# NLocalSAT: Boosting Local Search with Solution Prediction

**Wenjie Zhang**[1] , **Zeyu Sun**[1] , **Qihao Zhu**[1] , **Ge Li**[1] ,
**Shaowei Cai**[2,3] , **Yingfei Xiong**[1]  and  **Lu Zhang**[1*]

[1]Key Laboratory of High Confidence Software Technologies (Peking University), MoE;
Software Institute, Peking University, China

[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China

[3]School of Computer Science and Technology, University of Chinese Academy of Sciences, China

{zhang_wen_jie, szy_, zhuqh, lige, xiongyf, zhanglucs}@pku.edu.cn, caisw@ios.ac.cn

## Abstract

The Boolean satisfiability problem (SAT) is a famous NP-complete problem in computer science. An effective way for solving a satisfiable SAT problem is the stochastic local search (SLS). However, in this method, the initialization is assigned in a random manner, which impacts the effectiveness of SLS solvers. To address this problem, we propose *NLocalSAT*. *NLocalSAT* combines SLS with a solution prediction model, which boosts SLS by changing initialization assignments with a neural network. We evaluated *NLocalSAT* on five SLS solvers (CCAnr, Sparrow, CPSparrow, YalSAT, and probSAT) with instances in the random track of SAT Competition 2018. The experimental results show that solvers with *NLocalSAT* achieve $27\% \sim 62\%$ improvement over the original SLS solvers.

## 1 Introduction

Boolean satisfiability (also referred to as propositional satisfiability and abbreviated as SAT) is the problem to determine whether there exists a set of assignments for a given Boolean formula to make the formula evaluate to true. SAT is widely used in solving combinatorial problems, which are generated from various applications, such as program analysis [Harris *et al.*, 2010], program verification [Leino, 2010], and scheduling [Kasi and Sarma, 2013]. These applications first reduce the target problem into a SAT formula and then find a solution using a SAT solver. However, the SAT problem has proven to be NP-complete [Cook, 1971], which means that algorithms for solving SAT Instances may need exponential time in the worst case. Therefore, many techniques have been proposed to increase the efficiency of the search process of SAT solvers.

The state-of-the-art SAT solvers can be divided into two categories, CDCL (Conflict Driven Clause Learning) solvers and SLS (Stochastic Local Search) solvers. CDCL solvers are based on the deep backtracking search, which assigns one variable each time and backtracks when a conflict occurs. On the other hand, SLS solvers initialize an assignment for all variables and then find a solution by constantly flipping the assignment of variables to optimize some score.

Over the last few years, artificial neural networks have been widely used in many problems [Edwards and Xie, 2016; Selsam and Bjørner, 2019]. A neural network is a machine learning model with a large number of parameters. Neural networks have been used on many data structures, such as sequences [Mikolov *et al.*, 2010], images [Simonyan and Zisserman, 2015], and graphs [Edwards and Xie, 2016]. The graph convolutional network (GCN) [Edwards and Xie, 2016] is a neural network model on graph structures, which extracts both structural information and information on nodes in a graph. GCN performs well on many tasks on graphs.

There have been some studies on solving SAT problem with neural networks. Some of them use end-to-end neural networks to solve SAT problem directly as the outputs of the neural networks, while others use neural network predictions to boost CDCL solvers. Selsam et al. proposed an end-to-end neural network model to predict whether a SAT instance is satisfiable [Selsam *et al.*, 2019] in 2019. Later, Selsam et al. modified NeuroSAT to NeuroCore [Selsam and Bjørner, 2019]. NeuroCore guides CDCL solvers with unsat-core predictions which are computed every certain interval in the neural network on GPUs. CDCL solvers with NeuroCore solve 6%-11% more instances than the original.

In this paper, we propose *NLocalSAT*, which is the first method that uses a neural network to boost SLS solvers and the first off-line method to boost SAT solvers with neural networks. Different from NeuroCore which induces large overhead to CDCL by an on-line prediction[1], *NLocalSAT* uses the prediction in an off-line way. In this method, the neural network is computed only once for each SAT instance.

In our proposed method, we first train a neural network to predict the solution space of a SAT instance. Then, we combine SLS solvers with the neural network by modifying the solvers' initialization assignments under the guidance of the output of the neural network. Such combination induces limited overhead, and it can easily be applied to SLS solvers. Furthermore, we evaluated SLS solvers, and *NLocalSAT* solves $27\% \sim 62\%$ more instances than the original SLS solvers. Such experimental results show the effectiveness of

---

[1]The on-line prediction means to predict every certain interval in CDCL.
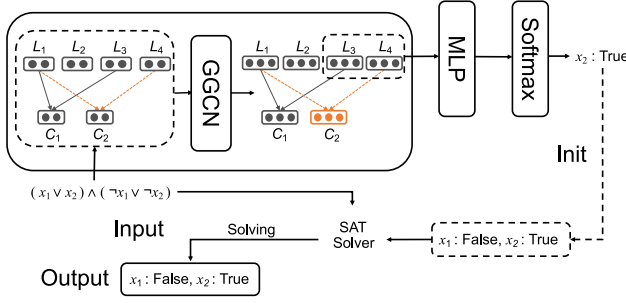
Figure 1: The overview of our model, *NLocalSAT*.

