# Towards Unsupervised End-to-End Learning for Partial Maximum Satisfiability Problems

Yuhan Ye* Cheng Chen† Qi Gao‡ Zaiwen Wen§

November 23, 2024

**Abstract**

Partial MaxSAT (PMS) is a practical extension of the MaxSAT problem, involving additional hard clauses that must be satisfied. Due to its NP-hardness, a significant obstacle lies in the unaffordable time consumption to obtain optimal solutions, resulting in the limited problem size of the training process when applying learning methods. This paper proposes U-SAT, a novel end-to-end learning framework that provides high-quality solutions for PMS problems. U-SAT employs a policy gradient method to train the model in an unsupervised manner, augmented by a filter function incorporated into the objective function to broaden the functional landscape. Moreover, we integrate unit propagation into the decoding process based on the structure of PMS. A weight adaptation scheme is proposed to facilitate further refinement to address challenging large-scale instances. Experimental results demonstrate that U-SAT achieves near-optimal solutions on synthetic datasets and achieves comparable performance against the state-of-the-art PMS solvers on the instances provided by the MaxSAT Evaluation (MSE) 2016.

## 1 Introduction

Combinatorial optimization (CO) plays a crucial role in computer science. The majority of challenges are caused by NP-hardness, rendering the pursuit of optimal solutions unfeasible for large-scale instances due to the exponentially growing time consumption. Consequently, heuristic approaches, founded on manually crafted rules, have been the prevailing methods that provide near-optimal solutions [1–3]. Recently, advancements in deep learning have introduced powerful learning models, leading to a growing interest in employing learning methods in the domain of CO [4, 5].

In this paper, we focus on the unsupervised learning method to address Partial MaxSAT (PMS), a practical extension of MaxSAT, in an end-to-end manner. PMS aims to maximize the number of satisfied soft clauses in a conjunctive normal formula (CNF) with the constraint that all the hard clauses must be satisfied. Although our main emphasis is on PMS problems, the algorithms discussed can be directly applied to address MaxSAT instances.

The previous works for learning methods on MaxSAT and logical reasoning can be divided into two categories. The first category focuses on integrating learning methods into traditional search frameworks to enhance the performance of heuristic components [6–14]. Additionally, other studies employ reinforcement learning methods as a hyper-heuristic framework for MaxSAT [15–17]. The second category involves the application of end-to-end models that take a CNF formula as input and directly provide a prediction of the optimal solutions [18–22]. This line of work is more similar to ours, which aims to train neural network models in a differentiable manner.

Current research on end-to-end framework mainly involves training a graph neural network (GNN) via supervised learning. This approach derives labels from optimal solutions obtained by existing complete solvers. While this framework demonstrates efficiency and generalization capabilities, it is constrained by the size of the problem instances within the training dataset because complete solvers struggle with large-scale instances.

*School of Mathematics, Peking University, Beijing, CHINA (2100010664@stu.pku.edu.cn), co-first author
†Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, CHINA(chen1999@pku.edu.cn), co-first author
‡Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, CHINA
§Beijing International Center for Mathematical Research, Peking University, Beijing, CHINA(wenzw@pku.edu.cn)

In contrast, solutions provided by existing incomplete solvers lack a quality guarantee and may exhibit sub-optimal performance in specific cases. Consequently, relying on incomplete solvers to generate solutions for supervised learning becomes unreliable.

Nevertheless, unsupervised learning proved to be a viable approach to overcome the obstacle of obtaining the optimal solution [23–30]. Note that a common method involves using a deep learning model that takes a representation of the CO problem instance as input and then outputs parameters of a solutions distribution, focusing on high-quality regions. This probabilistic approach to CO is used in recent works [24, 27, 29, 31]. A majority of them depend on the utilization of empirically selected loss functions, including the expectation of the objective function [24] and the cost function based on physics-inspired Hamiltonian [25].

Applying this framework to PMS poses challenges, primarily because formulating a PMS instance as an optimization problem results in complex objective functions and constraints, of which the expectation is challenging to be explicitly represented by the output distribution.

**Contributions**. In this paper, we address the aforementioned challenges by introducing a novel unsupervised learning framework, U-SAT, specifically tailored for solving PMS problems. We adopt a penalized objective function to overcome the constraints introduced by hard clauses in PMS. U-SAT comprises the following three phases:

- **Unsupervised training with policy gradient.** We employ a neural network to provide a parameterized policy distribution, intended to guide the discovery of high-quality solutions. The training process of U-SAT focuses on optimizing the empirical expectation of the objective function. To handle the intricate objective function in PMS problems, U-SAT employs the policy gradient method to calculate the descent direction for each input problem instance. Following the computation of the policy descent direction, classical stochastic gradient methods are applied for training.
- **Decoding with recursive rounding and unit propagation.** Rather than simple rounding or sampling, we integrate unit propagation into the decoding process. In each iteration, variables with high scores are assigned, and the problem instance is simplified via unit propagation. The simplified instance is input into the neural network to generate a new distribution for the next iteration. This iterative process continues until no variable is predicted with a high score, and a final solution is completed using Metropolis-Hastings sampling. When dealing with large-scale instances, this method is effective and critical for its ability to reduce the scale of the problem.
- **Refining the model with weight adaption.** Learning to produce solutions with the penalized objective function in PMS problems becomes particularly challenging, as the training and decoding process may be easily trapped in local maxima, especially for complex cases. In response to this challenge, we employ weight adaptation to adjust the penalty factor and refine the models with the penalized factors. As a result, the refined model exhibits a robust capability to escape local minima.

We show the efficiency of our approach through evaluations conducted on randomly generated instances and real-world cases sourced from the MaxSAT Evaluation 2016 dataset[1]. The experimental results show that our method consistently outperforms or achieves comparable performance with the state-of-the-art incomplete solver. Additionally, we extend our comparison by evaluating U-SAT against alternative unsupervised learning frameworks on CO problems, which are encoded into PMS instances for evaluation. The results demonstrate that U-SAT outperforms all other methods in these scenarios.

## 2 Preliminaries

Given a Boolean variable $x$, a literal is either itself $x$ or its negation $\neg x$. A CNF formula comprises a conjunction of multiple clauses, each being a disjunction of multiple literals. For instance, the formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$ is in CNF with two clauses. Given a CNF formula, the goal of solving MaxSAT is to determine a variable assignment that maximizes the satisfied clauses.

PMS is a practical variant of MaxSAT that introduces additional hard clauses that have to be satisfied. A PMS problem instance $\mathcal{P}$ comprises the following components: Boolean variables $x \in \{-1, 1\}^n$, the set of soft clauses $C^{(s)}$ aimed at being satisfied as much as possible, and the set of hard clauses $C^{(h)}$ that must be

[1] http://maxsat.ia.udl.cat/introduction/

satisfied. $C^{(s)}$ and $C^{(h)}$ consist of $m^{(s)}$ and $m^{(h)}$ clauses, respectively. We then formulate the PMS problem as a binary programming problem as follows:

$$\begin{aligned} \max \quad & \sum_{c^i \in C^{(s)}} \max\{c_1^i x_1, c_2^i x_2, \cdots, c_n^i x_n, 0\}, \\ \text{s.t.} \quad & \max\{c_1^i x_1, c_2^i x_2, \cdots, c_n^i x_n, 0\} = 1, \text{ for } c^i \in C^{(h)}, \\ & x \in \{-1, 1\}^n, \end{aligned} \tag{1}$$

where $c_j^i$ represents the sign of variable $x_j$ in clause $c^i$ , i.e.,

$$c_j^i = \begin{cases} 1, & x_j \text{ appears in the clause } c^i, \\ -1, & \neg x_j \text{ appears in the clause } c^i, \\ 0, & \text{else.} \end{cases}$$

The constraints in the PMS problem can be transformed into a penalty term within the objective function. As the left side of the equality constraints in (1) is at least 0, the absolute function of the penalty can be omitted. Consequently, we have the following binary programming problem:

$$\max \sum_{c^i \in C^{(s)} \cup C^{(h)}} w_i \max\{c_1^i x_1, c_2^i x_2, \cdots, c_n^i x_n, 0\}, \quad \text{s.t.} \quad x \in \{-1, 1\}^n, \tag{2}$$

where $w_i = 1$ for $c^i \in C^{(s)}$ and $w_i$ is the assigned penalty factor for $c^i \in C^{(h)}$. In the subsequent sections, $w_i$ is referred to as the "weight". For convenience, we reformulate the objective function as a minimization form, i.e.,

$$f(x|\mathcal{P}) = \sum_{c^i \in C^{(s)} \cup C^{(h)}} w_i (1 - \max\{c_1^i x_1, c_2^i x_2, \ldots, c_n^i x_n, 0\}). \tag{3}$$

In the later discussion, we separate $f(x|\mathcal{P})$ into two components, $f^{(s)}(x|\mathcal{P})$ and $f^{(h)}(x|\mathcal{P})$, which represent the sum of terms corresponding to soft clauses and hard clauses, respectively. The equivalent penalized minimization problem of (2) becomes

$$\min \quad f(x|\mathcal{P}), \quad \text{s.t.} \quad x \in \{-1, 1\}^n. \tag{4}$$

We denote the feasible domain of (1) as $\mathcal{X}^{(h)}$, which is the set of all solutions that satisfy the hard clauses, i.e., $\mathcal{X}^{(h)} = \left\{x \in \{-1, 1\}^n \mid f^{(h)}(x|\mathcal{P}) = 0\right\}$. The following assumption then guarantees the exactness of the penalized problem:

**Assumption 1.** *The original problems* (1) *is feasible, and the penalty parameter* $w_i$ *in equation* (2) *is sufficiently large, which satisfies* $\min\limits_{c^i \in C^{(h)}} w_i > \max\limits_{x \in \mathcal{X}^{(h)}} f^{(s)}(x|\mathcal{P})$.

# 3 Methodology

In this section, we introduce our unsupervised end-to-end learning framework, named U-SAT. We first employ a heterogeneous graph neural network (HGNN) model $\psi$ with parameter $\theta$ to provide distribution $p_\theta(\cdot|\mathcal{P})$. This distribution is parameterized as a vector across variables, denoted as $[p_1, p_2, \cdots, p_n]$, where $n$ represents the number of variables. Then, we train the model in a differentiable manner over training problem instances via the policy gradient method. We further develop an recursive decoding process to generate solutions from the well-trained model. For hard problem instances, we refine the model by incorporating weight adaptation into the training process to escape from local maxima and obtain improved solutions.

