Machine Intelligence Research
www.mi-research.net

20(5), October 2023, 640-655
DOI: 10.1007/s11633-022-1396-2

Machine Learning Methods in Solving the Boolean Satisfiability Problem

Wenxuan Guo[1] Hui-Ling Zhen[2] Xijun Li[2] Wanqian Luo[2]
Mingxuan Yuan[2] Yaohui Jin[1] Junchi Yan[1]

[1] MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai 200240, China

[2] Noah's Ark Laboratory, Huawei Ltd., Shenzhen 518129, China

Abstract: This paper reviews the recent literature on solving the Boolean satisfiability problem (SAT), an archetypal $\mathcal{NP}$-complete problem, with the aid of machine learning (ML) techniques. Over the last decade, the machine learning society advances rapidly and surpasses human performance on several tasks. This trend also inspires a number of works that apply machine learning methods for SAT solving. In this survey, we examine the evolving ML SAT solvers from naive classifiers with handcrafted features to emerging end-to-end SAT solvers, as well as recent progress on combinations of existing conflict-driven clause learning (CDCL) and local search solvers with machine learning methods. Overall, solving SAT with machine learning is a promising yet challenging research topic. We conclude the limitations of current works and suggest possible future directions. The collected paper list is available at https://github.com/Thinklab-SJTU/awesome-ml4co.

Keywords: Machine learning (ML), Boolean satisfiability (SAT), deep learning, graph neural networks (GNNs), combinatorial optimization.

Citation: W. Guo, H. L. Zhen, X. Li, W. Luo, M. Yuan, Y. Jin, J. Yan. Machine learning methods in solving the Boolean satisfiability problem. *Machine Intelligence Research*, vol.20, no.5, pp.640–655, 2023. http://doi.org/10.1007/s11633-022-1396-2

1 Introduction

The Boolean satisfiability problem, often referred to as SAT, is the first proven $\mathcal{NP}$-complete problem[1] in the field of computational complexity. This hard combinatorial problem consistently attracts researchers' attention for its wide application and for the variety of problems that can be reduced to SAT. For theoretical interests, numerous combinatorial problems can be expressed in propositional formulae and solved by running a SAT solver[2], e.g., graph coloring[3], vertex cover[4] and clique detection[5]. It also serves as a useful tool for automated theorem proving, one typical case of which is the resolution of Keller's conjecture[6]. Moreover, there are plenty of industrial applications of SAT solving, such as bounded model checking, configuration management, and equivalence checking in circuit design. It is a major component in logic synthesis, and many SAT solvers are designed specially for this task[7]. Hence, SAT solving not only promotes research progress but also enables a more economical workflow.

Since the $\mathcal{P}$ versus $\mathcal{NP}$ problem remains unsettled, researchers respect the difficulty of the SAT problem and struggle to design efficient SAT solvers. Meanwhile, machine learning (ML), especially the surging deep learning techniques has advanced into the combinatorial optimization field and yielded a number of promising new avenues of research[8]. Different areas such as graph matching[9] and mixture integer programming[10], have been well studied in recent years, and the SAT community is no exception. As one of the most frontier events for SAT solvers, the past SAT competition has also witnessed several winning solvers featuring machine learning techniques, including Kissat_MAB[11, 12] (ranked first in SAT competition 2021) and MapleCOMSPS[13] (ranked first in SAT competition 2016). In addition to the traditional solvers enhanced with machine learning components, end-to-end frameworks such as NeuroSAT[14] seek integration of machine learning and SAT solving, which enables automatic and organic deduction and saves human labor.

We note a few previous surveys related to machine learning techniques in solving SAT and other hard problems[15, 16]. Specifically, Popescu et al.[15] focused on a broader scope of constraint-solving problems. Another survey[16] is the closest to ours, where the author reviewed machine learning methods in solving SAT and quantified SAT (QSAT), especially for automated theorem proving. The reviewed methods in [16] were categor-

Review
Manuscript received on July 1, 2022; accepted on November 23, 2022; published online on June 1, 2023
Recommended by Associate Editor Li-Wei Wang
Colored figures are available in the online version at https://link.springer.com/journal/11633

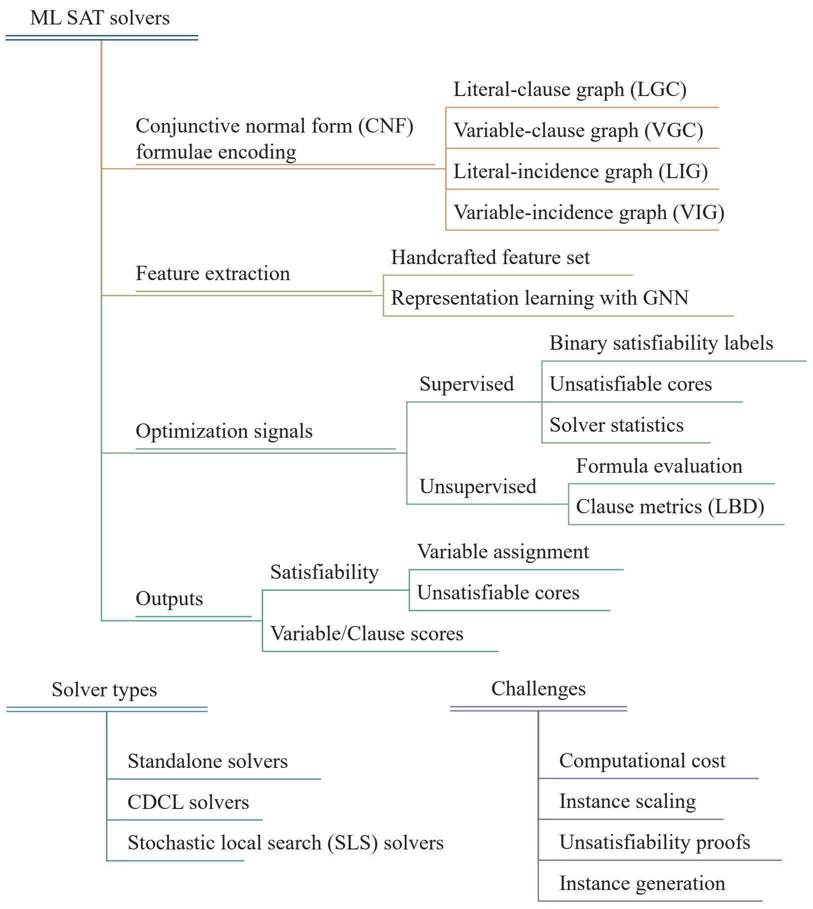


Fig. 1 Overview of SAT solvers with ML techniques

ized by the way of integration of machine learning into SAT solving. Based on its taxonomy, this survey discusses three primary patterns for this combination for SAT solving: 1) Standalone SAT solvers with pure machine learning methods; 2) Replacing some key components of existing conflict-driven clause learning (CDCL) solvers with learning-directed heuristics; 3) Modifying the local search solvers with learning-aided modules. In addition to solving the SAT problem itself with machine learning methods, we also provide a survey of instance generation for SAT with machine learning, which is an emerging yet promising topic in this field. Compared to [16], our survey focuses on more recent advancements and unfolds in a more compact and succinct way so that readers can be quickly informed of this rising area.

This survey encompasses directly optimizing SAT solving with the aid of machine learning techniques, e.g., multi-layer perceptron (MLP), naive Bayes, and neural networks, in the aforementioned three ways. Portfolio solvers and algorithm runtime prediction are not involved in this paper since it is a general technique applicable to other problems as well (see [17] for a survey). The extensions of SAT, e.g., maximum satisfiability problem (MAX-SAT), satisfiability modulo theories (SMT), and quantified Boolean formula problem (QBF), are also beyond the scope of this survey.

The rest of the paper is organized as follows. After giving some preliminaries in Section 2, we review three patterns of machine-learning-based SAT solving: Section 3.1 reviews standalone ML-SAT solvers, Section 3.2 discusses CDCL solvers enhanced with ML components, and Section 3.3 discusses learning-aided local search SAT solvers. In Section 4, machine learning methods for SAT instance generation are reviewed. Finally, we conclude in Section 5. An overview of this survey is illustrated in Fig. 1.

2 Preliminaries

This section first introduces basic definitions of the SAT problem and some classic learning-free SAT solvers that serve as paradigms for creating new solvers, with the key heuristics suitable for ML-based modification highlighted. It also provides some background to machine learning necessary for this survey.

2.1 Boolean satisfiability problem

In propositional logic, a Boolean formula is built from Boolean variables (only allowed to take value True or False) and three logic operators: conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$). The Boolean satisfiability problem aims to determine whether there exists a way of variable assignment so that a given Boolean formula evaluates to True. In the positive case, the formula is satisfiable, as opposed to unsatisfiable ones. If a SAT instance is satisfiable, it only takes polynomial time to verify an assignment. Otherwise, its unsatisfiability can be verified by a proof, such as clausal proofs.

Since every propositional formula can be transformed into an equivalent formula in conjunctive normal form (CNF), we only consider this form in the following discussion. A formula in CNF consists of a conjunction of clauses, where each clause is a disjunction of literals (a variable or its negation).