*NLocalSAT*.

**Contributions.** (1) We train a neural network to predict the solution of a SAT instance. (2) We propose a method to boost SLS solvers by modifying its initialization of assignments with the guidance of predictions of the neural network. (3) To the best of our knowledge, we are the first to combine SLS with a neural network model and we are the first to propose an off-line method to boost SAT solvers with a neural network.

## 2 Approach

Figure 1 shows an overview of our model, *NLocalSAT*. Our model combines a neural network and an SLS solver. Given a satisfiable input formula, the neural network is to output a candidate solution and the solver is to find a final solution with the guidance of the neural network. For an input formula, we first transfer it into a formula graph, which is further fed to a graph-based neural network for extracting features. Then the neural network in *NLocalSAT* outputs a candidate solution for the formula by a multi-layer perceptron after the neural network. We use this candidate solution to initialize the assignments of an SLS SAT solver to guide the search process.

### 2.1 Formula Graph

To take the structural information of an input formula into consideration, we first transfer it into a formula graph. A general Boolean formula can be any expressions consisting of variables, conjunctions, disjunctions, negations, and constants. All Boolean formulas can be reduced into an equisatisfiable conjunctive normal form (CNF) with linear length in linear time [Tseitin, 1983]. In a CNF, a SAT instance is a conjunction of clauses $C_1 \land C_2 \land \cdots \land C_n$. Each clause is a disjunction of literals (i.e., variables and negated variables) $C_i = L_{i1} \lor L_{i2} \lor \cdots \lor L_{in}$, where $L_{ij} = x_k$ or $L_{ij} = \neg x_k$. In this paper, we assume that all SAT problems are in CNFs. A SAT instance $S$ in the CNF can be seen as a bipartite graph $G = (C, L, E)$, where $C$ (clause set of $S$) and $L$ (literal set of $S$) are the node sets of $G$ and $E$ is the edge set of $G$. $(c, l)$ is in $E$ if and only if the literal $l$ is in the clause $c$. $A$ is the adjacent matrix of the bipartile graph $G$. The element $A_{ij}$ of the adjacent matrix equals to one when there is an edge between node $i$ and node $j$, otherwise 0. For example, $(x_1 \lor x_2) \land \neg (x_1 \land x_3)$ is a Boolean formula. This Boolean
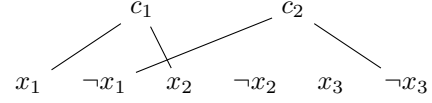


Figure 2: The bipartite graph representation for the CNF formula.

formula can be converted into $(x_1 \lor x_2) \land (\neg x_1 \lor \neg x_3)$ in the conjunctive normal form. The bipartite graph for this problem is shown in Figure 2. The adjacency matrix for this graph is

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

### 2.2 Graph-Based Neural Network

The graph-based neural network aims to predict the candidate solution for a SAT instance. The network consists of a gated graph convolutional network to extract structural information about the graph and a two-layer perceptron to predict the solution.

**Gated Graph Convolutional Network**

Inspired by NeuroSAT [Selsam *et al.*, 2019], we use a similar gated graph convolutional network (GGCN) to extract features of variables. The gated graph convolutional network (GGCN) takes the adjacency matrix as the input and outputs the features of each variable extracted from the graph.

In a SAT instance, the satisfiability is not influenced by the names of clauses and literals (e.g., the satisfiability of two formulas $(x_1 \lor x_2)$, $(x_3 \lor x_4)$). To use this property, for an input formula graph $G$, we initialize each clause $c_i \in G$ as a vector $\boldsymbol{c}^{(\text{init})} \in \mathbb{R}^d$, each literal $l_i \in G$ as another vector $\boldsymbol{l}^{(\text{init})} \in \mathbb{R}^d$, where $d$ is the embedding size and $d$ is set to 64 in this paper. These vectors are further fed to GGCN to extract structural information in the graph.