## 3.1 Model architecture

In recent years, several attempts have been made to solve MaxSAT and other CO problems using GNNs. As shown in Figure 1(a), we first transform a CNF formula into an edge-splitting factor graph (ESFG), distinguishing between soft clauses and hard clauses. If a variable appears as a positive literal in a clause, they are connected by a solid line; otherwise, they are connected by a dotted line. The ESFG is a heterogeneous graph with three node types and two edge types. We then employ a simple-HGN [32] to handle the ESFG, treating it as node-splitting tasks on variable nodes. The model architecture details are provided in the appendix.

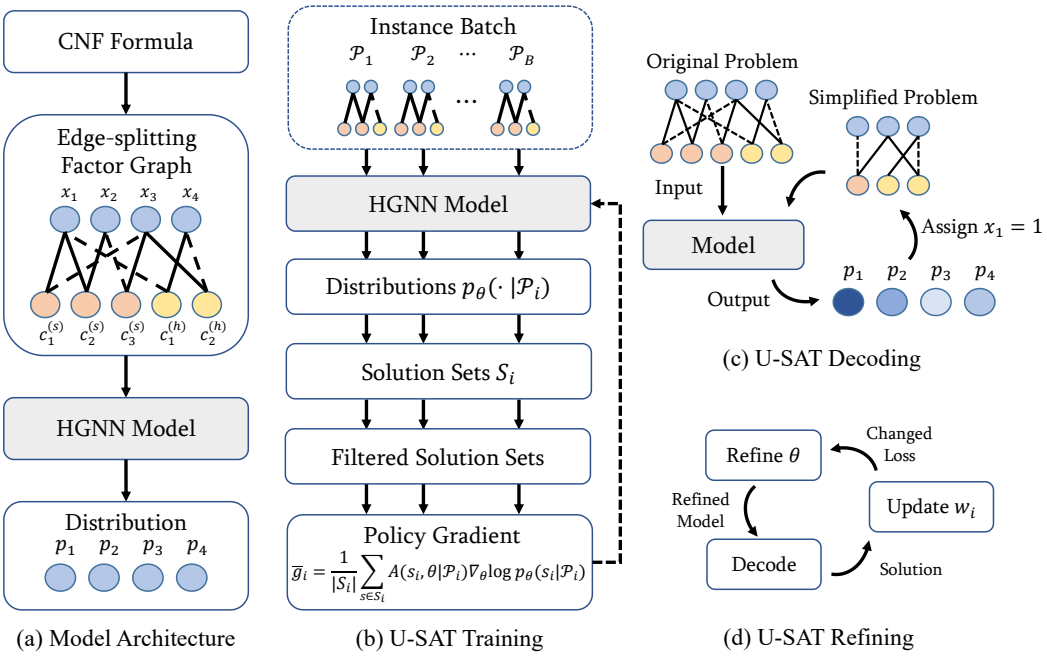


Figure 1: (a) The model architecture of the U-SAT involves transforming a PMS problem into an edge-splitting factor graph. In the illustrated graph, soft clauses include soft clause $x_1 \vee \neg x_3$, $x_1 \vee x_2$ and $\neg x_2 \vee x_3$ while hard clauses include $\neg x_1 \vee x_4$, $x_3 \vee \neg x_4$. The HGNN model takes the ESFG as input and outputs the variable-wise distribution. (b) During the training process, U-SAT first samples solutions from the output distribution and then applies a filter function. It then updates the model using the policy gradient method. (c) U-SAT employs a recursive decoding process. It only assigns variables with probabilities close to 0 or 1, then simplifies the problem and inputs it back into the model to generate a new distribution for the unassigned variables. (d) For intricate instances, U-SAT updates the penalty weights $w_i$ based on the satisfaction of hard clauses and then refines the model using a new loss function.

### 3.2 Unsupervised end-to-end training

In the U-SAT framework, we aim to train the model with parameters $\theta$ to guide the generation of high-quality solutions. Generally, we define the loss functions for a given problem instance $\mathcal{P}$ as $\mathcal{L}(\theta, \mathcal{P}) = \mathbb{E}_{p_\theta}[f(x|\mathcal{P})]$. In the previous work [33], a filter function $\mathcal{T}$ is applied on the $x$ to promote the algorithm's performance. $\mathcal{T}$ satisfies the following properties:

**Property 1.**

$$f(\mathcal{T}(x)|\mathcal{P}) \leq f(x|\mathcal{P});$$

**Property 2.**

$$f(\mathcal{T}(x)|\mathcal{P}) = f(x|\mathcal{P}) \Leftrightarrow \mathcal{T}(x) = x.$$

These properties ensure that the application of the filter function does not increase the value of the function $f$ and state that if applying the filter function does not change the value of the function $f$, then the filter should not change $x$. With the filter function $\mathcal{T}$, we define the loss function as

$$\mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P}) = \mathbb{E}_{p_\theta}[f(\mathcal{T}(x)|\mathcal{P})],$$

where $\mathcal{T}$ is the filter function that projects $x$ to a better solution on itself.

Given the training problem instance set $\Gamma = \{\mathcal{P}_1, \mathcal{P}_i, \cdots, \mathcal{P}_h\}$, the training process aims to optimize the following loss function over the uniform distribution on $\Gamma$:

$$L(\theta) = \mathbb{E}_{\mathcal{P}\sim\texttt{Uniform}(\Gamma)}\left[\mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P})\right] = \mathbb{E}_{\mathcal{P}\sim\texttt{Uniform}(\Gamma)}\left[\mathbb{E}_{x\sim p_\theta(\cdot|\mathcal{P})}[f(\mathcal{T}(x)|\mathcal{P})]\right]. \tag{5}$$

We will show that the policy gradient method does not require $f(\mathcal{T}(x)|\mathcal{P})$ to be differentiable over $x$. Thus, $\mathcal{T}$ could be an efficient deterministic algorithm, such as a local search or a greedy algorithm. For convenience, we represent $\mathbb{E}_{x\sim p_\theta(\cdot|\mathcal{P})}$ as $\mathbb{E}_{p_\theta(\cdot|\mathcal{P})}$ in the following part.

For an individual problem instance $\mathcal{P}$, the policy gradient of $\mathcal{L}_\mathcal{T}(\theta, \mathcal{P})$ is derived as

$$\nabla_\theta \mathcal{L}_\mathcal{T}(\theta, \mathcal{P}) = \mathbb{E}_{p_\theta(\cdot|\mathcal{P}_i)}[A(x, \theta|\mathcal{P})\nabla_\theta \log p_\theta(x|\mathcal{P})], \tag{6}$$

where $A(x, \theta|\mathcal{P}) = f(\mathcal{T}(x)|\mathcal{P}) - c$ and $c$ is an arbitrary constant. To compute the expectation in (6), we maintain a sample set $S$ and approximate the gradient as

$$\bar{g}(\theta, \mathcal{P}, S) = \frac{1}{|S|}\sum_{s\in S} A(s, \theta|\mathcal{P})\nabla_\theta \log p_\theta(s|\mathcal{P}). \tag{7}$$

Algorithm 1 shows the training process for a neural network designed to solve optimization problems. The

**Algorithm 1** U-SAT training process

**Input:** Training Problem Instances $\{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_h\}$.
**Set:** Learning rate $\alpha$, number of epochs $T$, solution-updating interval $t_0$, batch size $B$.
**Initialize:** Neural network parameters $\theta$ with random values, initial random solution sets for each problem instance $S_1, S_2, \cdots, S_h$ and their filtered solution $\hat{S}_i$.
**for** *iteration* $t \leftarrow 1$ ***to*** $T$ **do**
  **if** $t \sim \mathrm{mod} \sim t_0 = 0$ **then**
    **for** $i \leftarrow 1$ ***to*** $h$ **do**
      $S_i \leftarrow \texttt{MetroSampling}(p_\theta(\cdot|\mathcal{P}_i), \hat{S}_i)$ $\hat{S}_i \leftarrow \{\mathcal{T}(s)|s \in S_i\}$
  Randomly draw problem indices $\{i_1, i_2, \cdots, i_B\}$ Compute gradient $g^{(t)} \leftarrow \frac{1}{B}\sum_{j=1}^{B} \bar{g}(\theta, \mathcal{P}_{i_j}, S_{i_j})$ Compute direction $d^{(t)} = \texttt{Adam}(g^{(t)}, d^{(t-1)})$ Update parameters: $\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha \cdot d^{(t)}$

training process is tailored for a set of instances, denoted as $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_h\}$. The algorithm initializes the neural network parameters $\theta$ with random values and sets up initial solution sets $S_1, S_2, \ldots, S_h$ for each problem instance.

For each iteration $t$, a problem instance $\mathcal{P}_i$ is randomly selected. The gradient of the neural network, denoted as $g^{(t)}$, is computed using the gradient estimation $\bar{g}(\theta, \mathcal{P}_i, S_i)$ according to (7). The Adam optimizer is then employed to calculate the update direction $d^{(t)}$. The neural network parameters are updated using the computed direction, following the stochastic gradient descent paradigm.

At regular intervals determined by the solution-updating interval $t_0$, the algorithm triggers a two-step process for each problem instance $i$ among the set of instances $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_h$. Firstly, we update solution sets $S_i$ through a Metropolis-Hastings sampling technique based on the neural network's probability distribution $p_\theta(\cdot|\mathcal{P}_i)$. Secondly, a local search is applied to the sampled solutions, refining them and updating the set $\hat{S}_i$, which will be the start points for the next solution-updating process.

The U-SAT training process, as outlined, undergoes multiple epochs, iteratively refining neural network parameters and updating solution sets. This integration of neural networks with simple heuristic methods, i.e., local search, highlights the versatility and effectiveness of the U-SAT algorithm in overcoming challenges encountered by previous training methods.

### 3.3 Decoding with recursive rounding and unit propagation

In this subsection, we introduce the decoding process within the U-SAT framework. We initialize the candidate solution set $S$ and the best solution $s^*$ with random values. As in Section 3.2, the neural network $\psi$ generates a distribution $p = [p_1, p_2, \cdots, p_n]$, representing its predictions for solution assignments. We define the confidence score as follows

$$\mathrm{Score}(i) = \max\{1 - p_i, p_i\}. \tag{8}$$

The neural network exhibits low confidence in assigning values for variables whose output probabilities are close to 0.5. Consequently, previous works on CO prove that relying solely on straightforward rounding for solution construction may lead to diminished performance.

U-SAT employs the recursive rounding strategy: At each iteration, we only assign values on the variables with the confidence score surpassing a threshold $\delta$. Then, the problem can be significantly reduced by the assignments through unit propagation. This iterative process continues until a predefined number of iterations or until the scores of all unassigned variables fall below the threshold. Following the recursive rounding process, we fix the assigned variables and then sample from the output distribution over the unassigned variables. We apply local search as post-process on samples to obtain the best solution $s^*$. The details of our decoding process are shown in the appendix.