The complexity of SAT was proved by the Cook–Levin theorem[1], stating that SAT is $\mathcal{NP}$-complete. In other words, if there exists a deterministic polynomial algorithm for SAT, then every $\mathcal{NP}$ problem can be solved by a deterministic polynomial algorithm. Currently, SAT instances are mostly solved by optimized searching-based methods, with the exponential worst-case complexity.

2.2 Classic SAT solvers

Formally, a solver is a procedure aiming to solve the SAT problem: Given an input Boolean formula, a solver is supposed to yield the judgment of its satisfiability and provide a valid assignment if it is satisfiable. A complete solver is able to deduce that a SAT instance is unsatisfiable with a proof, as opposed to incomplete algorithms (see [18] for more details on unsatisfiability proofs).

Algorithm 1. Typical CDCL algorithm (adapted from Algorithm 4.1 in [19])

```
Input: A CNF formula $\varphi$
Function CDCL($\varphi$)
    $\nu = \emptyset$;          /* current assignment */
    if UnitPropagation ($\varphi$, $\nu$) == CONFLICT return UNSAT;
    $dl \leftarrow 0$;          /* decision level */
    while not AllVariablesAssigned ($\varphi$, $\nu$) do
        $(x, v)$ = PickBranchingVariable($\varphi$, $\nu$);
        $dl \leftarrow dl + 1$;
        $\nu \leftarrow \nu \cup \{(x, v)\}$;
        if UnitPropagation ($\varphi$, $\nu$) == CONFLICT
            $\beta$=ConflictAnalysis ($\varphi$, $\nu$);
            if $\beta < 0$ return UNSAT;
            else
                Backtrack($\varphi$, $\nu$, $\beta$);
                $dl \leftarrow \beta$;
    return SAT;
```

2.2.1 CDCL solvers

The CDCL algorithm, first proposed in the solver GRASP[20], is a popular complete SAT algorithm. As an improvement of the davis-putnam-logemann-loveland (DPLL) algorithm[21], its backbone is a backtracking-based search algorithm that selects a variable at a time for tentative assignment and backtracks chronologically once the reduced formula contains an empty clause. The primary feature of the CDCL algorithm, as suggested by its name, is that it learns new clauses from conflicts (invalid partial assignments which lead to unsatisfiability) and adds them to the original formula. The standard organization of a CDCL SAT solver is described in Algorithm 1, where we pay special attention to the following two concepts.

Variable selection

Variable selection heuristics, or branching heuristics, find the most "promising" unassigned variable to branch on. Popular heuristics include the VSIDS heuristic[22] and its variants[23] (e.g., EVSIDS[24]). Basically, a score is recorded for each variable, and each time the variable with the greatest score is selected. When a clause is learned by a CDCL solver, the score of the involved variables is increased by some amount. At regular intervals, a procedure called decaying is executed. For example, all scores are divided by some constants. In this way, variables in more recent conflicts are preferred. Other effective scoring heuristics were proposed and used in practice[25, 26].

Algorithm 2. WalkSAT algorithm[27] (adapted from Algorithm 6.2 in [19])

```
Input: A CNF formula $\varphi$
Parameters: Integers $MaxTries, MaxFlips$; noise parameter $p \in [0, 1]$
Function WalkSAT ($\varphi$)
    for $i \leftarrow 1$ to $MaxTries$ do
        $\sigma \leftarrow$ a random truth assignment for $\varphi$;
        for $j \leftarrow 1$ to $MaxFlips$ do
            if $\sigma$ satisfies $\varphi$ return $\sigma$;
            $C \leftarrow$ a random unsatisfied clause of $\varphi$;
            if $\exists$ variable $x \in C$ with $break\text{-}count = 0$
                $v \leftarrow x$;
            else
                With probability $p$:
                    $v \leftarrow$ a random variable in $C$;
                With probability $1 - p$:
                    $v \leftarrow$ a variable in $C$ with the smallest break-count;
            Flip $v$ in $\sigma$;
    return FAIL;
```

Literal block distance and glue clauses

CDCL solvers benefit from learning from conflicts. However, this convenience might consume a huge amount of memory space since the number of learned clauses grows exponentially. Therefore, it must perform clause deletion regularly. Literal block distance (LBD) is a metric proposed by [28], defined as the number of distinct decision levels of the variables in a clause. LBD can measure the quality of clauses due to the empirical observation that decision levels regularly decrease during search[28]. It also points out that clauses with an LBD of 2 are of vital importance and are thus termed "glue clauses".

2.2.2 Stochastic local search (SLS) solvers

The SLS algorithms are effective for solving random and hard combinatorial instances, a typical example of which is WalkSAT[27], as shown in Algorithm 2. As an incomplete solver, it starts from an initial variable assignment and flips the value of a selected variable at each iteration, until a legal assignment is found or the time limit is exceeded. To avoid getting trapped in the local minima, stochastic restarts are performed during the search if the restart criterion is met. The key heuristics involved in an SLS solver are the restart policy, initialization scheme, and variable selection for flipping. For example, GSAT[29] chooses the variable that minimizes the number of unsatisfied clauses after flipping. Modern examples of SLS solvers include Sparrow[30], ProbSAT[31], CCAnr[32], YalSAT[33]. In addition to pure SLS solvers, there has been an increasing interest in combining CDCL and SLS solvers to create a hybrid solver. A typical example of this hybridization is [34]. Moreover, Section 3.3 discusses how to create new heuristics for SLS solvers with machine learning techniques.

2.3 SAT benchmarks

The international SAT competition (SATCOMP)[1] is held annually to encourage new SAT solving techniques and benchmarks. Solvers submitted to the main track are evaluated on hundreds of instances within 5 000s. Participants are required to contribute new benchmark instances, ranging from industrial instances to combinatorial problems. The SATCOMP benchmarks have been widely used in the literature for experimental evaluation.

SATLIB[2] consists of a benchmark suite of SAT instances and a collection of SAT solvers[35]. There are different types of instances in SATLIB, including uniform random 3-SAT, graph coloring, and planning instances.

2.4 Graph representation of Boolean formulae

SAT instances have been extensively analyzed in the form of graphs for their structural features and measures[36]. There are four straightforward graph representations of a CNF formula: 1) literal-clause graph (LCG), 2) literal-incidence graph (LIG), 3) variable-clause graph (VCG), and 4) variable-incidence graph (VIG). LCG is a bipartite graph with literals on one side and clauses on the other side, with edges connecting literals to the clauses where they occur. LIG consists only of literal nodes and two literals have an edge if they co-occur in a clause. VCG and VIG are defined similarly by merging the positive and negative literals of the same variables. An illustration of the above four graph representations is shown in Fig. 2. The decreasing complexity of the four graphs suggests an increasing level of information compression: one can recover the original CNF formula from an LCG without any information loss, but barely characterize the formula given a VIG. Therefore, LCG and LIG are preferred in practice.

2.5 Machine learning

Machine learning is a long-standing discipline that focuses on building parameterized systems and improving their performance through experience. Many machine learning algorithms are intended to approximate a function, and the learning problem is to improve the accuracy of that function[37]. Specifically, when the function output is discrete, the task is called classification. Otherwise, the task of predicting continuous values is called regression. For many learning algorithms, the information in input problems is transformed and condensed through feature extraction, either manually or by components in the learning system.

2.5.1 Major paradigms of machine learning

Machine learning can be further categorized according to the way of guidance from experience. When the learning process is guided by samples of known input-output pairs of the function, it is called supervised learning. This is the most widely used form of machine learning and various methods have been developed, including decision trees, decision forests, logistic regression, support vector machines, neural networks plus a number of other. Conversely, unsupervised learning refers to learning with only unlabelled data.

Another popular field of machine learning in recent years is reinforcement learning (RL), where an agent is trained to interact with a dynamic environment in an optimal way. Specifically, the goal of an agent is to maximize the rewards it receives from the environment. Reinforcement learning methods are popular in recent attempts of solving combinatorial optimization problems because reinforcement learning is suitable for the discreteness of these hard problems. Representative RL algorithms include actor-critic[38] and Q-learning[39].

Over the past decade, deep learning[40] has emerged as a growing area of machine learning due to the rapid development of computing power. One prominent feature of deep learning models is the exponential increase in the number of parameters. With the help of gradient-based optimization algorithms, deep learning leads to a performance leap in various tasks.

2.5.2 Machine learning models for SAT solving

This section introduces some machine learning models and techniques that are involved in this survey, as well as how they relate to SAT solving.

Multilayer perceptrons (MLPs)

A multilayer perceptron is a fully connected neural network with multiple layers. It is a basic machine learn-

[1] https://satcompetition.github.io/

[2] https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

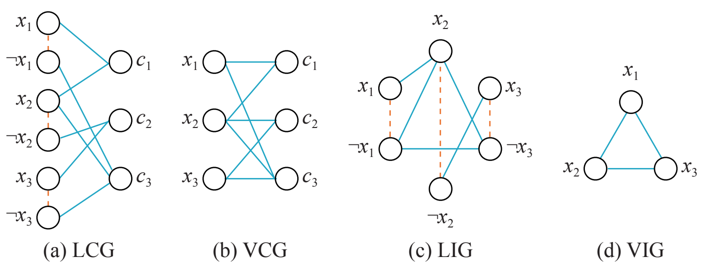


Fig. 2 Four graph representations of the propositional formula $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$. The dashed lines denote the common connections between complementary literals in GNNs for message passing.

ing model that can provide a nonlinear mapping from input to output. Usually, the parameters of MLPs are updated through an algorithm called backpropagation.