Each iteration of GGCN is an update for the vectors of these nodes where each node updates its vector by taking its neighbors' information (vectors). Formally, at the $t$-th iteration, the detailed computations for clause $c$ and literal $l$ are represented by

$$\boldsymbol{c}_t = \text{LSTMCell}(\boldsymbol{c}_{t-1}, \sum_{l' \in G} \tilde{A}_{cl'} \boldsymbol{l}'_{t-1}) \tag{1}$$

$$\boldsymbol{l}_t = \text{LSTMCell}(\boldsymbol{l}_{t-1}, \sum_{c' \in G} \tilde{A}_{c'l} \boldsymbol{c}'_t + \neg \boldsymbol{l}_{t-1}) \tag{2}$$

Here, $\tilde{A}$ is a normalized adjacency matrix for the graph (the detailed computation is presented in Equation 3). $\boldsymbol{l}'_t$ and $\boldsymbol{c}'_t$ denote the vector of the literal $l'$ and clause $c'$ at the $t$-th iteration. $\neg \boldsymbol{l}_t$ is the vector of negated literal of $l$ at the $t$-th iteration. $\boldsymbol{c}_0 = \boldsymbol{c}'_0 = \boldsymbol{c}^{(\text{init})}$, $\boldsymbol{l}_0 = \boldsymbol{l}'_0 = \boldsymbol{l}^{(\text{init})}$. The LSTMCell is a long short-term memory (LSTM) unit with layer normalization. We use symmetrical normalization on adjacency matrix.

$$\tilde{A} = S_1^{-1/2} A S_2^{-1/2}, \tag{3}$$

where $S_1$ and $S_2$ are the diagonal matrices with summation of $A$ in columns and rows.

We apply the GGCN layer of 16 iterations on the initial value and get a vector containing structural information about

each literal. Then, the two vectors for a literal and its negation are concatenated for each variable.

**Two-layer Perceptron**

After GGCN, vectors of nodes contain structural information of literals. A two-layer perceptron MLP with hidden 512 size is applied on the vector for each variable to extract classification information from the structural information. Through a softmax function, we get the probability for the variable to be true.

$$
\begin{aligned}
&P(v = \text{FALSE}), P(v = \text{TRUE}) \\
&= \text{softmax} \left\{ \mathbf{W}_2 \, \text{ReLU} \left[ \mathbf{W}_1 (\mathbf{l}_v : \mathbf{l}_{\neg v}) + \mathbf{b}_1 \right] + \mathbf{b}_2 \right\},
\end{aligned} \tag{4}
$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ are weights and biases for the two perceptrons and the colon indicates the connection of two vectors.

**Loss Function**

Our model is trained by minimizing the cross-entropy loss against the ground truth. For the predicted variables $< v_1, v_2 \cdots, v_n >$, where $n$ denotes the number of variables, the cross-entropy loss is computed as

$$
\begin{aligned}
Loss = -\sum_{i=1}^{n} [ &g(v_i) \log(P(v = \text{TRUE})) \\
&+ (1 - g(v_i)) \log(P(v = \text{FALSE}))],
\end{aligned} \tag{5}
$$

where $g(v_i)$ denotes the ground truth of $v_i$.

## 2.3 Training Data

The goal of our network is to predict the solution to the SAT instances, so we should generate training data for SAT instances with solution labeling. Due to the scarcity of SAT Competition data, using additionally generated small SAT instances could help provide thorough training. We generate two training datasets and train our model in order. Our model is first pretrained with a large number of generated small SAT instances. We generate tens of millions of small SAT instances with solution labeling. Such large amounts of data can make the training process more effective and avoid overfitting. These small instances can help our network better learn structural information. Then, our model is fine-tuned on a dataset generated from SAT Competitions. By finely tuning on the dataset from SAT Competition instances, our model can learn specific information in the field and learn to predict the solution on large instances.

**Small Generated Instances**

We generate small instances containing 10 to 20 variables by a random generator. The number of clauses is 2 to 6 times the number of variables. Each clause contains 3 variables with the probability of 0.8 and 2 variables with the probability of 0.2. Each clause is generated by randomly randomly selecting variables or negated variables from the instance.

After the generation of instances, we solve these instances with a SAT solver. We drop unsatisfied instances because our model only learns to predict the solution of a satisfiable SAT instance. However, there can be more than one solution for a specific SAT instance, which can confuse the neural network. To address this problem, we use a complete solver to find all

---

**Algorithm 1** Algorithm for stochastic local search solvers

**Data**: SAT instance $P$

  **while** End condition not reached **do**
    $S \leftarrow$ initialize assignment randomly;
    **while** Restart condition not reached **do**
      **if** $P$ evaluate to true under $S$ **then**
        Return $S$;
      **end if**
      $l \leftarrow$ Select a variable by some heuristics;
      Flip($S, l$);
    **end while**
  **end while**

---

solutions for each SAT instance. Then we label each variable based on whether itself or its negation appears more among all solutions.