### 3.4 Refining with weight adaption

The PMS problem instances are typically modeled from various types of problems. Although previous work has demonstrated that neural networks on CO have the potential to generalize on out-of-distribution instances, it often fails for sophisticated instances. Instead, we design a refining process for U-SAT. It aims to refine the trained models to provide improved quality solutions for a specified problem instance. This refining process can be viewed as an iterative algorithm.

In each iteration, U-SAT outputs a solution $s^*$ based on the decoding process. For each hard clause $c_i \in C^{(h)}$, the weight $w_i$ is updated based on the satisfaction status. If $c_i$ is unsatisfied, $w_i$ is incremented; otherwise, it is set to the initial penalty factor $\beta$. Then the algorithm performs policy gradient to update the neural network parameters similar to Algorithm 1, with the updated penalty weights $w_i$. The details of our refining process are shown in the appendix.

## 4 Theoretical analysis

In this section, we provide the theoretical guarantees of the U-SAT framework, especially for the training process. We first demonstrate that the loss function $\mathcal{L}$ serves as an upper bound for the objective function $f$ from a probabilistic perspective. Then, we show that the general filter function $\mathcal{T}$ can significantly improve this upper bound. In the end, we prove the first-order convergence of the training algorithm. Theoretical results ensure the effectiveness of our U-SAT framework during modeling. We leave the proofs of the main conclusions in the appendix.

### 4.1 Loss function $\mathcal{L}$

The following theorem demonstrates that the loss function $\mathcal{L}$ can offer an upper bound for the objective function $f$, which is inspired by the Erdős framework [24].

**Theorem 1.** *Under Assumption 1, and given that* $\mathcal{L}(\theta, \mathcal{P}) < \min_{c^i \in C^{(h)}} w_i$, *with a probability of at least* $t$, $x \sim p_\theta$ *satisfies:*

$$f^{(s)}(x|\mathcal{P}) < \frac{\mathcal{L}(\theta, \mathcal{P})}{1-t} \quad \textit{and} \quad f^{(h)}(x|\mathcal{P}) = 0.$$

It's worth mentioning that our defined $\mathcal{L}(\theta, \mathcal{P})$ is slightly different from the $\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P})$ defined in [24] with respect to the penalty term. However, an inequality relationship exists between them under certain conditions. This property ensures the validity of the theorem, as detailed in the appendix. In conclusion, our penalized loss function has two main roles. First, it confirms the existence of a comparatively optimal solution. At the same time, it ensures that the constraints are strictly followed.

### 4.2 Filter function $\mathcal{T}$

Without comprehensive knowledge of the problem and the structure of $\mathcal{T}$, we explore the role of $\mathcal{T}$ from a probabilistic perspective. Specifically, $\mathcal{T}$ is randomly chosen from the candidate set of the filter functions that satisfies Assumption 2:

**Assumption 2.** *Let* $\{s_1, \cdots, s_N\} = \{-1, 1\}^n$ *be a set where* $f(s_1|\mathcal{P}) \leq f(s_2|\mathcal{P}) \leq \cdots \leq f(s_N|\mathcal{P})$ *and integers* $n_i \geq n_{min} \geq 2$. *For every* $x \in \{-1, 1\}^n$, $\mathcal{T}(x) = \text{argmax}_{x' \in N(x)} f(x')$, *where* $|N(s_i)| = n_i$ *and the* $n_i - 1$ *elements in* $N(s_i) - \{s_i\}$ *are uniformly selected from* $\{-1, 1\}^n$.

Our goal is to demonstrate that a randomly chosen $\mathcal{T}$ enhances the performance, implying that a significant proportion of filter functions are efficient. Let $\mathcal{X}^*$ denote the set of optimal points $x^*$, and let $r = |\mathcal{X}^*|/2^n$. We provide the following theorem of the upper bound.

**Theorem 2.** *Under Assumption 1, and let $\mathcal{T}$ be a random filter function chosen from the candidate set that satisfies Assumption 2. Given that $\mathcal{L}(\theta, \mathcal{P}) - f^*_{\mathcal{P}} < \min_{c^i \in C^{(h)}} w_i$, with a probability of at least $t\delta$, $x \sim p_\theta$ satisfies:*

$$f^{(s)}(\mathcal{T}(x)|\mathcal{P}) - f^*_{\mathcal{P}} \leq \frac{(1-r)^{n_{\min}-1}(\mathcal{L}(\theta, \mathcal{P}) - f^*_{\mathcal{P}})}{(1-t)(1-\delta)} \quad \textit{and} \quad f^{(h)}(x|\mathcal{P}) = 0.$$

This theorem demonstrates that as the exploration scale $n_{\min}$ increases, the gap between the objective function $f$ and the optimal function value $f^*$ decays exponentially in a probabilistic sense, which illustrates the effectiveness of $\mathcal{T}$.

### 4.3 Convergence of SGD

Following the common assumption shown in previous works, we provide the convergence analysis of the U-SAT training process. For ease of notation, we let $\phi(x;\theta) = \log p_\theta(x)$. Assuming that for all $x$, $\phi(x;\theta)$ is $L$-smooth with respect to $\theta$, and $\sup_\theta \|\nabla_\theta \phi(x;\theta)\| \leq M$, we can establish the first-order convergence as follows.

**Theorem 3.** *Let $\{\theta^{(t)}\}$ be generated by Algorithm 1. During the sampling process, we choose $k$ starting points and construct $m$ Markov chains for each of them. If the stepsize is chosen as $\eta^t = \frac{c\sqrt{mk}}{\sqrt{t}}$ with $c \leq \frac{1}{2l}$ and $B$ is the batch size, then we have*

$$\min_{1 \leq t \leq \tau} \|\nabla_\theta L(\theta^{(t)})\|^2 \leq O\left(\frac{\log \tau}{\sqrt{\tau}} \sqrt{\left(\frac{1}{mk} + \frac{1}{m\sqrt{B}} + \frac{1}{B}\right)} + \frac{1}{m^2}\right). \tag{9}$$

## 5 Experiments

In this section, we provide the numerical results of the U-SAT framework. The experiments are carried out on datasets including randomly generated data, instances modeled from CO problems, and large-scale real-world instances from the MSE competition. We do not apply the refining process for the former two kinds of data, which correspond to the first two subsections. In our experimental setup, we employ the simple-HGN [32] as the base model, where parameters are almost the same as in the original work. The optimization process for all experiments is executed using the ADAM optimizer with a consistent learning rate of $1e-2$. The implementation of our model is realized in PyTorch, and the training process is performed on an NVIDIA A100 GPU. These settings are kept the same for the following subsections.

Table 1: Numerical results on synthetic data. We report the number of satisfied soft clauses and the ratio of satisfied soft clauses to the total number of soft clauses.

| | SATLike | | MCPG | | | U-SAT | | |
|---|---|---|---|---|---|---|---|---|
| Problem Set | obj | ratio | obj | ratio | time(s) | obj | ratio | time(s) |
| PL(500, 2000) | 610.8 | 0.825 | 586.8 | 0.793 | 11.0 | 602.8 | 0.815 | 3.3 |
| PL(500, 3000) | 900.9 | 0.781 | 892.6 | 0.773 | 12.0 | 896.5 | 0.777 | 4.4 |
| PL(1000, 4000) | 1192.1 | 0.824 | 1144.6 | 0.792 | 22.1 | 1151.2 | 0.796 | 6.8 |
| PL(2000, 8000) | 2463.6 | 0.824 | 2397.4 | 0.802 | 45.4 | 2402.0 | 0.803 | 11.2 |

### 5.1 Synthetic datasets

In this subsection, we demonstrate the effectiveness of U-SAT on synthetic datasets. We randomly generate the training datasets based on double Power-Law distribution [34], a non-uniform generating function that aims to simulate the characteristics of real-world instances. The datasets are denoted by PL$(n, m)$, where $n$ denotes the number of variables and $m$ denotes the number of clauses. The training dataset comprises 2000 instances, each of which is generated by PL$(500, 2000)$, and then changes the clauses with lengths of more than 2 to be hard clauses. The average length of clauses is set to 3. We utilize one of the state-of-the-art solvers, SATlike [35], to confirm the feasibility of all instances with the introduced hard clauses and provide the ground truth for testing by running it for 30 seconds per instance. Furthermore, we ensure that none of these instances can be solved by a complete solver within a 5-minute time limitation.

Table 1 shows the numerical results on synthetic datasets, each containing 500 instances. To demonstrate the effectiveness of U-SAT with MCPG [33], an iterative algorithm based on sampling and policy gradient approach to solving binary programming problems. The U-SAT model is only trained on PL(500, 2000) but also generalizes on other datasets. It shows that U-SAT maintains performance superior to MCPG and consumes less time. Particularly, U-SAT performs well when solving instances with more variables and clauses, indicating that the U-SAT framework is promising and generalizes well to much larger problems.

## 5.2 Instances modeled from CO problems

We evaluate U-SAT across various combinatorial problem instances, including instances from maximum clique (MC), maxcut, and minimum dominating set (MDS) problems. Following the previous work [36], we evaluate MC on RB graphs [37] and evaluate the other two tasks on BA graphs [38]. For all types of synthetic graph data, we generate two scales of datasets, which respectively contain around 200 to 300 vertices and 800 to 1200 vertices.

To apply the U-SAT framework, we first transform these problem instances into PMS instances. We then compare U-SAT with five unsupervised learning methods, including Erdős [24], Anneal [31], PI-GNN [25], RUN-CSP [23] and GFlowNet [36]. Erdős and Anneal are the two state-of-the-art probabilistic methods. GFlowNet is the latest unsupervised learning approach to solve CO problems. Since the original work provides no direct method, we only evaluate PI-GNN and RUN-CSP on MC and maxcut problems. For all datasets, we generate 4000 graphs for training, 500 for validation, and 500 for testing as GFlowNet. The performance of each method is evaluated based on three key metrics: the size of the objective value (size), the drop in performance compared to the optimal solution generated from gurobi (Drop), and the inference time.