Graph neural networks (GNNs)

Graph neural networks are neural networks that are specialized for graph data. Inspired by convolutional neural networks, graph neural networks extend deep neural models to non-Euclidean domains[41]. The key design of graph neural networks is the way of message passing among nodes. Generally, each node aggregates information from its neighbors so it captures both node-level features and structure information. GNN models are closely related to ML-SAT solvers as introduced in Section 2.4. Graph convolutional networks (GCNs) are a type of GNNs first introduced in [42]. A GCN layer defines a first-order approximation of a localized spectral filter on graphs.

GNNs have been popular for modeling and solving graph tasks from direct node/graph-level classification[42] and edge prediction[43] to combinatorial tasks on graphs[44]. GNNs have also been recently applied in electronic design automation[45] especially for placement[46] and routing[47]. They are extensively used in ML-based SAT solving due to the natural graph representations of SAT instances.

Recurrent neural networks (RNNs)

Recurrent neural networks are models with recurrent connections and previously input data can affect subsequent output. Thus, they can model input and/or output consisting of sequences of elements that are not independent[48]. Therefore, RNNs are widely applicable to time series data, machine translation, and music generation. Many combinatorial optimization tasks can be formulated as sequential problems and solved by RNNs[49].

Generative adversarial networks (GANs)

Generative adversarial networks are deep generative models that can be trained with little or no supervision. As first proposed in [50], a GAN consists of a generator and a discriminator, where the generator is trained to produce synthetic data, and the task of the discriminator is to distinguish between real samples and synthetic ones. Ideally, the generator learns to generate data with the same distribution of real samples and "fool" the discriminator. GAN has been used for SAT instance generation as discussed in Section 4.

3 Towards machine learning of SAT solving

In this section, we first discuss standalone ML-SAT solvers, which are built in a pure machine learning framework. We then review the ML components in CDCL and SLS solvers for performance boosts.

3.1 Standalone ML-SAT solvers

If we treat the SAT problem as a classification task, many machine learning models can serve as the classifier as long as we first extract features from input formulae, which was tried over a decade ago. Deep learning, on the other hand, changes the way of feature extraction and facilitates end-to-end frameworks to predict satisfiability. This section starts with the classifiers with a handcrafted feature set, followed by deep learning models for feature extraction and end-to-end learning.

3.1.1 Classifiers with handcrafted features

The successful portfolio SAT solver SATzilla[51] constructed a feature set and used ridge regression to fit a runtime prediction function for further algorithm selection. This feature set was manually designed to describe the property of an instance, and it consists of 48 features in 9 categories: problem size features; variable-clause graph features; variable graph features; balance features; proximity to Horn formula; DPLL probing features; and local search probing features. Despite the limitations of human intervention, primitive statistical methods were inspired to utilize this feature set along with basic machine learning models (e.g., MLP, decision tree, naive Bayes, etc.) to classify SAT instances into binary categories for satisfiability prediction[52–55], as summarized in Table 1. Devlin and O'Sullivan[52] trained and evaluated a variety of machine learning models on crafted, industrial, and random instances from the SAT competition and SATLIB, achieving accuracy above 90% on most benchmarks. Xu et al.[53] focused on 3-SAT instances at the phase transition and used decision forests with 61 features to predict satisfiability. Danisovszky et al.[54] built another 48-dimensional feature set with an emphasis on special problems and clause properties. The authors experimented with different structures of neural networks as well as basic machine learning classifiers and achieved the

Table 1 Research on ML-SAT classifiers with handcrafted features

Reference	ML methods	Features	Instance type	Benchmark/Generator
[52]	Random forest Best-first decision tree MLP 1-nearest neighbor with generation Naive Bayes	SATzilla[51]	Crafted, industrial, random 3-SAT, random	SATCOMP, Miroslav Velev's SAT benchmark suite◇[56], SATLIB
[53]	Decision forest Decision tree	SATzilla2009[57]	Random 3-SAT	[58]
[54]	MLP Naive Bayes Support vector machine Decision tree Random forest	CNFStats†	–	SATLIB
[55]	Linear classifier Dense neural network Decision tree Random forest Gaussian Naive Bayes Bernoulli Naive Bayes Multinomial Naive Bayes	SATzilla[51]	Random 3-SAT	SATLIB

†http://fmv.ektf.hu/files/CnfStats.java
◇http://www.miroslav-velev.com/sat benchmarks.html

best result of approximately 99% accuracy. Atkari et al.[55] selected 23 features, including generic features and graph properties, to predict the satisfiability of random 3-SAT instances from SATLIB.

Although this line of work achieved high accuracy on different benchmarks, a major drawback occurs in the feature extraction part. The SATzilla-style feature set includes DPLL and local-search probing features, and thus an instance is tried to be solved by different solvers before the classification. Moreover, the time for feature extraction of [52] can be as high as thousands of seconds for one instance, comparable to the runtime of a complete solving routine. It is also a concern that handcrafted features require expert knowledge and considerable overhead for trial and error to devise effective features.

Deep learning, on the other hand, offers various approaches to automatically extract valuable features. Due to the powerful fitting ability of these models, they are capable of mining complex and underlying features that can hardly be recognized by humans. The dimension of the feature set is also controllable and flexible so that information is compressed to match computing resources even when they are limited. Section 3.1.2 discusses various deep learning models for feature extraction and SAT solving.

3.1.2 End-to-end neural SAT solvers

As introduced in Section 2.4, CNF formulae can be easily expressed in the form of graphs, so it is natural to apply GNNs to graph representations of SAT instances.

Bünz and Lamm[59] made an early attempt from the aspect of natural language processing: It treated the CNF formulae as sentences in natural language with RNNs, but it led to failure. In another trial, it used LIG representation for GNN and one-hot edge features to differentiate clauses apart. Since the difficulty of random 3-SAT instances is sensitive to the clause-to-atom ratio, it was tested in three settings and achieved an accuracy of approximately 65%, quite above a random baseline and indicating a promising direction.

The seminal work NeuroSAT[14] (shown in Fig. 3) improved the above results and presented an end-to-end framework to predict satisfiability on random instances by a message passing neural network (MPNN). Different from sentences in natural language, Boolean formulae have unique properties of permutation invariance and negation invariance[3]. This feature was preserved in NeuroSAT by symmetric edge connection and message passing. Specifically, the CNF formulae are encoded as LCGs, and node embedding is iteratively updated in a two-stage fashion for clauses and literals. First, each clause updates the embedding by receiving messages from neighboring literals. Next, each literal receives messages from neighboring clauses and the complementary literal. At the final layer, a scalar vote is computed for each literal that represents its confidence in predicting the formula to be satisfiable, and the mean vote value decides the final output. The network parameters are updated by the cross-entropy loss of the label of instance and model output using the ADAM optimizer[60]. Algorithm 3 demonstrates the training process of NeuroSAT. To train and evaluate NeuroSAT with enough samples, Selsam et al.[14] proposed a distribution **SR**(n) $(10 \leq n \leq 40)$, which consists of pairs of random SAT instances on n variables such that one element of the pair is satisfiable, the other

[3]The satisfiability of a formula is not affected by permuting the variables, the clauses or the literals within a clause. It is also not affected by negating every literal corresponding to a given variable[14].

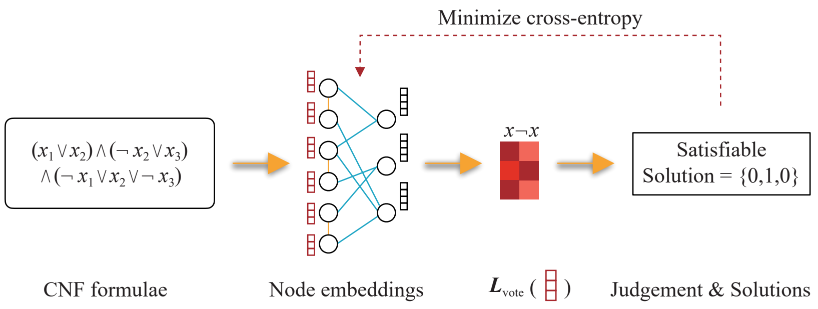


Fig. 3 Pipeline of the end-to-end SAT solver NeuroSAT[14]. Input CNF formulae are transformed into LCGs, and each literal and clause updates their embeddings from neighbors and complements. $\boldsymbol{L}_{\text{vote}}$ computes a vote scalar for each literal, which are aggregated to yield the final classification. Darker red of literals indicates a higher score in voting satisfiable, and the solution is yielded from clustering results of literal embeddings.

is unsatisfiable, and they differ by negating one literal occurrence in a single clause. Although the networks were trained in a supervised way only with the label of satisfiability, NeuroSAT attempted to yield a solution for instances with positive prediction. On **SR**(40), NeuroSAT reached an accuracy of 85% and solved 70% of SAT instances.

