].

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:

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

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. We use three separate GRUs: $GRU_{init}(\cdot)$, $GRU_f(\cdot)$, $GRU_b(\cdot)$. Among the three, $GRU_{init}(\cdot)$ is only used in the first forward pass. $GRU_f(\cdot)$ is used for all remaining forward passes. $GRU_b(\cdot)$ is used in the backward passes. We call one forward pass followed by one backward pass as an iteration. 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 $\mathcal{M} : x_n^{(k)} \rightarrow m_n^{(k)}$.

Our graph embedding layers share some similarities with [Amizadeh *et al.*, 2018] as both are built upon DAG-RNN. The difference here is mainly in the computation between two iterations. We use two GRUs for forward passes, because the size of a hidden state vector is different from that of the node type vector, whereas [Amizadeh *et al.*, 2018] introduced a function to project hidden state vectors into the space

of node type vectors after each iteration to keep the same dimensionality and use the same GRU. We argue that the projection could potentially introduce a loss of information and therefore, we employ two separate GRUs in the forward pass: $GRU_{init}(\cdot)$, $GRU_f(\cdot)$ to handle either the node type vector or a hidden state vector from the previous pass.

SAT assignment decoding layers. As discussed earlier, we would like to predict the Boolean assignments on the circuit input nodes *sequentially*. In this node-level prediction, we map a sequence of hidden state vectors of the circuit input nodes $X = (x_{i_1}, x_{i_2}, x_{i_3}, \dots)$ to a sequence of input assignment L . After iterations of message passing, these hidden state vectors encode the information related to the structure of the graph. If two input nodes are symmetric with respect to each other, we expect that their hidden state vectors will be the same. If we individually use each of these vectors to decode a 0-1 assignment (namely, the concurrent prediction), the symmetric nodes will certainly map to the same variable assignment. As we have discussed in Section 3, SAT solutions must take variable dependency into consideration, therefore, in our model, we need to associate variable assignments of the same SAT problem.

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 [Marques-Silva and Sakallah, 1999] or MiniSAT [Sorensson and Een, 2005] 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.

As a summary, we show the overall architecture of our

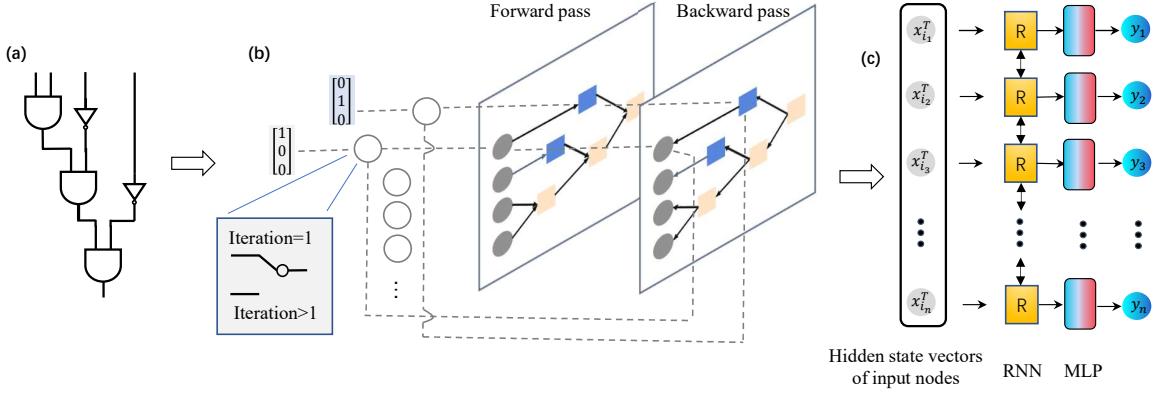


Figure 2: (a) An example of an AIG circuit, (b) the graph embedding layers (for simplicity, we only draw the connections for two nodes.) (c) the SAT assignment decoding layers

Table 1: Percentage of the symmetric circuit problem solved

AsymSAT w. LSTM	AsymSAT w. GRU	AsymSAT w.o. \mathcal{R} layer	NeuroSAT	DG-DAGRNN
100.00%	100.00%	0.00%	0.00%	0.00%

AsymSAT model in Figure 2.

4.3 Training

For AsymSAT, we apply the supervised learning method. We consider SAT solution prediction as a labeling problem — giving 0-1 labels to each of the circuit input nodes. We use Cross-Entropy for the loss function, denoted as:

$$\text{Loss} = - \sum_{i=1}^n [g(v_i) \log(P(v_i = 1)) + (1 - g(v_i)) \log(P(v_i = 0))] \quad (2)$$

where $g(v_i)$ is the ground truth of the SAT assignment on v_i generated by an oracle SAT solver, $P(v_i = 1)$ and $P(v_i = 0)$ are the predicted probability of v_i to be 1 or 0.

5 Experimental Evaluation

5.1 Data preparation

We prepare three datasets in total: the small-scale symmetric circuit examples, medium-size CNF formulas, and large random circuits with more than 1K logic gates.

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. However, our experiment result later will show that they are unable to predict any SAT solutions for this dataset.