Table 2: Experimental Results on max clique (MC), maxcut, and min dominating set (MDS). We report the size of output solutions, the drop to the optimal solutions, and the inference time. The results of gurobi is sourced from GFlowNet.

| | MC | | | maxcut | | | MDS | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | size ↑ | drop ↓ | time(s) | size ↑ | drop ↓ | time(s) | size ↓ | drop ↓ | time(s) |
| Small Graph ($\|V\|$ between 200 and 300). | | | | | | | | | |
| gurobi | 19.05 | 0.00% | 115 | 732.5 | 0.00% | 784 | 27.89 | 0.00% | 107 |
| Erdős | 12.52 | 34.29% | 29 | 695.2 | 5.09% | 46 | 30.71 | 9.18% | 43 |
| Anneal | 14.22 | 25.37% | 31 | 698.8 | 4.60% | 45 | 29.22 | 4.55% | 43 |
| PI-GNN | 11.52 | 39.54% | 20 | 688.5 | 6.00% | 77 | - | - | - |
| RUN-CSP | 12.84 | 32.58% | 284 | 692.6 | 5.44% | 277 | - | - | - |
| GFlowNet | 16.28 | 14.55% | 33 | 705.7 | 3.66% | 177 | 28.55 | 2.31% | 111 |
| U-SAT | **16.69** | 12.39% | 42 | **712.5** | 2.73% | 52 | **28.32** | 1.52% | 79 |
| Large Graph ($\|V\|$ between 800 and 1200). | | | | | | | | | |
| gurobi | 33.89 | 0.00% | 1000 | 2915.3 | 0.00% | 3929 | 103.8 | 0.00% | 828 |
| Erdős | 25.64 | 24.34% | 99 | 2872.8 | 1.46% | 119 | 116.57 | 10.95% | 173 |
| Anneal | 27.61 | 18.55% | 101 | 2867.9 | 1.63% | 128 | 110.09 | 5.71% | 167 |
| PI-GNN | 20.29 | 40.14% | 101 | 2803.8 | 3.82% | 422 | - | - | - |
| RUN-CSP | 21.35 | 37.00% | 781 | 2834.4 | 2.78% | 679 | - | - | - |
| GFlowNet | 31.61 | 6.73% | 221 | 2870.2 | 1.55% | 906 | 109.95 | 5.59% | 1444 |
| U-SAT | **32.15** | 5.13% | 158 | **2892.1** | 0.80% | 112 | **107.63** | 3.56% | 149 |

Table 2 shows the numerical results on instances modeled from CO problems. Compared to other algorithms, our method demonstrates a remarkable ability to obtain the best results across the unsupervised methods and keep a minimal drop in performance. It is worth mentioning that the comparison of inference time is potentially unfair due to the implementation details. For example, the decoding process of U-SAT employs `Numba` to accelerate Python functions. On the other hand, we observe that U-SAT may improve the solutions

by consuming more time, but we fix the number of iterations and interrupt this process when U-SAT achieves the best but not optimal solution. Nevertheless, the computational time of our method is competitive with other high-performance algorithms, underscoring its efficiency.

### 5.3 Large-scale PMS instances

We evaluate the U-SAT framework on large-scale real-world datasets and apply the refining process to promote its performance. The refining process is based on the model trained on randomly generated PMS instances. Our evaluation includes comparing U-SAT and SATlike, a state-of-the-art MaxSAT solver based on local search. The results are evaluated based on the relative gaps to the best solutions, which are either achieved by U-SAT or SATlike. The instances used in the evaluation are from the unweighted PMS track of the MSE 2016 competition. The running time for all instances was limited to 60 seconds.

Table 3: Results on instances from MSE 2016 competition.

| | pms_random | | | | pms_crafted | | | | pms_industrial | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Metrics | U-SAT | | SATlike | | U-SAT | | SATlike | | U-SAT | | SATlike | |
| gap | num | pct | num | pct | num | pct | num | pct | num | pct | num | pct |
| 0.00 (best) | 193 | 0.919 | 158 | 0.752 | 449 | 0.662 | 444 | 0.655 | 366 | 0.609 | 383 | 0.637 |
| $(0, 0.01]$ | 16 | 0.076 | 51 | 0.243 | 44 | 0.065 | 32 | 0.047 | 97 | 0.161 | 87 | 0.145 |
| $[0.01, 0.05)$ | 1 | 0.005 | 1 | 0.005 | 16 | 0.024 | 25 | 0.037 | 29 | 0.048 | 21 | 0.035 |
| $[0.05, 0.10)$ | 0 | 0.000 | 0 | 0.000 | 13 | 0.019 | 6 | 0.009 | 6 | 0.010 | 4 | 0.007 |
| $> 0.10$ | 0 | 0.000 | 0 | 0.000 | 4 | 0.006 | 11 | 0.016 | 2 | 0.003 | 2 | 0.003 |
| unsolved | 0 | 0.000 | 0 | 0.000 | 152 | 0.224 | 160 | 0.236 | 101 | 0.168 | 104 | 0.173 |
| total | 210 | | 210 | | 678 | | 678 | | 601 | | 601 | |

Table 4: Results on instances from MSE 2016 competition.

| | pms_random | | | | pms_crafted | | | |
|---|---|---|---|---|---|---|---|---|
| Metrics | U-SAT | | SATlike | | U-SAT | | SATlike | |
| gap | num | pct | num | pct | num | pct | num | pct |
| 0.00 (best) | 193 | 0.919 | 158 | 0.752 | 449 | 0.662 | 444 | 0.655 |
| $(0, 0.01]$ | 16 | 0.076 | 51 | 0.243 | 44 | 0.065 | 32 | 0.047 |
| $> 0.01$ | 1 | 0.005 | 1 | 0.005 | 185 | 0.273 | 202 | 0.298 |

Table 4 summarizes the experimental results. It indicates the number of instances where each algorithm successfully achieved optimal results, i.e., a zero gap. Notably, U-SAT outperforms SATlike on the random track and finds feasible solutions for more instances for both the crafted and industrial tracks. It also identifies more optimal solutions on the crafted track. Although U-SAT does not outperform SATlike on the industrial track, mainly because of the overly large scale, it demonstrates that the refining process based on a trained model has the potential for high efficiency.

## 6 Conclusion

This paper investigates the application of end-to-end learning to address PMS problems. A generic learning framework is proposed and designed to train the model in a differentiable manner. The decoding process is detailed, emphasizing the construction of solutions through unit propagation to reduce the problems. For challenging instances where the model's generalization struggles to satisfy constraints, a refining process is introduced, which involves adapting the weights of the pre-trained model using specialized techniques.

The limitation of this work is generalization to intricate and extremely large problems. PMS problems may

arise from various kinds of CO problems or real-world scenarios. It is impractical to train models for each specific problem source type. Due to the complexity of PMS problems, this article does not resolve this issue, and we leave the adjustment to future work. We hope our work can provide new insights into learning a more efficient algorithm for constrained combinatorial optimization problems in the research community.

# References


[1] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8:239–287, 2009.

[2] Angel A Juan, Javier Faulin, Scott E Grasman, Markus Rabe, and Gonçalo Figueira. A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, 2:62–72, 2015.

[3] Mohamed Abdel-Basset, Laila Abdel-Fatah, and Arun Kumar Sangaiah. Metaheuristic algorithms: A comprehensive review. *Computational intelligence for multimedia big data on the cloud with engineering applications*, pages 185–231, 2018.

[4] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

[5] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.

[6] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. In *Advances in Neural Information Processing Systems*, pages 7990–8001, 2019.

[7] Wei Zhang, Ziyang Sun, Qingtai Zhu, Guangyong Li, Shaowei Cai, Yunhao Xiong, and Lintao Zhang. Nlocalsat: Boosting local search with solution prediction. In *Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 1177–1183, 2020.

[8] Daniel Selsam and Nikolaj Bjørner. Guiding high-performance sat solvers with unsat-core predictions. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 336–353. Springer, 2019.

[9] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[10] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6348–6358, 2017.

[11] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[12] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 134–144. PMLR, 2020.

[13] Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[14] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.*, 134:105400, 2021.

[15] Jack Goffinet and Raghuram Ramanujan. Monte-carlo tree search for the maximum satisfiability problem. In *International Conference on Principles and Practice of Constraint Programming*, pages 251–267. Springer, 2016.

[16] Mourad Lassouaoui, Dalila Boughaci, and Belaid Benhamou. A multilevel synergy thompson sampling hyper-heuristic for solving max-sat. *Intelligent Decision Technologies*, 13(2):193–210, 2019.

[17] Junjie Zheng, Kun He, Jie Zhou, Youwei Jin, Chu-Min Li, and Felip Manyà. Bandmaxsat: A local search maxsat solver with multi-armed bandit. In *The Thirty-First International Joint Conference on Artificial Intelligence*, 2022.

[18] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 09–15 Jun 2019.

[19] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. In *International Conference on Learning Representations*, 2019.

[20] Chris Cameron, Rex Chen, Jack S Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3324–3331, 2020.

[21] Minghao Liu, Pei Huang, Fuqi Jia, Fan Zhang, Yuchen Sun, Shaowei Cai, Feifei Ma, and Jian Zhang. Can graph neural networks learn to solve the maxsat problem?(student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 16264–16265, 2023.

[22] S. Amizadeh, S. Matusevych, and M. Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2019.

[23] J. Toenshoff, M. Ritzert, H. Wolf, and M. Grohe. Run-csp: unsupervised learning of message passing networks for binary constraint satisfaction problems. 2019.

[24] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *NeurIPS 2020 34th Conference on Neural Information Processing Systems*, 2020.

[25] Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

[26] Haoyu Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *NeurIPS*, 2022.

[27] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *NeurIPS*, 2022.

[28] Nikolaos Karalias, Joshua Robinson, Andreas Loukas, and Stefanie Jegelka. Neural set function extensions: Learning with discrete functions in high dimensions. In *NeurIPS*, 2022.

[29] Yimeng Min, Frederik Wenkel, Michael Perlmutter, and Guy Wolf. Can hybrid geometric scattering networks help solve the maximum clique problem? In *NeurIPS*, 2022.

[30] Haoyu Peter Wang and Pan Li. Unsupervised learning for combinatorial optimization needs meta learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[31] Haoran Sun, Etash Kumar Guha, and Hanjun Dai. Annealed training for combinatorial optimization on graphs. In *OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop)*, 2022.

[32] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 1150–1160, 2021.

[33] Cheng Chen, Ruitao Chen, Tianyou Li, Ruichen Ao, and Zaiwen Wen. Monte carlo policy gradient method for binary optimization. *arXiv preprint arXiv:2307.00783*, 2023.

[34] Carlos Ansótegui, Maria Bonet, and Jordi Levy. Towards industrial-like random sat instances. pages 387–392, 01 2009.

[35] Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.

[36] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial optimization problems with gflownets, 2023.

[37] K. Xu and W. Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103, Mar 2000.

[38] Albert-Laszlo Barabási and Reka Albert. Emergence of scaling in random networks. *Science*, 286 5439:509–12, 1999.

[39] Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:98, 2021.