Algorithm 3. Training process of NeuroSAT[14]

Model parameters: Vector ($\boldsymbol{L}_{\text{init}}$, $\boldsymbol{C}_{\text{init}}$), MLP($\boldsymbol{L}_{\text{msg}}$, $\boldsymbol{C}_{\text{msg}}$, $\boldsymbol{L}_{\text{vote}}$), LSTM($\boldsymbol{L}_{\text{u}}, \boldsymbol{C}_{\text{u}}$)

```
for (instance $\varphi$, label $\phi$ ) in training dateset do
    $G \leftarrow$ LCG of $\varphi$;
    $M \in \mathbf{Z}_2^{2n\times m} \leftarrow$ adjacency matrix of $G$;
    $C^{(0)} \leftarrow \text{Tile}(\boldsymbol{C}_{\text{init}}, m)$;   /* clause node embeddings */
    $L^{(0)} \leftarrow \text{Tile}(\boldsymbol{L}_{\text{init}}, 2n)$;   /* literal node embeddings */
    $L_h^{(t)} \leftarrow \mathbf{0}, C_h^{(t)} \leftarrow \mathbf{0}$;   /* hidden states */
    for $t \leftarrow 0$ to $T-1$ do
        $(C^{(t+1)}, C_h^{(t+1)}) \leftarrow \mathbf{C}_{\text{u}}([C_h^{(t)}, M^{\text{T}}\boldsymbol{L}_{\text{msg}}(L^{(t)})])$;
        $(L^{(t+1)}, L_h^{(t+1)}) \leftarrow \boldsymbol{L}_{\text{u}}([L_h^{(t)}, \text{Flip}(L^{(t)}), M\boldsymbol{C}_{\text{msg}}$
        $(C^{(t+1)})])$;
    $L_*^{(T)} \leftarrow \boldsymbol{L}_{\text{vote}}(L^{(T)})$;
    $y^{(T)} \leftarrow \text{mean}(L_*^{(T)})$;
    $loss = \text{cross-entropy}(y^{(T)}, \phi)$;
    $(\boldsymbol{L}_{\text{init}}, \boldsymbol{C}_{\text{init}}, \boldsymbol{L}_{\text{msg}}, \boldsymbol{C}_{\text{msg}}, \boldsymbol{L}_{\text{vote}}, \boldsymbol{L}_{\text{u}}, \boldsymbol{C}_{\text{u}}) \leftarrow \text{ADAM}(loss)$;
```

Another end-to-end model that predicts satisfiability is [61]. To better capture the invariances of SAT instances (as defined in [14]), it proposed encoding CNF as permutation-invariant sparse matrices and used an exchangeable matrix architecture[62] for satisfiability prediction. Different from most GNN-based models, a SAT instance with n clauses and m variable is represented as a sparse binary tensor of shape $n \times m \times 2$. The input is then mapped to a d-dim embedding by a series of exchangeable matrix layers. Finally, the embedding is pooled to produce a scalar as the satisfiability prediction of the instance, and the network is supervised similarly to [14]. In the experiments, Cameron et al.[61] paid special attention to the generality across varying problem sizes of random 3-SAT as studied in [53]. After about 40 hours of training time, the exchangeable network managed to reach an accuracy above 80% for random SAT instances with 600 variables.

Despite NeuroSAT's impressive performance on random instances, its training paradigm bears limitations. First, it demands millions of training samples, which is inefficient and inconsistent with small instances it can solve (typically at most 40 variables). Second, satisfiability of instances is required beforehand and must be computed by other solvers, compromising the meaning of training a new solver for elementary instances. Hence, subsequent works prefer an unsupervised way and challenge more complex benchmarks.

QuerySAT[63] developed a recurrent neural SAT solver that was trained in an unsupervised fashion. By relaxing the variables to continuous values $[0, 1]$, the unsupervised loss $\mathcal{L}_\phi(x)$ for a formula ϕ is defined as

$$V_c(x) = 1 - \prod_{i\in c^+}(1-x_i)\prod_{i\in c^-}x_i,\ \ \mathcal{L}_\phi(x) = \prod_{c\in\phi}V_c(x) \quad (1)$$

where x_i is the value of the i-th variable and c^+ gives the set of variables that occur in clause c in the positive form and c^- in the negated form. The authors proved that this loss function is sufficient to uniquely identify the SAT formula ϕ. Different from [14], this loss is not only used at the final layer but also calculated for each query: at every time step, QuerySAT produces a query and evaluates a loss along with its gradient w.r.t. the query, which are then used for updating state vectors. The model was optimized to minimize the sum of all losses and generate a variable assignment. For empirical validation, QuerySAT used multiple benchmarks including k-SAT, 3-SAT, and other combinatorial problems, achieving an accuracy of over 90%.

In a similar vein, DG-DAGRNN[64] concentrated on the circuit satisfiability problem (Circuit-SAT), a special form of SAT, by unsupervised learning. This study proposed a neural Circuit-SAT solver which can harness structural information of the input circuits. The framework consists of a neural functional $\mathcal{F}_\theta$ which contains an embedding function, an aggregation function, and a classification function. To implement a fully differentiable training strategy, they proposed an explore-exploit mech-

Table 2 Summary of reviewed SAT solvers built with neural networks

Reference	Networks	Learning	Solver type	Instance type	Benchmark/Generator
[59]	GNN	Supervised	Standalone	Random 3-SAT	–
NeuroSAT[14]	GNN & LSTM	Supervised	Standalone	$\mathbf{SR}(n)$	–
[61]	Exchangeable networks	Supervised	Standalone	Random 3-SAT	[89]
QuerySAT[63]	GNN & Recurrent	Unsupervised	Standalone	$\mathbf{SR}(n)$, combinatorial	[14], SATRACE 2019
DG-DAGRNN[64]	DG-DAGRANN	Unsupervised	Standalone	$\mathbf{SR}(n)$, k-coloring	[14]
NeuroCore[65]	GNN	Supervised	CDCL	–	SATCOMP 2018
[66]	GNN & Attention	Supervised	DPLL & CDCL	$\mathbf{SR}(n)$	[14]
Graph-Q-SAT[67]	GNN	Reinforcement	CDCL	Random 3-SAT	SATLIB[35]
NeuroGlue[68]	GNN	Supervised & Reinforcement	CDCL	–	SATCOMP 2018, SATRACE 2019
GVE[69]	GNN	Reinforcement	CDCL	–	SATCOMP 2003-2019
NeuroCuber[70]	GNN	Supervised	Cube-and-conquer	Combinatorial	–
NeuroComb[71]	GNN	Supervised	CDCL	–	SATCOMP 2020
[78]	GNN	Reinforcement	SLS	Random 3-SAT, combinatorial	–
NLocalSAT[81]	GGCN	Supervised	SLS	Random	SATCOMP 2018

anism as in reinforcement learning. Specifically, they used the smooth min and max functions instead of hard ones in min-max circuits to allow the gradients to flow through all paths in the input circuit. Finally, they defined a satisfiability function to check if the resulting assignment satisfies the circuit. Following the settings in [14], DG-DAGRANN could converge faster than NeuroSAT, and the performance of NeuroSAT declines faster than DG-DAGRNN as the number of variables increases. In the graph k-coloring decision problem, DG-DAGRNN could solve 48% and 27% of the SAT instances in two generated datasets, respectively, while NeuroSAT failed to solve any of them, even when the number of iterations is large enough (e.g., 128 propagation iterations).

3.2 Learning-aided CDCL solvers

The full-stack SAT solvers with machine learning discussed in Section 3.1 are more of methodological than practical interests: They are trained with millions of samples and tested only on small random or combinatorial benchmarks with no guarantee of correctness, and thus fail to function for industrial purposes. Therefore, a more pragmatic way to boost solving large industrial instances is to modify existing CDCL solvers and replace the bottleneck components with machine learning modules. In practice, there are few suitable candidates for such a modification if we take into account the considerable computation time for neural networks. The most popular direction is the branching heuristics[65–71], plus some works on optimizing initialization[72], restart policy[73], and clause deletion[74].

3.2.1 Branching heuristics

Multi-armed bandit (MAB) is a classic model in reinforcement learning, which models an online learning scheme. MapleCOMSPS[13] proposed the learning rate branching (LRB) heuristic, with the core concept of learning rate to measure how fast each variable learns clauses. The branched variable is chosen using the exponential recency weighted average algorithm for MAB, where the reward for each variable is set to its learning rate. Kissat_MAB[11, 12] leverages MAB in a different fashion: The solver switches between two basic branching heuristics (variable state independent decaying sum (VSIDS) and conflict history-based branching heuristic (CHB)[75]) using the upper confidence bound (UCB) policy, with a reward function to estimate the quality of conflicts. Both solvers ranked top in the SAT competition and inspired subsequent works.

More academic attention is given to the deep learning paradigm. After the standalone neural network SAT solver NeuroSAT, the follow-up work, a more economical model NeuroCore[65], proposed to incorporate NeuroSAT into Minisat, a CDCL solver that implements the EVSIDS heuristic (a variant of VSIDS) and keeps an activity score for each variable. NeuroCore integrated the satisfiability prediction in NeuroSAT by periodically replacing the activity scores with the output from neural networks, termed periodic refocusing. The model of

NeuroCore is made up of three MLPs, one for updating the clause embedding based on the literals in it, one for updating the literal embedding based on the clauses it is in, and one for computing the scores for each variable as the output of NeuroCore. Different from the original NeuroSAT, the networks were trained with a focus on the unsatisfiable core. The logic behind this is that variables in the unsatisfiable core are prone to lead to conflicts, and thus are valuable for branching. Selgam and Bjorner[65] generated a dataset mapping unsatisfiable problems to the variables which are in the unsatisfiable cores. The hybrid solver neuro-minisat solved 10% more problems than Minisat on SATCOMP-2018 within the standard timeout of 5 000s, and a similar improvement was observed on Glucose.