After data generation, we pretrain *NLocalSAT* on these small instances. Small data instances have high overhead during training the neural network, so we combine several small instances into one large batch. Since our model is independent of the order and the size of SAT variables, we can combine many instances via simply putting them together.

**SAT Competition Instances**

To train our model on larger instances, we generate training data from SAT instances in random tracks of SAT Competitions. We use a SAT solver to get one solution for each instance. Instances that are not solved by the solver will be removed. We then use the solution as labeling in the training data.

Training on GPU requires more memory than evaluation. So, if the instance is too large to fit in the memory of our GPU during training, we will cut these instances into smaller ones. Let us denote the largest number of variables that can fit into the memory of our GPU as $N_L$. For every large instance, we first get a solution $S_0$ with a SAT solver. Then, we sample $N_L$ variables $X_0$ from all variables. For one clause $c$ in the original SAT instance, if $c$ contains no literals from $X_0$, the clause is removed. If $c$ is not satisfied on $X_0$ after removing literals that are not from $X_0$, the clause is also removed. Clauses with only one literal are also removed to prevent the instance from being too easy. Otherwise, the clause $c$ remains in the instance. If the sampling generates a instance with too few clauses, the instance is removed because this will lead to too many solutions.

## 2.4 Combination with Local Search

Stochastic local search algorithms can be considered as an optimization process. The solver flips the assignments of variables to maximize some total score. For example, we can use the number of clauses that evaluate to true as the score. When the score reaches the number of clauses, the instance is solved. Algorithm 1 shows a general algorithm in a local search solver.

In SLS solvers, it is not counter-intuitive that the initial assignment has a great impact on whether it can quickly find a solution, because there can be many local optimum in the instance and badly initialized assignments near a local optimum

---

**Algorithm 2** Initialization of variables with *NLocalSAT*

---

**Data**: Probability $p_0$, Assignment of variables assignment, Neural network predictions $N$

  **for** $i <$ number of variables **do**
    **if** rand$() < p_0$ **then**
      assignment$[i] \leftarrow N[i]$
    **else**
      assignment$[i] \leftarrow \neg N[i]$
    **end if**
  **end for**

---

**Algorithm 3** Algorithm for stochastic local search solvers with *NLocalSAT* (The underlined code is the different part from the original solvers)

---

**Data**: SAT instance $P$

  $N \leftarrow$ predicting solution of $P$ by *NLocalSAT*;
  **while** End condition not reached **do**
    $S \leftarrow$ initialize assignment with $N$;
    **while** Restart condition not reached **do**
      **if** $P$ evaluate to true under $S$ **then**
        Return $S$;
      **end if**
      $l \leftarrow$ Select a variable by some heuristics;
      Flip$(S, l)$;
    **end while**
  **end while**

---

can cause the SLS solvers to get stuck. Intuitively, the closer to the solution of the instance, the easier it is to find a solution. In order to avoid falling into the local optimum, these solvers restart the searching process by reassigning new random values to variables after a period of time without a score increase. However, most of the existing SLS solvers initialize assignments in a random manner. Random generation of initial values can explore more space for the SAT instance. However, if the distance between initial values and solution values is too large, the solver is more likely to fall into a local optimum.

We propose a new initialization method using the output of our neural network. Before starting the solver, we run our neural network to predict a solution. We replace the initialization function with our neural initialization, where, instead of randomly generating 0 or 1, the function assigns the predicted values to a variable with a probability of $p_0$ and assigns the negation of the predicted value with a probability of $1 - p_0$. This neural-initialization function can keep the assignment with a probability of $p_0$ to explore near the candidate solution and explores new solution space with a probability of $1 - p_0$ in case that the neural network's prediction is wrong. The initialization process is shown in Algorithm 2. Algorithm 3 shows the architecture of our modified SLS solvers. Note that our neural network model is executed only once for one SAT instance. Though the cost of computing a neural network is high, the cost of calling a neural network only once is acceptable, which consumes 0.1 seconds to tens of seconds depending on the size of the instance.

# 3 Experiments

## 3.1 Datasets

Our model was trained on a dataset with generated instances with small SAT instances (denoted as $Dataset_{small}$) and a dataset with instances in random tracks of SAT Competitions in 2012, 2013, 2014, 2016, 2017 (denoted as $Dataset_{comp}$) [2]. Our model was evaluated on instances in the random track of SAT Competition in 2018 (denoted as $Dataset_{eval}$) with 255 SAT instances in total. We found that there are several duplicate instances in 2018 and previous years, so we removed them from the training and validation datasets to ensure instances in the $Dataset_{eval}$ are generated with different random seeds with those in $Dataset_{comp}$. So, it's almost impossible to have isomorphic instances between these two datasets. However, there will be some similar substructures between the training set and the test set, so that neural networks can predict by learning these substructures.