# A Details of decoding and refining

## A.1 Decoding process

Algorithm 2 provides the pseudo-code for the decoding process. It maintains a problem instance $\mathcal{P}'$, initialized as a duplicate of the original problem instance. $\mathcal{P}'$ will be consistently simplified during the iterations. At iteration $t$, the selection of variable assignments, $x_i$, is determined based on their confidence scores according to Equation (8). If the score surpasses a threshold $\delta$, the algorithm proceeds to update the variable assignment $x_i$ and performs unit propagation on $\mathcal{P}'$ with this assignment. The iterative process continues until a predefined number of iterations or until the scores of all unassigned variables fall below the threshold.

Once the recursive rounding process is completed, the algorithm utilizes Metropolis-Hastings sampling on the output distribution over the unassigned variables to obtain a set of solutions $S$. Subsequently, a local search is applied to each sampled solution over the unassigned variables, resulting in a refined set $\hat{S}$ to satisfy the hard clauses. The best solution, $s^*$, is then selected from the union of the refined set and the current best solution.

**Algorithm 2** U-SAT Decoding Process

**Input :** Neural network $\psi$ with parameter $\theta$, original problem instance $\mathcal{P}$ with variable size $n$.
**Initialize:** Initial solution sets $S$ with random values, initial best solution $s^*$ with random values.
$\mathcal{P}' \leftarrow \mathcal{P}$
**for** *iteration* $t \leftarrow 1$ ***to*** $n$ **do**
  $p \leftarrow p_\theta(\cdot|\mathcal{P}')$  $I \leftarrow \{i \sim | \sim \text{Score}(i) > \delta\}$  **if** $I$ *is empty* **then**
    **break**
  **for** $i \in I$ **do**
    $z \leftarrow \text{sign}(p_i - 0.5)$  $x \leftarrow z$  $\mathcal{P}' \leftarrow \texttt{UnitPropagation}(\mathcal{P}', x_i = z)$
$S \leftarrow \texttt{MetroSampling}(\psi(\mathcal{P}'|\theta), \hat{S})$  $\hat{S} \leftarrow \{\texttt{LocalSearch}(s, \mathcal{P}') \mid \text{for } s \text{ in } S\}$  $s^* \leftarrow \texttt{SelectBest}(\hat{S} \cup s^*)$
**Output :** $s^*$

## A.2 Refining process

Algorithm 2 provides the pseudo-code for the refining process. In each iteration, the refining algorithm outputs a solution $s^*$ based on Algorithm 2. Subsequently, for each unsatisfied constraint $c_i$ in the set $C^{(h)}$, penalty weights $w_i$ are updated based on the satisfaction status of $s^*$. If $c_i$ is unsatisfied, the penalty weight $w_i$ is incremented; otherwise, it is set to the initial penalty factor $\beta$.

In the subsequent phase, the algorithm performs gradient-based optimization to update the neural network parameters, which is similar to Algorithm 1. Gradients are computed with respect to the updated penalty weights $w_i$. This process is repeated for a few number of steps $k$. The iterative refinement enhances the neural network's ability to output solutions that better satisfy the given constraints.

The U-SAT Refining Process concludes by outputting the best solution $s^*$ obtained during the refinement iterations. This refined solution reflects the improved adaptation of the algorithm to the specific problem instance, extending the capability of the U-SAT approach in refining solutions based on neural network predictions and constraint satisfaction.

**Algorithm 3** U-SAT Refining Process

**Input :** Neural network $\psi$ with parameter $\theta$, problem instance $\mathcal{P}$ with variable size $n$, initial penalty factor $\beta$.
$\mathcal{P}' \leftarrow \texttt{ReduceProblem}(\psi, \mathcal{P})$
**for** *iteration* $\leftarrow 1$ ***to*** $T$ **do**
  $s^* \leftarrow \texttt{Predict}(\psi, \mathcal{P})$  **for** $c_i$ ***in*** $C^{(h)}$ **do**
    **if** $c_i$ *is unsatisfied with* $s^*$ **then**
      penalty weight $w_i \leftarrow w_i + 1$
    **else**
      penalty weight $w_i \leftarrow \beta$
  $\psi \leftarrow \texttt{Train}(\psi, \{\mathcal{P}\}, w)$
**Output:** the best $s^*$ during the iterations.

### A.3 Unit Propagation

Unit Propagation (UP) is a well-recognized technique used in SAT solvers. A significant number of decimation algorithms are based on UP due to its two key properties: preserving the instance's satisfiability and potentially simplifying the CNF formula significantly.

In our work, we implement UP in the decoding process, triggered when an value is assigned to the variable surpassing the confidence score threshold $\delta$ at each iteration. A fixed variable assignment $x_i$ may lead to the simplification of clauses, producing more unit clauses. For any unit clause, the variable is assigned to satisfy it. In other words, if the unit clause $\{x_i\}$ is present in the formula, $x_i$ is assigned as true. On the contrary, if the unit clause $\{\neg x_i\}$ appears, $x_i$ is assigned as false. The formula is then conditioned and possibly simplified on this setting, marking a unit propagation step. UP refers to repeating this procedure of identifying unit clauses and propagating simplifications after any value is assigned to the variables.

During unit propagation, a conflict may arise when both unit clauses $x_i$ and $\neg x_i$ appear. Managing this situation is crucial in decimation algorithms. In U-SAT, we address this by identifying when the unit propagation process fails and retracting the assignment action that led to this conflicting unit propagation process.

### A.4 Parallel MH sampling

In this section we introduce our sampling method. Although binary solutions are relatively straightforward to obtain, direct sampling based on the Bernoulli distribution lacks locality, and therefore precludes the use of previously obtained samples. Instead, we choose to use parallel MH sampling.

Algorithm 4 shows our sampling procedure for a single starting point, which is based on the Metropolis-Hastings algorithm to establish a Markov chain with the stationary distribution $p_\theta(\cdot)$.

**Algorithm 4** A Parallel MH algorithm with filter function

**Input:** The starting state $x_0$, transition number $t$, number of chain $m$, proposal distribution $Q(x'|x)$.
**Function** `sampling`($x_0$, $t$, $m$)**:**
  **for** $j = 1$ **to** $m$ **do in parallel**
    Copy the starting state $x_0^j = x_0$ for this chain **for** $v = 0$ **to** $t-1$ **do in parallel**
      Propose a new state $x'$ by sampling from a proposal distribution $Q(x'|x_v^j)$ Compute the acceptance probability $\alpha(x'|x_v^j) = \min\left(1, \frac{p_\theta(x_v^j)Q(x_v^j|x')}{p_\theta(x')Q(x'|x_v^j)}\right)$ Generate a random number $u \sim \text{Uniform}(0,1)$ **if** $u < \alpha(x'|x_v^j)$ **then**
        Set $x_{v+1}^j = x'$
      **else**
        Set $x_{v+1}^j = x_v^j$
    Obtain $s^j = x_t^j$
  **return** the sample set $S = \{s^1, s^2, \cdots, s^m\}$

In our sampling process, we begin with $k$ starting points, and construct $m$ Markov chains for each of them. The computation of the $km$ chains can be efficiently implemented with matrix operations on the modern computational device. With the distribution $p_{\theta^t}(\cdot)$, the transition procedure is the same as the ordinary MH algorithm. For all the chains, we only select the last states of a chain, and obtain a sample batch $S_i = \{s_i^1, s_i^2, ..., s_i^m\}$ for each of the starting point $s_i$. Hence, we get $k$ batches and obtain $km$ raw samples $s_i^j$ for $1 \le i \le k, 1 \le j \le m$.

## B Equivalence of the penalized problem

In this section, we prove that given the original problem is feasible and the penalty factor is sufficiently large, the penalized problem is equivalent to the original problem. We first revisit the objective function of the original PMS problem, which we reformulate as a minimization form:

$$f^{(s)}(x|\mathcal{P}) = \sum_{c^i \in C^{(s)}} (1 - \max\{c_1^i x_1, c_2^i x_2, \ldots, c_n^i x_n, 0\}).$$

This reformulation expresses (1) in an equivalent form:

$$\min \quad f^{(s)}(x|\mathcal{P}), \quad \text{s.t.} \quad x \in \mathcal{X}^{(h)}. \tag{10}$$

To incorporate constraints into the formulation, we introduce the penalized term

$$f^{(h)}(x|\mathcal{P}) = \sum_{c^i \in C^{(h)}} w_i(1 - \max\{c_1^i x_1, c_2^i x_2, \ldots, c_n^i x_n, 0\}).$$

Then, we express the penalized objective as

$$f(x|\mathcal{P}) = f^{(s)}(x|\mathcal{P}) + f^{(h)}(x|\mathcal{P}).$$

Subsequently, we have the equivalent penalized minimization problem (4). Intuitively, when the penalty parameter $w_i$ in problem (4) is sufficiently large, one would expect the problem (4) to be equivalent to (10), implying that our penalty framework is exact. Theorem 4 gives a formal description of this equivalence.

**Theorem 4.** *Under Assumption 1, problem* (4) *is equivalent to problem* (10)*, i.e., they have the same set of optimal solutions $\mathcal{X}^*$ and the same optimal objective function value $f^*_{\mathcal{P}}$.*

*Proof.* We first prove that if $x^*$ is the optimal solution of (4), it satisfies $f^{(h)}(x^*|\mathcal{P}) = 0$. Assuming for the sake of contradiction that $f^{(h)}(x^*|\mathcal{P}) \neq 0$, there exists $c^i \in C^{(s)}$ for which $\max\{c_1^i x_1, c_2^i x_2, \ldots, c_n^i x_n, 0\} = 0$ by the definition of $f^{(h)}$. In this case, we have $f^{(h)}(x^*|\mathcal{P}) \geq w_i$, leading to

$$f(x^*|\mathcal{P}) \geq f^{(h)}(x^*|\mathcal{P}) \geq \min_{c^i \in C^{(h)}} w_i.$$

According to Assumption 1, there exists an $x' \in \mathcal{X}^{(h)}$ such that $f^{(h)}(x'|\mathcal{P}) = 0$. Therefore, we have

$$f(x'|\mathcal{P}) = f^{(s)}(x^*|\mathcal{P}) \leq \max_{x \in \mathcal{X}^{(h)}} f^{(s)}(x|\mathcal{P}).$$

In conjunction with Assumption 1, we deduce that $f(x'|\mathcal{P}) < f(x^*|\mathcal{P})$, contradicting the optimality of $x^*$. This implies that any optimal solution $x^*$ of (4) satisfies $f^{(s)}(x^*|\mathcal{P}) = f(x^*|\mathcal{P})$ and $x^* \in \mathcal{X}^{(h)}$. Furthermore, given that $\mathcal{X}^{(h)} \subseteq \{-1, 1\}^n$, any optimal solution $\hat{x}^*$ of (10) also satisfies $f(\hat{x}^*|\mathcal{P}) = f^{(s)}(\hat{x}^*|\mathcal{P})$ and $\hat{x}^* \in \{-1, 1\}^n$. Hence, (10) and (4) are equivalent. □

Since (10) and (4) are the equivalent minimization forms of (1) and (2), these four optimization problems are equivalent if Assumption 1 holds.