Medium-size randomly generated CNF formulas. We generate random CNF formulas in the same way as described by [Selsam *et al.*, 2018]. 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 [Amizadeh *et al.*, 2018].

Large randomly generated AIGs. We generate random AIGs using the AIGEN tool [Jacobs and Sakr, 2021], 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 [Selsam *et al.*, 2018].

5.2 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 [Selsam *et al.*, 2018] and [Amizadeh *et al.*, 2018]. 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 [Amizadeh *et al.*, 2018]. We train and test all three models on a server with two NVIDIA GeForce RTX 3090 GPUs.

Experiments for AsymSAT Configurations

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 two con-

Table 2: 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%

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%

figurations for our AsymSAT model: one uses LSTM and the other uses GRU in the bi-directional RNN layer (the \mathcal{R} layer). We also add one case of removing the \mathcal{R} layer in AsymSAT as comparison. In this experiment, we set the learning rate of AsymSAT models as 10^{-3} and the number of iterations as 5. Table 1 illustrates the result for the symmetric circuits on five different models. Just as we discussed in Section 3, 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. This shows the effectiveness of the \mathcal{R} layer for symmetric circuits.

Setting the number of iterations. In this experiment, we study the effect of changing the number of iterations on our network. We set the iteration to be 5, 10, 15, and 20, respectively, and test on a mixed dataset with instances from $SR(3)$ to $SR(10)$. We can see for AsymSAT with GRU, increasing the number of iteration from 5 to 10 will greatly improve the accuracy, then the solving rate barely increases for more iterations. AsymSAT with LSTM shows a similar result, while it peaks at around 15 iterations. It seems that AsymSAT with LSTM may have a higher potential to achieve a better accuracy. Therefore, in the following experiments on SAT solving, we mainly use AsymSAT with LSTM for comparison.

Experiments for SAT solving

In the following experiments, we compare the three models: our AsymSAT model with bi-directional LSTM in the \mathcal{R} layer, the NeuroSAT model, and the DG-DAGRNN model. We measure the performance using solution rate rather than the accuracy of predicting satisfiability. Solution rate is defined as the percentage of problems on which the network is able to predict one satisfying assignment.

We admit that for different models, sometimes it is hard to make an absolutely fair comparison. As AsymSAT is supervised under the multi-bit SAT solution, whereas NeuroSAT is one-bit SAT/UNSAT supervision, AsymSAT is much more training-data-efficient and it can converge several orders of magnitude faster than NeuroSAT, although their parameter

counts are roughly in the same scale (around 200K). Actually, the authors of NeuroSAT reported that it would take 10^7 SAT problems to train a full-fledged NeuroSAT [Selsam, 2018]. Although our training set is much smaller due to limitations of computing resources, we didn’t observe the overfitting problem in the experiments judging from the training loss.

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 2 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%. Furthermore, we supply more experimental results regarding larger $SR(n)$ problems. When trained from $SR(3)$ to $SR(10)$, AsymSAT outperforms NeuroSAT on $SR(20)$, $SR(40)$, $SR(60)$ and $SR(80)$. The result is shown in Table 6. In our experiment, the performance of DG-DAGRNN is non-competitive to the other two. We conjecture that the unsupervised learning method in DG-DAGRNN suffers from the vanishing gradient problem if trained on circuits converted from CNF. We provide a detailed analysis of DG-DAGRNN in the appendix.

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

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: Solution rate under different iterations

# of iterations	5	10	15	20
AsymSAT w. GRU	80.63%	90.32%	90.25%	89.11%
AsymSAT w. LSTM	79.76%	90.45%	93.07%	91.80%

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.

In summary, our AsymSAT model is capable of breaking the tie in symmetric circuits and it achieves a higher solution rate in comparison with NeuroSAT and DG-DAGRNN on both medium-size CNF problems and large-size Circuit-SAT problems. This shows the effectiveness of using RNN to account for variable dependency in GNN-based SAT solving.

6 Related Works

6.1 SAT Solvers

There are two main categories of machine-learning-based SAT solvers: the end-to-end SAT solvers and the solvers using machine-learning as just the heuristics. NeuroSAT [Selsam *et al.*, 2018], DG-DAGRNN [Amizadeh *et al.*, 2018] and our AsymSAT all belong to the first category, where machine learning methods are used to directly predict the SAT outcome. In the second category, machine learning methods only serve as a heuristic, guiding the classic algorithms. For example, NeuroCore [Selsam and Bjørner, 2019] used GNN to compute scores for variable selection in SAT solving and NLocal-SAT [Zhang *et al.*, 2020] used GNN to predict one potential solution as the starting point of the stochastic local search (SLS) process. There are also other techniques to support SAT solving. For example, QuerySAT proposed to use multiple SAT queries to increase accuracy [Ozolins *et al.*, 2021], and [Li *et al.*, 2022] suggested it is helpful to transfer SAT problems from different application domains to a unified underlying distribution.