Other works inspired by the NeuroSAT framework combine the GNN module with CDCL solvers to determine the variable to branch on. For example, Jaszczur et al.[66] used a similar network architecture as NeuroSAT and predicted satisfiability for each literal as well as the whole formula. Instead of two-stage update iterations, the node embeddings for clauses and variables were updated simultaneously by aggregating received messages. The authors explored two ways of aggregation: a simple average function and a modified attention mechanism. In the latter, the receiver node accepts or rejects each message according to the key and query vectors.

Graph-Q-SAT[67] utilized reinforcement learning instead of supervised learning for label efficiency. In the RL setting, each state contains unassigned variables and unsatisfied clauses containing these variables. At each step, the GNN-based agent chooses the next variable and the value to assign. This is implemented by formulating CNF formulae as VCGs and indicating the polarity of literals by edge vectors. After multiple iterations, each variable node produces two scores, representing the value of assigning it as true or false. A straightforward policy is taken by selecting the truth value corresponding to the maximal score. For evaluation, Graph-Q-SAT was compared to Minisat using the VSIDS heuristic, and the RL method used fewer steps to solve random 3-SAT instances.

Besides direct supervision of satisfiability and crafted unsupervised loss, another approach is to use statistics produced by solvers as supervision, such as the LBD and glue variables (those that are likely to occur in glue clauses). NeuroGlue[68] trained a neural network to predict the glue variables. The authors followed NeuroCore[65] and applied the periodic refocusing technique on the state-of-the-art SAT solver, CaDiCaL[76] by replacing the exponential variable state independent decaying sum (EVSIDS) activity scores with network outputs. The training data were generated by running CaDiCaL and counting the number of times each variable appears in glue clauses, used as supervision for glue variable prediction. There is also a reinforcement learning module that selects variables sequentially in an episode. The reward favors small glue levels. Glue variables elimination (GVE)[69] used two separate modules to determine branching variables and their values. There is a GNN-based glue variable selector by RL similar to [68] and another long short-term memory (LSTM) module that predicts the value of variables. Finally, the simplified CNF formulae are sent to a deterministic solver. Both NeuroGlue and GVE were tested on industrial benchmarks. NeuroGlue improves the PAR-2 score of CaDiCaL, while the complex architecture of GVE significantly increases running time.

In addition to the conflict-driven pattern, there is also a variable selection heuristic in the cube-and-conquer paradigm[77]. This technique aims to reduce the complexity of the SAT solver by partitioning a SAT instance into subproblems (cubes), which are then solved (conquered) by CDCL solvers in parallel, and there is a variable selection heuristic for cubing. Each selection adds two new leaves to the search tree that correspond to different assignments of the variable. Then the cutoff heuristic is used to check the new formulae and freeze some leaves if they are easy for CDCL. NeuroCuber[70] applied the network architectures of NeuroCore to the cube-and-conquer framework with an emphasis on delete resolution asymmetric tautologies (DRAT) proof occurrence counts. Besides the variable scoring head and the clause scoring head in NeuroSAT, NeuroCuber used another variable scoring head to predict occurrence counts of variables in DRAT proofs, which can be roughly thought of as a compressed representation of resolution trees. In [70], it was assumed that if a variable occurred frequently in a resolution tree, branching on it would minimize the average size of the resolution trees (and proportionally the solving times) for the leaves. Through experiments on datasets of unsatisfiable problems, they showed that models trained to predict DRAT variable counts usually outperform those trained to predict the occurrence of a variable in an unsatisfiable core.

Most of the above works fall short either in terms of applicability to industrial problems or computational efficiency. NeuroComb[71] proposed embedding GNN prediction into CDCL solvers in a more balanced way. To reduce the cost of periodic refocusing in NeuroCore, it adopted offline predictions computed before launching the CDCL solver. During the searching process, the dynamic branching heuristic (e.g., VSIDS) is periodically interrupted by this static information for a short time, so that the heuristic is under a constant but slow influence of GNN predictions.

3.2.2 Variable initialization

In search-based CDCL algorithms, the variables branched on are assigned a binary value of True or False based on some initialization scheme. The most basic initialization is by random. The work[72] assumed that an initial value close to solutions could provide considerable speedup for solving the problem, and it proposed to train a logistic regression module to predict the satisfiability of 3-SAT formulae with 10 predefined features as input. The preferred initialization value for each variable is determined by a series of Monte Carlo trials with satisfiability prediction from the trained predictor. The author reported a decrease of 23% in runtime for satisfiable instances if preprocessing time is not considered, which even outweighs the decrease in runtime. The essence of this method is very similar to classifiers in Section 3.1.1, and the logistic regression predictor can be replaced by peer methods.

3.2.3 Restart policy

Restarts are not only useful for SLS solvers but also effective for CDCL solvers. During the search process, a restart occurs when a certain number of conflicts are met and the solver discards the current partial assignment but keeps the learned clauses and searches from the start again. Liang et al.[73] designed a new restart policy called machine learning-based restart (MLR) that triggers a restart when the predicted LBD of the next learned clause is above a certain threshold. The MLB heuristic uses the LBDs of the last three learned clauses and their products as features to fit a linear function that predicts the LBD of the next learned clause. The MLR restart policy performs better than Luby but worse than Glucose, as demonstrated on SATCOMP benchmarks.

3.2.4 Clause deletion

Another entry point where learning can aid the CDCL solver is the clause deletion heuristic, which stands for the selection of useless clauses to be deleted learned from conflicts due to memory constraints. Vaezipoor et al.[74] formulated this task as a reinforcement learning problem and implemented an OpenAI Gym-compatible environment, SAT-Gym. Since the ultimate goal is to reduce the running time of the SAT solver, the reward is related to the number of logical operations performed by the solver until an instance is solved. Similar to restarting, the clause deletion heuristic also relies on the LBD metric to evaluate clause quality. Vaezipoor et al.[74] optimized a policy that outputs an LBD threshold as action by policy gradient so that all clauses with LBD values above the threshold are deleted.

3.3 Learning-aided SLS solvers

Due to the straightforward framework of SLS solvers, the available heuristics for machine learning extension are fewer and simpler than CDCL solvers. For example, the variable selection does not need to predict the value of the variable, since the only operation is flipping. The following works focus on variable selection and initialization in SLS solvers.

3.3.1 Variable selection

The work[78] proposed a variable selection heuristic for SLS solvers, which is computed by a graph neural network through deep reinforcement learning with a curriculum. The policy network, a GNN, takes as input a CNF formula in VCG form along with the current assignment and outputs a probability over variables, corresponding to their chances to be flipped in the next iteration. From the aspect of reinforcement learning, the reward is defined as whether the assignment satisfies the formula. The authors employed the REINFORCE algorithm[79] to optimize the policy network. For faster convergence, they opted for curriculum learning[80] and gradually increased the problem size. The empirical results of the learned heuristics are comparable to WalkSAT on small combinatorial instances, but it suffers considerable overhead since variable selection is required in every iteration.

3.3.2 Variable initialization

A possible solution to decrease computational cost is using off-line training and focusing on less frequent operations, such as initialization, which only occurs after restarts. NLocalSAT[81] followed this direction and boosted the performance of SLS solvers by guiding initialization assignments with a neural network. NLocalSAT feeds the CNF formula in LCG form into a gated graph convolutional network (GGCN) for feature extraction, whose output is a predicted solution. The actual initialization process accepts the prediction for a high probability and preserves the ability for exploration. Compared to [78], the neural network is called only once for each instance. Within a given time limit, NLocalSAT can solve more instances on multiple benchmarks, and this modification proves to be useful for various solvers.

4 Machine learning for SAT instance generation

In addition to solving the SAT problem itself, there have been some attempts to generate SAT instances with the help of machine learning techniques, especially in industrial scenarios. Different from random instances which can be easily generated according to explicit rules, industrial instances encode real-world problems and follow certain structural characteristics. The task of pseudo-industrial instance generation was nominated as one of the ten challenges in propositional reasoning and search early in [82].

An intriguing fact about real-world instances is that many of them can be effectively yet unexplainably solved by CDCL solvers. Meanwhile, SAT instances in certain scenarios (e.g., logic equivalence checking) tend to be unsolvable for the same type of solvers. Therefore, researchers have defined and studied various measures[36] to distinguish these instances, hoping to unravel why or why not solvers work on them. This understanding of instances conversely promotes new heuristics in SAT solvers.

Another motivation for SAT instance generation derives from the training process of machine learning, which requires a large number of instances while keeping a balance between classes in the dataset. Therefore, an effective tool for instance generation helps with data augmentation and improves training.

Non-ML research on pseudo-industrial SAT instance generation is mainly based on structural measures and probabilistic methods. For example Giráldez-Cru and Levg[83] proposed the community attachment (CA) model to generate new instances of given modularity Giráldez-Cru and Levg[84] leveraged the notions of locality and popularity to portray the scale-free structure of real-world instances in their popularity-similarity (PS) model.