# C Loss function $\mathcal{L}$

To provide the proof for Theorem 1, we first revisit the construction of the probability method. The original problem can be reformulated as:

$$\min f^{(s)}(x|\mathcal{P}) \quad \text{subject to} \quad x \in \mathcal{X}^{(h)}.$$

To incorporate constraints, we draw inspiration from the penalty function method in constraint optimization and add a penalty term. If we formulate the loss function $\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P})$ of the original constrained problem following the probability method in [39], the probability penalty function is defined as $f^p(x|\mathcal{P}) = f^{(s)}(x|\mathcal{P}) + 1_{x \notin \mathcal{X}^{(h)}}\beta$, where $\beta$ is a scalar. The expected value of $f^p(x|\mathcal{P})$ represents the probability penalty loss

$$\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P}) = \mathbb{E}_{p_\theta}[f^{(s)}(x|\mathcal{P})] + \beta P(x \notin \mathcal{X}^{(h)}). \tag{11}$$

Using the Markov inequality, the objective function $f^{(s)}(x|\mathcal{P})$ and the loss function $\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P})$ conform to the following bounding theorem, which is a special case of Theorem 1 in [39].

**Theorem 5.** *Assume that the objective function $f^{(s)}(x|\mathcal{P})$ is non-negative, fix any $\beta > \max_{x \in \{-1,1\}^n} f^{(s)}(x)$, and let $\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P}) < \beta$. With a probability of at least $t$, $x \sim p_\theta$ satisfies*

$$f^{(s)}(x|\mathcal{P}) < \frac{\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P})}{1 - t} \quad \textit{and} \quad x \in \mathcal{X}^{(h)},$$

**Remark 1.** *Theorem 1 of [39] considers a general objective function $F(x)$, and its non-negativity is required. However, according to our definition of $f^{(s)}(x|\mathcal{P})$, this condition is clearly satisfied. This is also the reason why we reformulated the objective function into the form of* (3)*.*

## C.1 Loss function $\mathcal{L}(\theta, \mathcal{P})$

In order to complete the proof of Theorem 1, we need to compare $\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P})$ with the loss function $\mathcal{L}(\theta, \mathcal{P})$ we use. In fact, when the penalty factor satisfies certain conditions, they have the inequality relationship we require, denoted as lemma 1. The proof of this lemma is based on union bound, and it fully considers the feasible domain structure of the MaxSAT problem.

**Lemma 1.** *If the penalty factors corresponding to the two loss functions satisfy* $\beta \leq \min_{c^i \in C^{(h)}} w_i$*, then*

$$\mathcal{L}(\theta, \mathcal{P}) \geq \mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P}).$$

*Proof.* Let $\mathcal{X}_i$ denote the set of points in $\{-1, 1\}^n$ that satisfy the clause $c_i$, then $\mathcal{X}^{(h)} = \bigcap_{c^i \in C^{(h)}} \mathcal{X}_i$. By union bound, we have

$$\sum_{c^i \in C^{(h)}} P_{p_\theta}(x \notin \mathcal{X}_i) \geq P_{p_\theta}(x \notin \mathcal{X}^{(h)}). \tag{12}$$

We then derive the following relationships we need. According to the definition of $\mathcal{L}(\theta, \mathcal{P})$,

$$\begin{aligned} \mathcal{L}(\theta, \mathcal{P}) &= \mathbb{E}_{p_\theta}[f(x|\mathcal{P})] \\ &= \mathbb{E}_{p_\theta}[f^{(s)}(x|\mathcal{P})] + \mathbb{E}_{p_\theta}[f^{(h)}(x|\mathcal{P})]. \end{aligned} \tag{13}$$

Note that $f^{(s)}(x|\mathcal{P}) = \sum_{c^i \in C^{(h)}} w_i(x \notin \mathcal{X}_i)$, by interchanging the order of summation, we obtain:

$$\begin{aligned} \mathbb{E}_{p_\theta}[f^{(h)}(x|\mathcal{P})] &= \sum_x p_\theta(x) \sum_{c^i \in C^{(h)}} w_i(x \notin \mathcal{X}_i) \\ &= \sum_{c^i \in C^{(h)}} w_i \sum_x p_\theta(x)(x \notin \mathcal{X}_i) \\ &= \sum_{c^i \in C^{(h)}} w_i P_{p_\theta}(x \notin \mathcal{X}_i). \end{aligned} \tag{14}$$

Given that $w_i \geq \beta$, and combining with equation (12), we obtain

$$\begin{aligned} \sum_{c^i \in C^{(h)}} w_i P_{p_\theta}(x \notin \mathcal{X}_i) &\geq \beta \sum_{c^i \in C^{(h)}} P_{p_\theta}(x \notin \mathcal{X}_i) \\ &\geq \beta P_{p_\theta}(x \notin \mathcal{X}^{(h)}). \end{aligned} \tag{15}$$

Combining (13), (14), (15), (11), lemma 1 is proven. □

## C.2 Proof of theorem 1

By combining Theorem 5 and Lemma 1, we finally conclude the proof of Theorem 1:

*Proof.* Fix $\beta = \min_{c^i \in C^{(h)}} w_i$, then $\mathcal{L}(\theta, \mathcal{P}) < \beta$. Notice that $\min_{c^i \in C^{(h)}} w_i > \max_{x \in \{-1,1\}^n} f^{(s)}(x)$ according to Assumption 1, and $\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P}) \leq \mathcal{L}(\theta, \mathcal{P})$ according to Lemma 1, thus $\beta > \max_{x \in \{-1,1\}^n} f^{(s)}(x)$ and $\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P}) < \beta$, which indicates that conditions of Theorem 5 are satisfied. Combining with Lemma 1, we know that with probability at least $t$, $x \sim p_\theta$ satisfies

$$f^{(s)}(x|\mathcal{P}) < \frac{\mathcal{L}_{\mathcal{E}}(\theta, \mathcal{P})}{1-t} \leq \frac{\mathcal{L}(\theta, \mathcal{P})}{1-t} \quad \text{and} \quad x \in \mathcal{X}^{(h)},$$

under the condition that $f^{(s)}(x|\mathcal{P})$ is non-negative. □

### C.3 A strengthened corollary

Notice that in the proof of Theorem 1, we only require a non-negative lower bound for $f^{(s)}(x|\mathcal{P})$. Therefore, if we replace it with $f^{(s)}(x|\mathcal{P}) - f^{(s)}(x^*|\mathcal{P}) =: \tilde{f}^{(s)}(x|\mathcal{P})$, this conclusion still holds.

Since the original problem (1) is feasible (Assumption 1), by Theorem 4, $f^{(s)}(x^*|\mathcal{P}) = f(x^*|\mathcal{P}) = f^*_{\mathcal{P}}$, the corresponding loss function is derived as $\tilde{\mathcal{L}}(\theta, \mathcal{P}) = \mathbb{E}_{p_\theta}[\tilde{f}^{(s)}(x|\mathcal{P})] = \mathcal{L}(\theta, \mathcal{P}) - f^*_{\mathcal{P}}$. Hence, there stands the following strengthened result:

**Corollary 1.** *Under Assumption 1, and given that* $\mathcal{L}(\theta, \mathcal{P}) - f^*_{\mathcal{P}} < \min_{c^i \in C^{(h)}} w_i$, *with a probability of at least* $t$, $x \sim p_\theta$ *satisfies:*

$$f^{(s)}(x|\mathcal{P}) - f^*_{\mathcal{P}} < \frac{\mathcal{L}(\theta, \mathcal{P}) - f^*_{\mathcal{P}}}{1-t} \quad \textit{and} \quad x \in \mathcal{X}^{(h)}.$$

Since $f^*_{\mathcal{P}}$ is not easy to obtain in general problems, this Corollary only serves as a theoretical guarantee to prove the conclusions in the following text.

## D Filter function $\mathcal{T}$

To prove Theorem 2, we first point out the following Lemma 1:

**Theorem 4.4** For $\delta \in (0, 1)$, then with probability at least $\delta$ over the choice of $\mathcal{T}$,

$$\frac{\mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P}) - f^*_{\mathcal{P}}}{\mathcal{L}(\theta, \mathcal{P}) - f^*_{\mathcal{P}}} \leq \frac{(1-r)^{n_{min}-1}}{1-\delta}.$$

*Proof.* Let $M = |\mathcal{X}^*|$, and without loss of generality $\forall j \leq M$, $f(s_j) = f^*_{\mathcal{P}}$, $\forall j \geq M+1$, $f(s_j) > f^*_{\mathcal{P}}$.

According to the properties of function $\mathcal{T}$, for all $1 \leq j \leq M$, $f(\mathcal{T}(s_j)) = f^*_{\mathcal{P}}$; for all $M+1 \leq j \leq N$, we have

$$\begin{aligned} P\left(f(\mathcal{T}(s_j)|\mathcal{P}) > f^*_{\mathcal{P}}\right) &= P(\mathcal{T}(s_j) \notin \mathcal{X}^*) \\ &\leq P(\forall s \in N(s_j) - \{s_j\}, s \notin \mathcal{X}^*) \\ &= \left(\frac{N-M}{N}\right)^{|N(s_j)|-1} \\ &\leq \left(1 - \frac{M}{N}\right)^{n_{min}-1} \\ &= (1-r)^{n_{min}-1}. \end{aligned} \tag{16}$$

The above derivation illustrates the effect of $\mathcal{T}$ when applied to a non-minimum point $s_j$. It shows that, with a probability of at least $1 - (1-r)^{n_{\min}-1}$, it turns $\mathcal{T}(s_j)$ into a minimum point.