The $Dataset_{comp}$ and the $Dataset_{eval}$ both contain two categories of instances, i.e., uniformly generated random SAT instances (denoted as *Uniform*) [Heule, 2018] and hard SAT instances generated with a predefined solution (denoted as *Predefined*) [Balyo and Chrpa, 2018].

## 3.2 Pretraining

In $Dataset_{small}$, we generated about $2.5 \times 10^7$ small instances and combined them into about $4 \times 10^5$ batches with about ten thousand variables each as our pretraining dataset. We generated 200 batches in the same approach with different random seeds as validation data during pretraining.

We trained our model to converge using the Adam [Kingma and Ba, 2015] optimizer with its default parameters by minimizing the loss function. After pretraining, the precision on the validation dataset of $Dataset_{small}$ is 98%.

## 3.3 Training

We used $Dataset_{comp}$ as the training dataset and the validation dataset. We loaded the pretrained model and continued to train with the same optimizer and loss function. After training, the precision on the validation dataset of $Dataset_{comp}$ is 95%.

## 3.4 Evaluation

We tested our proposed method on five recent SLS solvers, i.e., CCAnr [Cai *et al.*, 2015], Sparrow [Balint and Fröhlich, 2010], CPSparrow, YalSAT [Biere, 2016], probSAT [Balint and Schöning, 2018]. These solvers have performed very well among SLS solvers on random tracks of SAT Competitions in recent years. CCAnr is an SLS solver proposed in 2015 to capture structural information on SAT instances. CCAnr is a variant of CCASat [Cai and Su, 2013]. CCAnr performs better on all tracks of SAT Competitions than CCASat. Sparrow is a clause weighting SLS solver. CPSparrow is a combination of Sparrow and a preprocessor Coprocessor [Balint and Manthey, 2013]. CPSparrow is the

---

best pure SLS solver in the random track of SAT Competition 2018. YalSAT is the champion of the random track of SAT Competition 2017.

Due to the strong randomness of SLS solvers, the experiments for SLS solvers were performed three times with three different random seeds and then we aggregated the results.

We evaluated these original solvers and those modified with *NLocalSAT* on $Dataset_{eval}$. We also evaluated three other solvers MapleLCMDistChronoBT [Ryvchin and Nadel, 2018], gluHack, and Sparrow2Riss [Balint and Manthey, 2018] under the same set. MapleLCMDistChronoBT and Sparrow2Riss are the champions of SAT Competition 2018 in the main track and the random track. gluHack is the best CDCL solver of SAT Competition 2018 in the random track. MapleLCMDistChronoBT is a CDCL solver with recently proposed techniques to improve performance such as chronological backtracking [Nadel and Ryvchin, 2018], learned clause minimization [Luo *et al.*, 2017], and so on. Sparrow2Riss is a combination of Coprocessor, Sparrow, and a CDCL solver Riss.

We set up a timeout limit to 1000 seconds. Solvers failed to find a solution within the time limit will be killed immediately. In our experiments, $p_0$ is set to 0.9. Our experiments were performed on a work station with an Intel Xeon E5-2620 CPU and a TITAN RTX GPU. During our experiment, the time for initialization of the GPU environment was ignored but the time of the GPU computation was included in the total time.

## 4 Results

### 4.1 Number of Instance Solved

Table 1 shows the number of instances solved in 1000 seconds time limit. Each row represents a tested solver. The experiments of SLS solvers are performed three times to reduce the randomness of results. Each number in the rows of SLS solvers is the average and the standard deviation of results in the three experiments. Each column in the table represents a category of instances in the dataset. The number in parentheses indicates the total number of instances in the category. $Dataset_{eval}$ contains unsatisfiable instances [Heule, 2018]. However, no solvers reported unsatisfiability on any instances within the time limit. So, the solved problems in Table 1 are all satisfiable ones.

The experimental result shows that solvers with *NLocalSAT* solve more instances than the original ones. CCAnr, Sparrow, CPSparrow, YalSAT, and probSAT with *NLocalSAT* solve respectively 41%, 30%, and 27%, 62%, and 62% more instances than the original solvers. This improvement has been shown in *Predefined* instances and in *Uniform* instances on Sparrow and CPSparrow. Sparrow with *NLocalSAT* and CPSparrow with *NLocalSAT* solve more instances than all other solvers including the champions on SAT Competition 2018. Sparrow2Riss is a combination of a preprocessor, an SLS solver, and a CDCL solver, thus showing good performance, but the SLS solvers with *NLocalSAT* still outperforms Sparrow2Riss. CDCL solvers perform well on *Predefined* instances and *NLocalSAT* can help to improve performance particularly on this category, from which we can conclude that