Although in this paper we mainly investigate the importance of addressing variable dependency in the end-to-end ML SAT solvers (the first category), we argue that our technique is general and may benefit neural SAT solvers in the second category as well. For example, in NLocal-SAT, if we

Table 6: Solution rate for the larger $SR(n)$ problems

	$SR(20)$	$SR(40)$	$SR(60)$	$SR(80)$
AsymSAT	55.40%	27.20%	12.00%	5.00%
NeuroSAT	33.90%	19.60%	8.50%	4.50%

can provide a more accurate initial guess with the help of a tie-breaker proposed in this paper, the later stochastic local search process may be able to reach a satisfying solution with less searching effort.

6.2 Symmetric Breaking in GNN-based SAT Solving

Preferential Labeling [Sun *et al.*, 2022] is another method that can potentially break the tie between two symmetric variables in GNN-based SAT solving. It assigns distinct initial embeddings to variables, so symmetric nodes can therefore be distinguished. However, biased initialization also introduces artefact for GNN. In order to smooth out the artefact, each round of training or inference must evaluate the network under multiple random permutations of the initial embeddings. In the training phase, Preferential Labeling picks the permutation that produces the lowest loss and only optimizes the network parameters under this permutation. Meanwhile, the inference process takes the averaged output among all attempted permutations as the final prediction. Compared to Preferential Labeling, we regard our method as a lower-cost solution for tie-breaking in SAT solving. Our appendix details the comparison on performance and cost.

7 Conclusion

This paper addresses the need of considering variable dependency when designing a machine-learning model for SAT solving. Specifically, the satisfying assignment to one variable is closely related to those made to other variables within the same SAT problem. 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. Although this paper focuses on the end-to-end machine-learning-based SAT solvers, using RNNs to account for variable dependency may also benefit other hybrid SAT solvers that use machine learning as a guiding heuristic.

References

- [Amizadeh *et al.*, 2018] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2018.
- [Audemard and Simon, 2014] Gilles Audemard and Laurent Simon. Glucose in the SAT 2014 competition. *Proceedings of SAT Competition*, 2014:31, 2014.
- [Brand, 1993] Daniel Brand. Verification of large synthesized designs. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 534–537. IEEE, 1993.
- [Cho *et al.*, 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [Duan *et al.*, 2022] Haonan Duan, Pashootan Vaezipoor, Max B. Paulus, Yangjun Ruan, and Chris J Maddison. Augment with care: Contrastive learning for the Boolean satisfiability problem. *arXiv preprint arXiv:2202.08396*, 2022.
- [Gupta *et al.*, 2006] Aarti Gupta, Malay K. Ganai, and Chao Wang. SAT-based verification methods and applications in hardware verification. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 108–143. Springer, 2006.
- [Jacobs and Sakr, 2021] Swen Jacobs and Mouhammad Sakr. AIGEN: Random generation of symbolic transition systems. In *International Conference on Computer Aided Verification*, pages 435–446. Springer, 2021.
- [Li *et al.*, 2022] Min Li, Zhengyuan Shi, Qiuxia Lai, Sadaf Khan, and Qiang Xu. DeepSAT: An EDA-driven learning framework for SAT. *arXiv preprint arXiv:2205.13745*, 2022.
- [Marques-Silva and Sakallah, 1999] Joao P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
- [Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535, 2001.
- [Ozolins *et al.*, 2021] Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural SAT solver. *arXiv preprint arXiv:2106.07162*, 2021.
- [Prestwich, 2009] Steven David Prestwich. CNF encodings. *Handbook of satisfiability*, 185:75–97, 2009.
- [Selsam and Bjørner, 2019] Daniel Selsam and Nikolaj Bjørner. NeuroCore: Guiding high-performance SAT solvers with unsat-core predictions. 2019.
- [Selsam *et al.*, 2018] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.
- [Selsam, 2018] Daniel Selsam. Guidance on reproducing experiments in paper. <https://github.com/dselsam/neurosat/issues/2>, 2018.
- [Shi *et al.*, 2021] Feng Shi, Chonghan Lee, Mohammad Khairul Bashar, Nikhil Shukla, Song-Chun Zhu, and Vijaykrishnan Narayanan. Transformer-based machine learning for fast SAT solvers and logic synthesis. *arXiv preprint arXiv:2107.07116*, 2021.
- [Shuai *et al.*, 2016] Bing Shuai, Zhen Zuo, Bing Wang, and Gang Wang. DAG-recurrent neural networks for scene labeling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3620–3629, 2016.
- [Sorensson and Een, 2005] Niklas Sorensson and Niklas Een. Minisat v1. 13-a SAT solver with conflict-clause minimization. *SAT*, 2005(53):1–2, 2005.
- [Sun *et al.*, 2022] Zeyu Sun, Wenjie Zhang, Lili Mou, Qihao Zhu, Yingfei Xiong, and Lu Zhang. Generalized equivariance and preferential labeling for GNN node classification. 2022.
- [Tseitin, 1983] Grigori S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.
- [Yolcu and Póczos, 2019] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for Boolean satisfiability. *Advances in Neural Information Processing Systems*, 32, 2019.
- [Zhang *et al.*, 2020] Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. NLocSAT: Boosting local search with solution prediction. *arXiv preprint arXiv:2001.09398*, 2020.