We consider the difference in the expected value of $f(\mathcal{T}(x)|\mathcal{P})$ on the probability distribution $p_\theta$ before and after applying $T$ to $x$. Since $f(\mathcal{T}(s_j)) \leq f(s_j)$ for all $M+1 \leq j \leq N$, we conclude that

$$\begin{aligned} \mathbb{E}_{\mathcal{T}}[\mathbb{E}_{p_\theta} f(\mathcal{T}(x)|\mathcal{P}) - f^*_{\mathcal{P}}] &= \mathbb{E}_{\mathcal{T}}\left[\sum_{j=M+1}^{N} p_\theta(s_j)(f(\mathcal{T}(s_j)|\mathcal{P}) - f^*_{\mathcal{P}})\right] \\ &= \sum_{j=M+1}^{N} p_\theta(s_j)\mathbb{E}_{\mathcal{T}}\left[(f(\mathcal{T}(s_j)|\mathcal{P}) - f^*_{\mathcal{P}})\right] \\ &\leq \sum_{j=M+1}^{N} p_\theta(s_j)(f(s_j) - f^*_{\mathcal{P}})P(f(\mathcal{T}(s_j)|\mathcal{P}) > f^*_{\mathcal{P}}) \\ &\leq (1-r)^{n_{min}-1}\mathbb{E}_{p_\theta}[f(x|\mathcal{P}) - f^*_{\mathcal{P}}]. \end{aligned}$$

Since both the number of summands and the expectation of each term are finite, we can interchange the expectation and summation, hence the equality in the second line holds. The inequality in the fourth line is

due to (16). Notice that $\mathbb{E}_{p_\theta}[f(\mathcal{T}(x)|\mathcal{P}) - f^*_{\mathcal{P}}] \geq 0$, by Markov's inequality, with probability at least $\delta$ over the choice of $\mathcal{T}$,

$$\mathbb{E}_{p_\theta}[f(\mathcal{T}(x)|\mathcal{P}) - f^*_{\mathcal{P}}] \leq \frac{(1-r_{min})^{n_{min}-1}}{1-\delta}\mathbb{E}_{p_\theta}[f(x|\mathcal{P}) - f^*_{\mathcal{P}}].$$

By combining the definition of the loss function, we finish the proof of Lemma D. □

Under the condition of Theorem 2, according to Corollary 1, with a probability of at least $t$, $x \sim p_\theta$ satisfies

$$f^{(s)}(\mathcal{T}(x)|\mathcal{P}) - f^*_{\mathcal{P}} < \frac{\mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P}) - f^*_{\mathcal{P}}}{1-t} \quad \text{and} \quad x \in \mathcal{X}^{(h)},$$

where $\mathcal{X}^*$ is the set of optimal points $x^*$.

According to Lemma D, with a probability of at least $\delta$, it holds that $\frac{\mathcal{L}_{\mathcal{T}}(\theta,\mathcal{P})-f^*_{\mathcal{P}}}{1-t} \leq \frac{(1-r)^{n_{min}-1}(\mathcal{L}(\theta,\mathcal{P})-f^*_{\mathcal{P}})}{(1-t)(1-\delta)}$, where $r = \frac{|\mathcal{X}^*|}{2^n}$. Under Assumption 2, the selection of $x \sim p_\theta$ is independent of $\mathcal{T}$. Therefore, with a probability of at least $t\delta$, both of the above inequalities hold true. This concludes the proof of Theorem 2.

# E Convergence analysis

In this subsection, we provide a detailed analysis of the convergence theory. Initially, we have the following basic assumption, which is the standard assumption in the previous works on the theoretical analysis of policy gradient algorithms.

**Assumption 3.** *For any $x \in \{-1,1\}^n$, $\phi(x;\theta)$ is differentiable with respect to $\theta$. Moreover, there exists constants $M_1, M_2 > 0$ such that, for any $x \in \{-1,1\}^n$,*

1. $\sup_{\theta\in\mathbb{R}^n} \|\nabla_\theta \phi(x;\theta)\| \leq M_1$,
2. $\|\nabla_\theta \phi(x;\theta^1) - \nabla_\theta \phi(x;\theta^2)\| \leq M_2\|\theta^1 - \theta^2\|,\ \forall \theta^1, \theta^2 \in \mathbb{R}^n$.

It is trivial that $f(T(x)|\mathcal{P}_i)$ is bounded uniformly for all $i \in [h]$, and we denote this upper bound as $B(f \circ T)$. As Assumption 3 holds, we have $\|A(x,\theta|\mathcal{P}_i)\nabla_\theta \log p_\theta(x)\| \leq B(f \circ T)M_1$. This indicates that the gradient $\nabla_\theta \mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P}_i)$ defined by (6) is bounded uniformly. Therefore, the variance of $\nabla_\theta \mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P}_i)$ under uniform sampling is bounded, and we denote this bound as $M_3$.

**Definition 1.** $M_3 = \mathrm{Var}_{\mathcal{P}\sim\mathtt{Uniform}(\Gamma)}[\nabla_\theta \mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P})] = \frac{1}{h}\sum_{i=1}^h (\nabla_\theta \mathcal{L}_{\mathcal{T}}(\theta, \mathcal{P}_i) - \nabla_\theta L(\theta))^2$.

Additionally, we estimate the stochastic gradient using MCMC sampling instead of direct sampling. The distinction is that MCMC sampling is neither independent nor unbiased. We first present the following definition related to the finite-state time-homogenous Markov chain.

**Definition 2** (spectral gap)**.** *Let $P$ be the transition matrix of a finite-state time homogenous Markov chain. Then the spectral gap of $P$ is defined as*

$$\gamma(P) := \frac{1 - \max\{|\lambda_2(P)|, |\lambda_{2^n}(P)|\}}{2} \in (0,1],$$

*where $\lambda_i(P)$ is the $i$-th largest eigenvalue of the matrix $P$.*

The spectral gap determines the ergodicity of the corresponding Markov chain. A positive spectral gap guarantees that the samples generated by the MCMC algorithm can approximate the desired distribution. For a non-stationary Markov chain, it takes a long time to make the distribution of its current state sufficiently close to its stationary distribution. Therefore, a limited number of transitions leads to bias in gradient estimation, which can be bounded by the spectral gap.

We warm up the Markov chain and discard the beginning $m_0$ samples, leaving $m$ samples for estimating the stochastic gradient. For a certain problem instance $\mathcal{P}_{i_0}$, the following lemma gives error bounds of the stochastic gradient with sampling:

**Lemma 2.** *Let Assumption 3 hold and $i_0 \in [h]$. $S = \{S^j_{i_0}\}_{1\leq i\leq k, \sim 1\leq j\leq m}$ be the samples generated from the Metropolis-Hastings algorithm for problem instance $\mathcal{P}_{i_0}$. Suppose that the Markov chains have the stationary distribution $p_\theta$ and the initial distribution $\nu$. Let $\gamma = \inf_{i_0\in[h],\theta\in\mathbb{R}} \gamma(P_{i_0,\theta}) \in (0,1]$ be the infimum of spectral*

*gaps of the Markov transition matrices $P_{i_0,\theta}$ for all $i_0$ and $\theta$. Then, the sampling error of the stochastic gradient $\bar{g}(\theta, \mathcal{P}_{i_0}, S_{i_0})$ is given by*

$$\|\nu\bar{g}(\theta, \mathcal{P}_{i_0}, S_{i_0}) - \nabla_\theta \mathcal{L}_\mathcal{T}(\theta, \mathcal{P}_{i_0})\| \leq \frac{\chi M_1(B(f \circ T))}{m\gamma} =: B(m),$$
$$\nu\|\bar{g}(\theta, \mathcal{P}_{i_0}, S_{i_0}) - \nabla_\theta \mathcal{L}_\mathcal{T}(\theta, \mathcal{P}_{i_0})\|^2 \leq \frac{8nC(k)M_1^2(B(f \circ T))^2}{mk\gamma} =: V(m,k),$$

*where* $\chi = \chi(\nu, p_\theta) :=_{p_\theta} \left[\left(\frac{d\nu}{dp_\theta} - 1\right)^2\right]^{1/2}$ *is the square-chi divergence between* $\nu$ *and* $p_\theta$ *and* $C(k) = 1 + k\log(1+\chi)$.

The proof of Lemma 2 is similar to the Lemma 3 in [33]. The bias $B(m)$ has an order of $O\left(\frac{1}{m}\right)$ and the variance $V(m,k)$ has an order of $O\left(\frac{1}{mk}\right)$. The Metropolis-Hastings algorithms ensure enough locality and randomness. These error bounds show how hyper-parameters influence the stochastic gradient.

In Algorithm 1, $\{\mathcal{P}_{i_1}, \mathcal{P}_{i_2}, \cdots, \mathcal{P}_{i_B}\}$ is a randomly selected subset from the training problem set $\Gamma = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_h\}$. Let's denote the index set $\mathcal{I} = \{i_1, i_2, \cdots, i_B\}$. For the convenience of computation, we assume that $\mathcal{I}$ is a subset of $[h]$ with elements that are possibly repeated. Each element of $\mathcal{I}$ is independently and uniformly chosen from $[h]$. The following lemma gives the error bounds of the stochastic gradient over $\Gamma$.

**Lemma 3.** *Let Assumption 3 hold and* $\bar{g}(\theta, \mathcal{I}) = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \bar{g}(\theta, P_i, S_i)$*, the remaining notation is the same as in Lemma 2. Then, the sampling error of* $\bar{g}(\theta) = \bar{g}(\theta, \mathcal{I})$ *is given by*

$$\|\mathbb{E}_\mathcal{I}\nu\bar{g}(\theta, \mathcal{I}) - \nabla_\theta L(\theta)\| \leq B(m),$$
$$\mathbb{E}_\mathcal{I}\nu\|\bar{g}(\theta, \mathcal{I}) - \nabla_\theta L(\theta)\|^2 \leq V(m,k) + 2B(m)\sqrt{\frac{M_3}{|\mathcal{I}|}} + \frac{M_3}{|\mathcal{I}|} = O\left(\frac{1}{mk} + \frac{1}{m\sqrt{|\mathcal{I}|}} + \frac{1}{|\mathcal{I}|}\right),$$

*Proof.* Since $L(\theta) = \frac{1}{h}\sum_{i=1}^h \mathcal{L}_\mathcal{T}(\theta, \mathcal{P}_i)$, thus $\nabla_\theta L(\theta) = \frac{1}{h}\sum_{i=1}^h \nabla_\theta \mathcal{L}_\mathcal{T}(\theta, \mathcal{P}_i)$, the first inequality is trivial according to Lemma 2.