| Solver | Predefined(165) | Uniform(90) | Total(255) |
|---|---|---|---|
| CCAnr | 107.3 ± 1.2 | 18.0 ± 0.8 | 125.3 ± 1.2 |
| CCAnr with *NLocalSAT* | 165.0 ± 0.0 | 12.7 ± 0.9 | **177.7 ± 0.9** |
| Sparrow | 126.7 ± 0.5 | 23.7 ± 1.7 | 150.3 ± 1.2 |
| Sparrow with *NLocalSAT* | 165.0 ± 0.0 | 31.0 ± 0.8 | **196.0 ± 0.8** |
| CPSparrow | 128.0 ± 0.8 | 27.0 ± 1.6 | 155.0 ± 1.4 |
| CPSparrow with *NLocalSAT* | 165.0 ± 0.0 | 32.0 ± 0.8 | **197.0 ± 0.8** |
| YalSAT | 75.0 ± 0.0 | 49.5 ± 1.5 | 124.5 ± 1.5 |
| YalSAT with *NLocalSAT* | 165.0 ± 0.0 | 37.3 ± 0.9 | **202.3 ± 0.9** |
| probSAT | 75.7 ± 0.5 | 51.0 ± 0.0 | 126.7 ± 0.5 |
| probSAT with *NLocalSAT* | 165.0 ± 0.0 | 40.7 ± 1.2 | **205.7 ± 1.2** |
| Sparrow2Riss | 165 | 23 | 188 |
| gluHack | 165 | 0 | 165 |
| MapleLCMDistBT | 165 | 0 | 165 |

Table 1: Number of instances solved in time limit.

| Solver | Predefined(165) | Uniform(90) | Total(255) |
|---|---|---|---|
| CCAnr | 747 ± 15 | 1644 ± 11 | 1063 ± 7 |
| CCAnr with *NLocalSAT* | 0.30 ± 0.00 | 1736 ± 16 | **613 ± 5** |
| Sparrow | 472 ± 7 | 1531 ± 24 | 846 ± 3 |
| Sparrow with *NLocalSAT* | 0.26 ± 0.00 | 1379 ± 9 | **487 ± 3** |
| CPSparrow | 457 ± 11 | 1454 ± 38 | 809 ± 13 |
| CPSparrow with *NLocalSAT* | 0.35 ± 0.06 | 1385 ± 9 | **489 ± 3** |
| YalSAT | 1095 ± 3 | 962 ± 33 | 1048 ± 9 |
| YalSAT with *NLocalSAT* | 0.26 ± 0.00 | 1226 ± 23 | **433 ± 8** |
| probSAT | 1086 ± 5 | 928 ± 3 | 1030 ± 3 |
| probSAT with *NLocalSAT* | 0.26 ± 0.00 | 1155 ± 26 | **408 ± 9** |
| Sparrow2Riss | 107.7 | 1560 | 620 |
| gluHack | 15.0 | 2000 | 715 |
| MapleLCMDistBT | 8.8 | 2000 | 711 |

Table 2: Average running time with timeout penalty (PAR-2).

*NLocalSAT* can improve particularly on those instances, on which CDCL solvers perform well.

### 4.2 Time of Solving Instances

Figure 3 shows the relationship between the number of instances solved and time consumption comparing solvers with *NLocalSAT* and without *NLocalSAT*. In this figure, we can see that some simple instances which are solved within 1 second with the original solver need more solving time with *NLocalSAT* than the original solver. This is because the neural network computation takes a certain amount of time before the solver starts and this time is especially noticeable for simple instances. But on hard instances, our modifications can improve the solver significantly.