Recently, deep generative models have been adopted for this task, including [85–87]. SATGEN[85] is a generative model trained in an unsupervised way. Different from previous models that parameterize certain graph measures, it uses a GAN to implicitly capture graph-based features of SAT instances. Specifically, real formulae are transformed into LIG as input to GAN for training. The learned model then generates new LIGs which are recovered to CNF formulae by a greedy hill-climbing approach. For evaluation, SATGEN was compared with the aforementioned CA and PS models for the statistics of generated graphs, e.g., clustering coefficients, modularities, and clause degrees. The results show that SATGEN has a satisfactory performance. An obvious disadvantage of SATGEN is that an LIG does not uniquely map to a SAT formula; thus, the model cannot differentiate between different instances with the same LIG representation. This information loss also leads to extra post-processing and imposed constraints in the recovering stage.

The subsequent work G2SAT[86] improves SATGEN and focuses on bipartite LCG representation. It proposed a novel node-merging (node-splitting) algorithm to generate bipartite graphs from (to) a forest. The key component of G2SAT is a GCN-based classifier that decides node pairs for merging. Specifically, G2SAT adopted GraphSAGE[88], where node embeddings are updated by previous embeddings of itself and neighbors. To construct the training dataset, the node splitting operation is applied to real-world SAT instances, and the inverse sequences of node splitting are used for supervision. G2SAT conducted a similar evaluation as SATGEN and achieved better results. Additionally, the authors attempted to develop better SAT solvers by tuning with synthetic instances, and G2SAT achieved a performance gain of 18%.

Garzón et al.[87] modified the G2SAT framework and explored different methods of graph representations and node embedding. Overall, there are three variants of G2SAT, namely GCN2S, EGNN2S, and ECC2S. First, GCN2S replaced GraphSAGE with graph convolution in [42] for node embedding. On top of GCN2S, the most distinguishing modification of [87] is that it takes edge features into consideration. Instead of using LCGs, EGNN2S and ECC2S encode CNF formulae as signed VCGs, so that the edges between variables and clauses indicate whether the literal is positive or negative. During message passing, node embeddings incorporate edge features based on dynamic edge filters. Garzón et al.[87] also provided an extensive experimental evaluation of G2SAT and three variants. With 41 instances used for training, no model consistently leads in the performance of community structure and the clustering coefficient statistics; all four models fail to reproduce these two features for nearly half of the instances. As for the satisfiability and hardness of synthetic instances, all models tend to produce a majority of unsatisfiable instances, and synthetic instances are much easier than input instances. Overall Garzón et al.[87] pointed out several limitations of current deep generative models, which are inspiring for future avenues.

5 Conclusion and outlook

In this survey, we review recent progress on solving the SAT problem with machine learning and generating SAT instances with ML techniques, with an emphasis on deep learning models. We summarize and compare different works to provide an overview of this topic. As an emergent area of interest, the integration of SAT solving and machine learning has undergone rapid development. End-to-end SAT solvers have come to reality and been evaluated on random instances, even with better results on new instances from the SAT competition. Meanwhile, recent years have also witnessed a line of works on the combination of learning-aided heuristics in existing solvers, yielding apparent improvements in search efficiency and effectiveness.

Nevertheless, several challenges remain to be solved, and the following problems merit further investigation. The major concern of current ML-SAT solvers is that they require a considerable amount of time and samples for training, in contrast to the instance scale they manage to solve. In many cases, the substantial computational time may offset the performance gain. The intrinsic explainability problem of neural networks poses another question of how people should trust the predictions of an ML-aided SAT solver when it does not provide precise proofs.

Currently, there is no consistent benchmark and evaluation pipeline for ML-based SAT solvers, as summarized in Tables 1 and 2. Although many papers compare their models with peer methods, these results cannot be gathered and compared directly due to the difference in training datasets: the sources of training datasets and the number of samples vary. The biggest reason for this inconsistency is the inherent feature that ML models depend on enough data for learning. Therefore, constructing a universal ML-friendly benchmark is meaningful but also requires much effort. There have also been some efforts in the instance generation problem, as discussed in Section 4, inviting future solutions to this problem.

Acknowledgements

This work was supported by National Key Research and Development Program of China (No. 2020AAA0107600), National Science Foundation of China (No. 62102258), Shanghai Pujiang Program, China (No. 21PJ1407300), Shanghai Municipal Science and Technology Major Project, China (No. 2021SHZDZX0102), Science and Technology Commission of Shanghai Municipality Project, China (No. 22511105100), and also sponsored by Huawei Ltd, China.

Declarations of conflict of interest

The authors declared that they have no conflicts of interest to this work.

References

[1] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, Shaker Heights, USA, pp. 151–158, 1971. DOI: 10.1145/800157.805047.

[2] K. Iwama, S. Miyazaki. SAT-variable complexity of hard combinatorial problems. In *Proceedings of IFIP Transactions A: Computer Science and Technology*, vol. 51, pp. 253–258, 1994.

[3] M. N. Velev. Exploiting hierarchy and structure to efficiently solve graph coloring as SAT. In *Proceedings of IEEE/ACM International Conference on Computer-aided Design*, IEEE, San Jose, USA, pp. 135–142, 2007. DOI: 10.1109/ICCAD.2007.4397256.

[4] R. Plachetta, A. Van Der Grinten. SAT-and-Reduce for vertex cover: Accelerating branch-and-reduce by SAT solving. In *Proceedings of Symposium on Algorithm Engineering and Experiments*, Philadelphia, USA, pp. 169–180, 2021. DOI: 10.1137/1.9781611976472.13.

[5] S. Skansi, K. Šekrst, M. Kardum. A different approach for clique and household analysis in synthetic telecom data using propositional logic. In *Proceedings of the 43rd International Convention on Information, Communication and Electronic Technology*, IEEE, Opatija, Croatia, pp. 1286–1289, 2020. DOI: 10.23919/MIPRO48935.2020.9245421.

[6] J. Brakensiek, M. Heule, J. Mackey, D. Narváez. The resolution of Keller's conjecture. In *Proceedings of the 10th International Joint Conference on Automated Reasoning*, Springer, Paris, France, pp. 48–65, 2020. DOI: 10.1007/978-3-030-51074-9_4.

[7] H. T. Zhang, J. H. R. Jiang, A. Mishchenko. A circuit-based sat solver for logic synthesis. In *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, IEEE, Munich, Germany, 2021. DOI: 10.1109/ICCAD51958.2021.9643505.

[8] Y. Bengio, A. Lodi, A. Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021. DOI: 10.1016/j.ejor.2020.07.063.

[9] J. C. Yan, S. Yang, E. Hancock. Learning for graph matching and related combinatorial optimization problems. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, ACM, Yokohama, Japan, Article number 694, 2021. DOI: 10.5555/3491440.3492134.

[10] J. Y. Zhang, C. Liu, X. J. Li, H. L. Zhen, M. X. Yuan, Y. W. Li, J. C. Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, vol. 519, pp. 205–217, 2023. DOI: 10.1016/j.neucom.2022.11.024.

[11] M. S. Cherif, D. Habet, C. Terrioux. Combining VSIDS and CHB using restarts in SAT. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming*, Dagstuhl, Germany, vol. 210, Article number 20, 2021. DOI: 10.4230/LIPIcs.CP.2021.20.

[12] M. S. Cherif, D. Habet, C. Terrioux. Kissat MAB: Combining VSIDS and CHB through multi-armed bandit. In *Proceedings of SAT Competition: Solver and Benchmark Descriptions*, University of Helsinki, Helsinki, Finland, pp. 15–16, 2021.

[13] J. H. Liang, C. Oh, V. Ganesh, K. Czarnecki, P. Poupart. MapleCOMSPS, MapleCOMSPS LRB, MapleCOMSPS CHB. In *Proceedings of SAT Competition: Solver and Benchmark Descriptions*, University of Helsinki, Helsinki, Finland, pp. 52–53, 2016.

[14] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. De Moura, D. L. Dill. Learning a SAT solver from single-bit supervision. In *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, USA, 2019.

[15] A. Popescu, S. Polat-Erdeniz, A. Felfernig, M. Uta, M. Atas, V. M. Le, K. Pilsl, M. Enzelsberger, T. N. T. Tran. An overview of machine learning techniques in constraint solving. *Journal of Intelligent Information Systems*, vol. 58, no. 1, pp. 91–118, 2022. DOI: 10.1007/s10844-021-00666-5.

[16] S. B. Holden. Machine learning for automated theorem proving: Learning to solve SAT and QSAT. *Foundations and Trends® in Machine Learning*, vol. 14, no. 6, pp. 807–989, 2021. DOI: 10.1561/2200000081.

[17] F. Hutter, L. Xu, H. H. Hoos, K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, vol. 206, pp. 79–111, 2014. DOI: 10.1016/j.artint.2013.10.003.

[18] M. J. H. Heule, A. Biere. Proofs for satisfiability problems. *All About Proofs, Proofs for All*, vol. 55, no. 1, pp. 1–22, 2015.

[19] A. Biere, M. Heule, H. Van Maaren, T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, Amsterdam, The Netherlands: IOS Press, 2009.