Table 2 compares the average running time with the timeout penalty (PAR-2) between different solvers. The PAR-2 running time of instances that are not solved by the solver in the time limit is twice of the time limit (i.e., 2000 seconds in our experiments). The PAR-2 score was used in previous SAT Competitions. Values in this table show that solvers with *NLocalSAT* are slightly slower on easy instances but much faster on hard instances, particularly, on *Predefined*. Solvers with *NLocalSAT* can find a solution faster than those without *NLocalSAT* on most instances, i.e., solvers with *NLocalSAT* are more effective than the original ones.
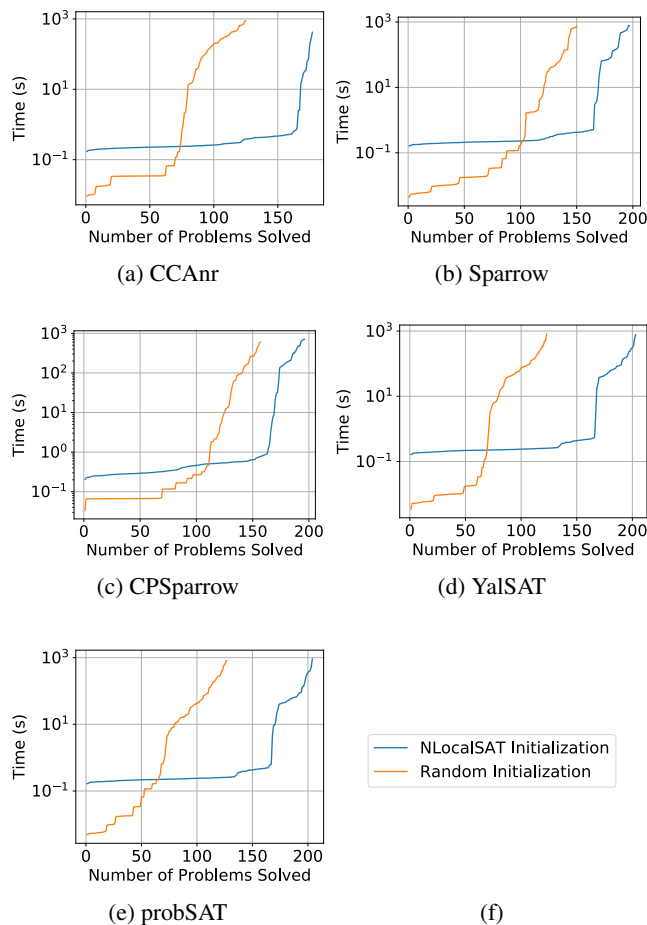
Figure 3: Cactus plots comparing *NLocalSAT* initialization with random initialization over CCAnr, Sparrow, and CPSparrow.

## 4.3 Discussion

To find out why our method can boost SLS solvers, we analyzed the experimental result on CCAnr. The geometric mean of ratios of steps that the solver took to find a solution with *NLocalSAT* to the steps without *NLocalSAT* is 0.005. Namely, the solver with *NLocalSAT* takes much fewer steps to find a solution overall. Among all solved instances, the average proportion of correctly predicted variables (i.e., those where the predicted value by the neural network and the final value by the solver are the same) is 0.88, and, with a chi-square analysis, this rate and the speedup of our approach shows a correlation with the p-value $3 \times 10^{-5}$. This experimentally verified our intuition that the closer the initial values are to the solution of the instance, the easier the solver can find a solution. Our neural network can give better initial values, which can boost SLS solvers.

## 5 Related Work

Recently, several studies have investigated how to make use of neural networks in solving NP-complete constraint problems. There are two categories of methods to solve NP-complete problems using neural networks. The first cat-

egory of methods is end-to-end approaches using end-to-end neural networks to solve SAT instances, i.e., given the instance as an input, the neural network outputs the solution directly. In these methods, the neural network can learn to solve the instance itself [Amizadeh *et al.*, 2019; Galassi *et al.*, 2018; Selsam *et al.*, 2019; Xu *et al.*, 2018; Prates *et al.*, 2019]. However, due to the accuracy and structural limitations of neural networks, the end-to-end methods can only solve small instances. The other category of methods is heuristic methods that treat neural networks as heuristics [Balunovic *et al.*, 2018; Li *et al.*, 2018; Selsam and Bjørner, 2019]. Among these methods, traditional solvers work with neural networks together. Neural networks generate some predictions, and the solvers use these predictions as heuristics to solve the instance. Constraints in the instances can be maintained in the solver, so these methods can solve large-scale instances. Our proposed method (i.e., *NLocalSAT*) belongs to the second category, heuristic methods, so *NLocalSAT* can be used for larger instances than those end-to-end methods. Balunovic et al. [Balunovic *et al.*, 2018] proposed a method to learn a strategy for Z3, which greatly improves the efficiency of Z3. Li et al. [Li *et al.*, 2018] proposed a model on solving maximal independent set problems with a tree search algorithm. NeuroCore [Selsam and Bjørner, 2019] is a method to improve CDCL solvers with predictions of unsat-cores. None of these methods is used to boost stochastic local search solvers with solution predictions and none of these is an off-line method to boost SAT solvers. The training data used in *NLocalSAT* are solutions of instances or solution space distribution of instances, which is also different from previous works, where NeuroSAT uses the satisfiability of instances and NeuroCore uses unsat-core predictions.

## 6 Conclusion and Future Work

This paper explores a novel perspective of combining SLS with a solution prediction model. We propose *NLocalSAT* to boost SLS solvers. Experimental results show that *NLocalSAT* significantly increases the number of instances solved and decreases the solving time for hard instances. In particular, Sparrow and CPSparrow with our proposed *NLocalSAT* perform better than state-of-the-art CDCL, SLS, and hybrid solvers on the random track of SAT Competition 2018.

*NLocalSAT* can boost SLS SAT solvers effectively. With this learning-based method, we may build a domain-specific SAT solver without expertise in the domain by training *NLocalSAT* with SAT instances from the domain. It is also interesting to further explore building domain-specific solvers with *NLocalSAT*.

## Acknowledgements

# References

[Amizadeh *et al.*, 2019] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *ICLR*, 2019.

[Balint and Fröhlich, 2010] Adrian Balint and Andreas Fröhlich. Improving stochastic local search for SAT with a new probability distribution. In *SAT*, volume 6175 of *Lecture Notes in Computer Science*, pages 10–15, 2010.

[Balint and Manthey, 2013] Adrian Balint and Norbert Manthey. Boosting the performance of SLS and CDCL solvers by preprocessor tuning. In *POS-13. Fourth Pragmatics of SAT workshop*, volume 29 of *EPiC Series in Computing*, pages 1–14, 2013.

[Balint and Manthey, 2018] Adrian Balint and Norbert Manthey. Sparrowtoriss 2018. *Proceedings of SAT Competition*, 2018.

[Balint and Schöning, 2018] Adrian Balint and Uwe Schöning. probsat. *Proceedings of SAT Competition*, 2018.

[Balunovic *et al.*, 2018] Mislav Balunovic, Pavol Bielik, and Martin T. Vechev. Learning to solve SMT formulas. In *NeurIPS*, pages 10338–10349, 2018.

[Balyo and Chrpa, 2018] Tomás Balyo and Lukás Chrpa. Using algorithm configuration tools to generate hard SAT benchmarks. In *SOCS*, pages 133–137, 2018.

[Biere, 2016] Armin Biere. Splatz, lingeling, plingeling, treengeling, yalsat entering the sat competition 2016. *Proceedings of SAT Competition*, 2016.

[Cai and Su, 2013] Shaowei Cai and Kaile Su. Local search for boolean satisfiability with configuration checking and subscore. *Artif. Intell.*, 204:75–98, 2013.

[Cai *et al.*, 2015] Shaowei Cai, Chuan Luo, and Kaile Su. Ccanr: A configuration checking based local search solver for non-random satisfiability. In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 1–8, 2015.

[Cook, 1971] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[Edwards and Xie, 2016] Michael Edwards and Xianghua Xie. Graph based convolutional neural network. *arXiv preprint arXiv:1609.08965*, 2016.

[Galassi *et al.*, 2018] Andrea Galassi, Michele Lombardi, Paola Mello, and Michela Milano. Model agnostic solution of csps via deep learning: A preliminary study. In *CPAIOR*, volume 10848 of *Lecture Notes in Computer Science*, pages 254–262, 2018.

[Harris *et al.*, 2010] William R. Harris, Sriram Sankaranarayanan, Franjo Ivancic, and Aarti Gupta. Program analysis via satisfiability modulo path programs. In *POPL*, pages 71–82, 2010.

[Heule, 2018] Marijn JH Heule. Generating the uniform random benchmarks. *Proceedings of SAT Competition*, page 55, 2018.

[Kasi and Sarma, 2013] Bakhtiar Khan Kasi and Anita Sarma. Cassandra: proactive conflict minimization through optimized task scheduling. In *ICSE*, pages 732–741, 2013.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Leino, 2010] K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR-16*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370, 2010.

[Li *et al.*, 2018] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *NeurIPS*, pages 537–546, 2018.

[Luo *et al.*, 2017] Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for CDCL SAT solvers. In *IJCAI*, pages 703–711, 2017.

[Mikolov *et al.*, 2010] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048, 2010.

[Nadel and Ryvchin, 2018] Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *SAT*, volume 10929 of *Lecture Notes in Computer Science*, pages 111–121, 2018.

[Prates *et al.*, 2019] Marcelo O. R. Prates, Pedro H. C. Avelar, Henrique Lemos, Luís C. Lamb, and Moshe Y. Vardi. Learning to solve np-complete problems: A graph neural network for decision TSP. In *AAAI*, pages 4731–4738, 2019.

[Ryvchin and Nadel, 2018] Vadim Ryvchin and Alexander Nadel. Maple_lcm_dist_chronobt: Featuring chronological backtracking. *Proceedings of SAT Competition*, 2018.

[Selsam and Bjørner, 2019] Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *SAT*, volume 11628 of *Lecture Notes in Computer Science*, pages 336–353, 2019.

[Selsam *et al.*, 2019] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *ICLR*, 2019.

[Simonyan and Zisserman, 2015] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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

[Xu *et al.*, 2018] Hong Xu, Sven Koenig, and T. K. Satish Kumar. Towards effective deep learning for constraint satisfaction problems. In *CP*, volume 11008 of *Lecture Notes in Computer Science*, pages 588–597, 2018.