[20] J. P. Marques-Silva, K. A. Sakallah. GRASP: A search al-

gorithm for propositional satisfiability. *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999. DOI: 10.1109/12.769433.

[21] M. Davis, G. Logemann, D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962. DOI: 10.1145/368273.368557.

[22] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, IEEE, Las Vegas, USA, pp. 530–535, 2001. DOI: 10.1145/378239.379017.

[23] A. Biere, A. Fröhlich. Evaluating CDCL variable scoring schemes. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Austin, USA, pp. 405–422, 2015. DOI: 10.1007/978-3-319-24318-4_29.

[24] A. Biere. Adaptive restart strategies for conflict driven SAT solvers. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Guangzhou, China, pp. 28–33, 2008. DOI: 10.1007/978-3-540-79719-7_4.

[25] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Bordeaux, France, pp. 123–140, 2016. DOI: 10.1007/978-3-319-40970-2_9.

[26] F. Xiao, C. M. Li, M. Luo, F. Manyà, Z. Lü, Y. Li. A branching heuristic for SAT solvers based on complete implication graphs. *Science China Information Sciences*, vol. 62, no. 7, Article number 72103, 2019. DOI: 10.1007/s11432-017-9467-7.

[27] B. Selman, H. A. Kautz, B. Cohen. Local search strategies for satisfiability testing. In *Proceedings of a DIMACS Workshop on Cliques, Coloring, and Satisfiability*, New Brunswick, USA, pp. 521–532, 1993.

[28] G. Audemard, L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, ACM, Pasadena, USA, pp. 399–404, 2009. DOI: 10.5555/1661445.1661509.

[29] B. Selman, H. Levesque, D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, San Jose, USA, pp. 440–446, 1992. DOI: 10.5555/1867135.1867203.

[30] A. Balint, A. Fröhlich. Improving stochastic local search for SAT with a new probability distribution. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Edinburgh, UK, pp. 10–15, 2010. DOI: 10.1007/978-3-642-14186-7_3.

[31] A. Balint, U. Schöning. Choosing probability distributions for stochastic local search and the role of make versus break. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Trento, Italy, pp. 16–29, 2012. DOI: 10.1007/978-3-642-31612-8_3.

[32] S. W. Cai, C. Luo, K. L. Su. CCAnr: A configuration checking based local search solver for non-random satisfiability. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Austin, USA, pp. 1–8, 2015. DOI: 10.1007/978-3-319-24318-4_1.

[33] A. Biere. Splatz, lingeling, plingeling, treengeling, YalSAT entering the SAT competition. In *Proceedings of SAT Competition: Solver and Benchmark Descriptions*, Helsinki, Finland, pp. 44–45, 2016.

[34] S. W. Cai, X. D. Zhang. Deep cooperation of CDCL and local search for SAT. In *Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Barcelona, Spain, pp. 64–81, 2021. DOI: 10.1007/978-3-030-80223-3_6.

[35] H. H. Hoos, T. Stützle. SATLIB: An online resource for research on SAT. In *Proceedings of the Highlights of Satisfiability Research in the Year 2000*, Amsterdam, The Netherlands, pp. 283–292, 2000.

[36] T. N. Alyahya, M. El Bachir Menai, H. Mathkour. On the structure of the boolean satisfiability problem: A survey. *ACM Computing Surveys*, vol. 55, no. 3, Article number 46, 2023. DOI: 10.1145/3491210.

[37] M. I. Jordan, T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, vol. 349, no. 6245, pp. 255–260, 2015. DOI: 10.1126/science.aaa8415.

[38] B. Xi, R. Wang, Y. H. Cai, T. Lu, S. Wang. A novel heterogeneous actor-critic algorithm with recent emphasizing replay memory. *International Journal of Automation and Computing*, vol. 18, no. 4, pp. 619–631, 2021. DOI: 10.1007/s11633-021-1296-x.

[39] C. J. C. H. Watkins, P. Dayan. Q-learning. *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992. DOI: 10.1007/BF00992698.

[40] Y. LeCun, Y. Bengio, G. Hinton. Deep learning. *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[41] J. Zhou, G. Q. Cui, S. D. Hu, Z. Y. Zhang, C. Yang, Z. Y. Liu, L. F. Wang, C. C. Li, M. S. Sun. Graph neural networks: A review of methods and applications. *AI Open*, vol. 1, pp. 57–81, 2020. DOI: 10.1016/j.aiopen.2021.01.001.

[42] T. N. Kipf, M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France, [Online], Available: https://openreview.net/forum?id=HkwoSDPgg, 2017.

[43] M. H. Zhang, Y. X. Chen. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ACM, Montreal, Canada, pp. 5171–5181, 2018. DOI: 10.5555/3327345.3327423.

[44] H. J. Dai, E. B. Khalil, Y. Y. Zhang, B. Dilkina, L. Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ACM, Long Beach, USA, pp. 6351–6361, 2017. DOI: 10.5555/3295222.3295382.

[45] D. S. Lopera, L. Servadei, G. N. Kiprit, R. Wille, W. Ecker. A comprehensive survey on electronic design automation and graph neural networks: Theory and applications. *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 2, Article number 15, 2022. DOI: 10.1145/3543853.

[46] R. Y. Cheng, J. C. Yan. On joint learning for solving place-

ment and routing in chip design. In *Proceedings of the 34th Conference on Neural Information Processing Systems*, pp. 16508–16519, 2021.

[47] R. Y. Cheng, X. L. Lv, Y. Li, J. J. Ye, J. Y. Hao, J. C. Yan. The policy-gradient placement and generative routing neural networks for chip design. In *Proceedings of the 36th Conference on Neural Information Processing Systems*, 2022.

[48] Z. C. Lipton, J. Berkowitz, C. Elkan. A critical review of recurrent neural networks for sequence learning, [Online], Available: https://arxiv.org/abs/1506.00019, 2015.

[49] O. Vinyals, M. Fortunato, N. Jaitly. Pointer networks. In *Proceedings of Advances in Neural Information Processing Systems 28*, Montreal, Canada, pp. 2692–2700, 2015.

[50] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ACM, Montreal, Canada, vol. 2, pp. 2672–2680, 2014. DOI: 10.5555/2969033.2969125.

[51] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, vol. 32, pp. 565–606, 2008. DOI: 10.1613/jair.2490.

[52] D. Devlin, B. O′Sullivan. Satisfiability as a classification problem. In *Proceedings of Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[53] L. Xu, H. Hoos, K. Leyton-Brown. Predicting satisfiability at the phase transition. In *Proceedings of AAAI Conference on Artificial Intelligence*, Toronto, Canada vol. 26, pp. 584–590, 2021. DOI: 10.1609/aaai.v26i1.8142.

[54] M. Danisovszky, Z. G. Yang, G. Kusper. Classification of SAT problem instances by machine learning methods. In *Proceedings of the 11th International Conference on Applied Informatics*, Eger, Hungary, pp. 94–104, 2020.

[55] A. Atkari, N. Dhargalkar, H. Angne. Employing machine learning models to solve uniform random 3-SAT. In *Proceedings of GUCON 2019 Data Communication and Networks*, Springer, pp. 255–264, 2020. DOI: 10.1007/978-981-15-0132-6_17.

[56] M. N. Velev. Exploiting signal unobservability for efficient translation to CNF in formal verification of microprocessors. In *Proceedings of Conference on Design, Automation and Test in Europe*, IEEE, Paris, France, pp. 266–271, 2004. DOI: 10.1109/DATE.2004.1268859.

[57] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown. Satzilla 2009: An automatic algorithm portfolio for sat. *SAT*, vol. 4, pp. 53–55, 2009.

[58] L. Simon. 2002. [Online], Available: http://www.satcompetition.org/2003/TOOLBOX/genAlea.c.

[59] B. Bünz, M. Lamm. Graph neural networks and Boolean satisfiability. [Online], Available: https://arxiv.org/abs/1702.03592, 2017.

[60] D. P. Kingma, J. Ba. Adam: A method for stochastic optimization. [Online], Available: https://arxiv.org/abs/1412.6980, 2015.

[61] C. Cameron, R. Chen, J. Hartford, K. Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *Proceedings of Conference on Artificial Intelligence*, New York, USA, vol. 34, pp. 3324–3331, 2020. DOI: 10.1609/aaai.v34i04.5733.

[62] J. Hartford, D. Graham, K. Leyton-Brown, S. Ravanbakhsh. Deep models of interactions across sets. In *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, pp. 1909–1918, 2018.

[63] E. Ozolins, K. Freivalds, A. Draguns, E. Gaile, R. Zakovskis, S. Kozlovics. Goal-aware neural SAT solver. In *Proceedings of International Joint Conference on Neural Networks*, IEEE, Padua, Italy, 2022. DOI: 10.1109/IJCNN55064.2022.9892733.

[64] S. Amizadeh, S. Matusevych, M. Weimer. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, USA, 2019.

[65] D. Selsam, N. Bjorner. Guiding high-performance SAT solvers with unsat-core predictions. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing*, Springer, Lisbon, Portugal, pp. 336–353, 2019. DOI: 10.1007/978-3-030-24258-9_24.

[66] S. Jaszczur, M. Łuszczyk, H. Michalewski. Neural heuristics for SAT solving, [Online], Available: https://arxiv.org/abs/2005.13406, 2020.

[67] V. Kurin, S. Godil, S. Whiteson, B. Catanzaro. Can Q-learning with graph networks learn a generalizable branching heuristic for a SAT solver? In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ACM, Vancouver, Canada, Article number 806, 2020. DOI: 10.5555/3495724.3496530.

[68] J. M. Han. Enhancing SAT solvers with glue variable predictions, [Online], Available: https://arxiv.org/abs/2007.02559, 2020.

[69] Z. Zhang, Y. Zhang. Elimination mechanism of glue variables for solving SAT problems in linguistics. In *Proceedings of the Asian Conference on Language*, ACL, pp. 147–167, 2021. DOI: 10.22492/issn.2435-7030.2021.11.

[70] J. Han. Learning cubing heuristics for SAT from DRAT proofs. In *Proceedings of the 5th Conference on Artificial Intelligence and Theorem Proving*, Aussois, France, 2020.

[71] W. X. Wang, Y. Hu, M. Tiwari, S. Khurshid, K. McMillan, R. Miikkulainen. NeuroComb: Improving SAT solving with graph neural networks, [Online], Available: https://arxiv.org/abs/2110.14053, 2021.

[72] H. Z. Wu. Improving SAT-solving with machine learning. In *Proceedings of ACM SIGCSE Technical Symposium on Computer Science Education*, Seattle, USA, pp. 787–788, 2017. DOI: 10.1145/3017680.3022464.

[73] J. H. Liang, C. Oh, M. Mathew, C. Thomas, C. X. Li, V. Ganesh. Machine learning-based restart policy for CDCL SAT solvers. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing*, Springer, Oxford, UK, pp. 94–110, 2018. DOI: 10.1007/978-3-319-94144-8_6.

[74] P. Vaezipoor, G. Lederman, Y. H. Wu, R. Grosse, F. Bacchus. Learning clause deletion heuristics with reinforcement learning. In *Proceedings of the 5th Conference on Artificial Intelligence and Theorem Proving*, Aussois, France, 2020.

[75] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki. Exponential recency weighted average branching heuristic for

SAT solvers. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, Phoenix, USA, pp. 3434–3440, 2016. DOI: 10.5555/3016100.3016385.

[76] A. Biere. CaDiCaL, lingeling, plingeling, treengeling and YalSAT entering the SAT competition 2018. In *Proceedings of SAT Competition: Solver and Benchmark Descriptions*, SAT, University of Helsinki, Helsinki, Finland, pp. 13–14, 2018.

[77] M. J. H. Heule, O. Kullmann, V. W. Marek. Solving very hard problems: Cube-and-conquer, a hybrid SAT solving method. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 4864–4868, 2017. DOI: 10.24963/ijcai.2017/683.

[78] E. Yolcu, B. Póczos. Learning local search heuristics for Boolean satisfiability. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, ACM, Red Hook, USA, Article number 718, 2019. DOI: 10.5555/3454287.3455005.

[79] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992. DOI: 10.1007/BF00992696.

[80] Y. Bengio, J. Louradour, R. Collobert, J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ACM, Montreal, Canada, pp. 41–48, 2009. DOI: 10.1145/1553374.1553380.

[81] W. J. Zhang, Z. Y. Sun, Q. H. Zhu, G. Li, S. W. Cai, Y. F. Xiong, L. Zhang. NLocalSAT: Boosting local search with solution prediction. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pp. 1177–1183, 2020. DOI: 10.24963/ijcai.2020/164.

[82] B. Selman, H. Kautz, D. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, ACM, Nagoya, Japan, pp. 50–54, 1997. DOI: 10.5555/1624162.1624170.

[83] J. Giráldez-Cru, J. Levy. Generating SAT instances with community structure. *Artificial Intelligence*, vol. 238, pp. 119–134, 2016. DOI: 10.1016/j.artint.2016.06.001.

[84] J. Giráldez-Cru, J. Levy. Popularity-similarity random SAT formulas. *Artificial Intelligence*, vol. 299, Article number 103537, 2021. DOI: 10.1016/j.artint.2021.103537.

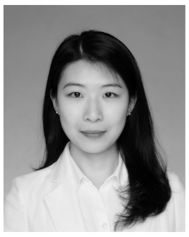
[85] H. Z. Wu, R. Ramanujan. Learning to generate industrial SAT instances. In *Proceedings of the 20th International Symposium on Combinatorial Search*, Napa, USA, 2019. DOI: 10.1609/socs.v10i1.18493.

[86] J. X. You, H. Z. Wu, C. Barrett, R. Ramanujan, J. Leskovec. G2SAT: Learning to generate SAT formulas. In *Proceedings of the 33rd Conference on Neural Information Processing Systems*, Vancouver, Canada, 2019.

[87] I. Garzón, P. Mesejo, J. Giráldez-Cru. On the performance of deep generative models of realistic SAT instances. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing*, Dagstuhl, Germany, Article number 3, 2022. DOI: 10.4230/LIPIcs.SAT.2022.3.

[88] W. L. Hamilton, R. Ying, J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ACM, Long Beach, USA, pp. 1025–1035, 2017. DOI: 10.5555/3294771.3294869.

[89] J. M. Crawford, L. D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, vol. 81, no. 1–2, pp. 31–57, 1996. DOI: 10.1016/0004-3702(95)00046-1.



Wenxuan Guo received the B.Sc. degree in computer science and technology from Shanghai Jiao Tong University, China in 2021. Currently, she is a Ph.D. degree candidate in computer science and technology at Department of Computer Science and Engineering, Shanghai Jiao Tong University, China.

Her research interests include machine learning and combinatorial optimization.

E-mail: arya_g@sjtu.edu.cn

ORCID iD: 0000-0001-6336-3819

Hui-Ling Zhen received the B.Sc. degree in numerical mathematics and the Ph.D. degree in applied mathematics from Beijing University of Posts and Telecommunications, China in 2011 and 2016, respectively. She was a post-doctoral research fellow in City University of Hong Kong, China from 2016 to 2019. Currently, she is a research scientist in Noah's Ark Laboratory, Huawei, China since 2019. She has published over 60 peer-reviewed papers in mainstream conferences and journals.

Her research interests include large-scale optimization, constraint programming, as well as their applications in supply chain management and chip design.

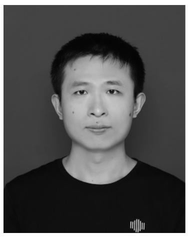
E-mail: zhenhuiling2@huawei.com

Xijun Li received M.Sc. degree in computer science from Shanghai Jiao Tong University, China in 2018. Currently, he is a senior researcher of Huawei Noah's Ark Laboratory, China, and also is a Ph.D. degree candidate in Electronic Engineering and information science at University of Science and Technology of China (HUAWEI-USTC Joint Ph.D. Program). He has published several papers on top peer-reviewed conferences and journals (SIGMOD, KDD, ICDE, DAC, CIKM, ICDCS, TCYB, etc.).

His research interests include learning to optimize combinatorial optimization problem and machine learning for computer systems.

E-mail: xijun.li@huawei.com (Corresponding author)

ORCID iD: 0000-0002-9013-1180



Wanqian Luo received the B.Sc. degree in numerical mathematics and the M.Eng. degree in software engineer from South China University of Technology, China in 2016 and 2019 respectively. After that, he is a research engineer of Huawei Noah's Ark Laboratory, China.

His research interests include applied formal methods, Boolean satisfiability problem, as well as the applications in chip design.

E-mail: luowanqian1@huawei.com

Mingxuan Yuan received the Ph. D. degree in computer science from Hong Kong University of Science and Technology, China in 2011. He is currently a principal researcher of Huawei Noah's Ark Laboratory, China.

His research interests include data-driven optimization algorithms, data-driven SAT/MIP solving algorithms and data-driven EDA algorithm.

E-mail: Yuan.Mingxuan@huawei.com

Yaohui Jin received the Ph. D. degree in electronic engineering from Shanghai Jiao Tong University, China in 2000. He is a tenured professor in School of Electronic Information and Electrical Engineering and Artificial Intelligence Institute, Shanghai Jiao Tong University, China. He was a member of technical staff at Bell Labs Research, China from 2000 to 2002.

His research interests include software defined infrastructure, spatial and temporal data mining as well as natural language understanding.

E-mail: jinyh@sjtu.edu.cn (Corresponding author)

ORCID iD: 0000-0001-6158-6277

Junchi Yan received the B. Eng. degree in automation from University of Science and Technology Beijing, China in 2008, and the M. Sc. degree in pattern recognition and intelligent systems, and the Ph.D. degree in information and communication engineering, both from Shanghai Jiao Tong University, China in 2011 and 2015, respectively. He is an associate professor with Department of Computer Science and Engineering, and AI Institute of Shanghai Jiao Tong University, China. Before that, he was a senior research staff member with IBM Research where he started his career since April 2011. He served as Area Chair for NeurIPS/ICML/CVPR/AAAI/ACM-MM and Senior PC for IJCAI/CIKM, and Associate Editor for *Pattern Recognition*.

His research interests include machine learning (especially for combinatorial optimization) and computer vision.

E-mail: yanjunchi@sjtu.edu.cn (Corresponding author)

ORCID iD: 0000-0001-9639-7679