For convenience, we denote that $g_i = \nabla_\theta L(\theta, P_i)$, $g = \nabla_\theta L(\theta)$, $g_\mathcal{I} = \frac{1}{\mathcal{I}}\sum_{i\in\mathcal{I}} g_i$, $\bar{g}_\mathcal{I} = g(\theta, \mathcal{I})$ and $\bar{g}_i = \bar{g}(\theta, P_i, S_i)$. Notice that the variance can be decomposed as follows:

$$\mathbb{E}_\mathcal{I}\nu\|\bar{g}_\mathcal{I} - g\|^2 = \mathcal{I}\|g_\mathcal{I} - g\|^2 + \mathbb{E}_\mathcal{I}\nu\|\bar{g}_\mathcal{I} - g_\mathcal{I}\|^2 + 2\mathbb{E}_\mathcal{I}\nu(\bar{g}_\mathcal{I} - g_\mathcal{I})^T(g_\mathcal{I} - g).$$

According to Lemma 3, along with the simple calculations of mean variance in repeated independent sampling,

$$\mathbb{E}_\mathcal{I}\nu\|\bar{g}_\mathcal{I} - g_\mathcal{I}\|^2 \leq V(m,k), \quad \mathcal{I}\|g_\mathcal{I} - g\|^2 \leq \frac{M_3}{|\mathcal{I}|}.$$

Due to the Cauchy-Schwarz inequality in the sense of expectation, we conclude that

$$\begin{aligned}\mathbb{E}_\mathcal{I}\nu(\bar{g}_\mathcal{I} - g_\mathcal{I})^T(g_\mathcal{I} - g) &\leq \left(\mathbb{E}_\mathcal{I}\nu\|\bar{g}_\mathcal{I} - g_\mathcal{I}\|\right)\left(\mathcal{I}\|g_\mathcal{I} - g\|\right) \\ &\leq B(m)\sqrt{\mathcal{I}\|g_\mathcal{I} - g\|^2} \\ &= B(m)\sqrt{\frac{M_3}{|\mathcal{I}|}}.\end{aligned}$$

By combining estimates of the above three terms, we have completed the proof of the second inequality. □

Under Assumption 3, we can prove the $l$-smoothness of the cost function $L(\theta)$ and perform a standardized analysis in stochastic optimization. The $l$-smoothness of the loss function is shown in the following lemma, whose proof is similar to the Lemma 4 in [33].

**Lemma 4.** *Let Assumption 3 hold. There exists a constant* $l > 0$ *such that* $L(\theta)$ *is* $l$*-smooth, that is*

$$\|\nabla_\theta L(\theta^1) - \nabla_\theta L(\theta^2)\| \leq l\|\theta^1 - \theta^2\|, \ \forall \theta^1, \theta^2 \in \mathbb{R}^d. \tag{17}$$

It follows from (17) that

$$L(\theta^2) \leq L(\theta^1) + \left\langle \nabla_\theta L(\theta^1), \theta^2 - \theta^1 \right\rangle + \frac{l}{2}\|\theta^2 - \theta^1\|^2. \tag{18}$$

In our discussion, Algorithm 1 updates the policy with the stochastic gradient $\bar{g}(\theta)$. We have known the error bound of $\bar{g}(\theta)$ and the $l$-smoothness of the cost function. By analyzing the expected descent in each iteration, we can establish first-order convergence and provide the rate of convergence as follows, which is the full version of Theorem 3 in the main text:

**Theorem 6.** *Let $\{\theta^{(t)}\}$ be generated by Algorithm 1. During the sampling process, we choose $k$ starting points and construct $m$ Markov chains for each of them. If the stepsize satisfies $\eta^t \leq \frac{1}{2L}$, then for any $\tau$, we have*

$$\min_{1\leq t\leq \tau} \|\nabla_\theta L(\theta^{(t)})\|^2 \leq O\left(\frac{1}{\sum_{t=1}^{\tau}\eta^t} + \frac{\sum_{t=1}^{\tau}(\eta^t)^2}{\sum_{t=1}^{\tau}\eta^t}\left(\frac{1}{mk} + \frac{1}{m\sqrt{\mathcal{I}}} + \frac{1}{\mathcal{I}}\right) + \frac{1}{m^2}\right). \tag{19}$$

*In particular, if the stepsize is chosen as $\eta^t = \frac{c\sqrt{mk}}{\sqrt{t}}$ with $c \leq \frac{1}{2l}$, then we have*

$$\min_{1\leq t\leq \tau} \|\nabla_\theta L(\theta^{(t)})\|^2 \leq O\left(\frac{\log\tau}{\sqrt{\tau}}\sqrt{\left(\frac{1}{mk} + \frac{1}{m\sqrt{\mathcal{I}}} + \frac{1}{\mathcal{I}}\right)} + \frac{1}{m^2}\right), \tag{20}$$

*where $B$ is the batch size.*

*Proof.* The MCPG updates with the stochastic gradient as $\theta^{t+1} = \theta^t - \eta^t\bar{g}(\theta^t)$. For convenience, we denote that $L^t = L(\theta^t)$, $g^t = \nabla_\theta L(\theta^t)$ and $\hat{g}^t = \bar{g}(\theta^t)$ for any $t \geq 1$.Using the Lipschitz condition of $L$, it follows from (18) that

$$\begin{aligned} L^{t+1} &\leq L^t + \left\langle g^t, \theta^{t+1} - \theta^t \right\rangle + \frac{l}{2}\|\theta^{t+1} - \theta^t\|^2 \\ &= L^t - \eta^t \left\langle g^t, \hat{g}^t \right\rangle + \frac{\eta^{t2} l}{2}\|\hat{g}^t\|^2 \\ &= L^t - (\eta^t - \eta^{t2} l)\left\langle g^t, \hat{g}^t - g^t \right\rangle - \left(\eta^t - \frac{\eta^{t2} l}{2}\right)\|g^t\|^2 + \frac{\eta^{t2} l}{2}\|\hat{g}^t - g^t\|^2. \end{aligned} \tag{21}$$

Taking the condition expectation of (21) over $\theta^t$, we obtain that

$$\begin{aligned} L^{t+1}|\theta^t \leq & L^t|\theta^t - (\eta^t - \eta^{t2} l)\left\langle g^t, \hat{g}^t|\theta^t - g^t \right\rangle \\ & - \left(\eta^t - \frac{\eta^{t3} l}{2}\right)\|g^t\|^2 + \frac{\eta^{t2} l}{2}\|\hat{g}^t - g^t\|^2|\theta^t. \end{aligned} \tag{22}$$

Using the fact $|\langle x, y\rangle| \leq (\|x\|^2 + \|y\|^2)/2$, we have

$$-(\eta^t - \eta^{t2} l)\left\langle g^t, \hat{g}^t|\theta^t - g^t \right\rangle \leq \frac{\eta^t - \eta^{t2} l}{2}\|g^t\|^2 + \frac{\eta^t}{2}\|\hat{g}^t|\theta^t - g^t\|^2. \tag{23}$$

Therefore, we plug in (23) to (22) and then get

$$\begin{aligned} & 2(L^t|\theta^t - L^{t+1}|\theta^t) \\ \geq & \eta^t\|g^t\|^2 - \eta^t \cdot \|\hat{g}^t|\theta^t - g^t\|^2 - \eta^{t2} l \cdot \|\hat{g}^t - g^t\|^2|\theta^t \\ \geq & \eta^t\|g^t\|^2 - \eta^t \cdot B^2(m) - \eta^{t2} l \cdot \left(V(m,k) + 2B(m)\sqrt{\frac{M_3}{|\mathcal{I}|}} + \frac{M_3}{|\mathcal{I}|}\right), \end{aligned} \tag{24}$$

where the second inequality is due to Lemma 3. Since $L^t|\theta^t = L^t|\theta^{t-1}$, summing (24) over $t = 1, \ldots, \tau$ and taking the total expectation yields

$$\begin{aligned} & \left(\sum_{t=1}^{\tau}\eta^t\right)\min_{1\leq t\leq\tau}\|g^t\|^2 \leq \sum_{t=1}^{\tau}\eta^t \cdot \|g^t\|^2 \\ \leq & 2(L^1 - L^{\tau+1}|\theta^\tau) + \sum_{t=1}^{\tau}\eta^{t2} \cdot l\left(V(m,k) + 2B(m)\sqrt{\frac{M_3}{|\mathcal{I}|}} + \frac{M_3}{|\mathcal{I}|}\right) + \sum_{t=1}^{\tau}\eta^t \cdot B^2(m) \\ \leq & O(1) + \sum_{t=1}^{\tau}\eta^{t2} \cdot O\left(\frac{1}{mk} + \frac{1}{m\sqrt{|\mathcal{I}|}} + \frac{1}{|\mathcal{I}|}\right) + \sum_{t=1}^{\tau}\eta^t \cdot O\left(\frac{1}{m^2}\right). \end{aligned} \tag{25}$$

Dividing both sides by $\sum_{t=1}^{\tau}\eta^t$ gives the inequality (19). □

# F Partial MaxSAT formulation for MIS and maxcut

Consider a graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. Let $x_v$ be a binary decision variable for each vertex $v \in V$, where

$$x_v = \begin{cases} 1, & \text{if vertex } v \text{ is in the independent set } S, \\ -1, & \text{otherwise.} \end{cases}$$

The Minimum Independent Set(MIS) problem can then be formulated as the following binary programming optimization problem:

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v, \\ \text{s.t.} \quad & x_u + x_v \leq 1, \quad \forall (u, v) \in E, \\ & x_v \in \{-1, 1\}, \quad \forall v \in V. \end{aligned}$$

Here, the objective is to minimize the total number of vertices in the independent set. The first constraint ensures that no two adjacent vertices are both in the independent set, and the second constraint specifies that each decision variable is binary.

We reformulate the MIS problem as a partial MaxSAT problem in order to apply our method. For each pair of vertices $u$ and $v$ that are adjacent in the graph ($(u, v) \in E$), add a hard clause to ensure that at least one of the vertices is not in the independent set

$$\neg X_u \vee \neg X_v, \quad \forall (u, v) \in E. \qquad \text{(hard clause)}$$

This ensures that at least one of $u$ or $v$ is not included in the independent set. The soft clauses only contain the variable itself

$$X_v, \quad \forall v \in V.. \qquad \text{(soft clause)}$$

Consider a graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. Consider a graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. Let $x_v$ be a binary decision variable for each vertex $v \in V$, where

$$x_v = \begin{cases} 1, & \text{if vertex } v \text{ is in one part of the cut } C, \\ -1, & \text{otherwise.} \end{cases}$$

The maxcut problem can then be formulated as the following binary programming optimization problem:

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{(u,v) \in E} (1 - x_u x_v), \\ \text{s.t.} \quad & x_u \in \{-1, 1\}, \quad \forall u \in V. \end{aligned}$$

Here, the objective is to maximize the total number of edges in the cut. The constraint ensures that no two adjacent vertices are both on the same side of the cut and the decision variables are binary.

We reformulate the maxcut problem as a partial MaxSAT problem for the application of our method. For each pair of vertices $u$ and $v$ that are adjacent in the graph ($(u, v) \in E$), add two soft clauses:

$$\neg X_u \vee \neg X_v, \quad \forall (u, v) \in E, \qquad \text{(soft clause)}$$

$$X_u \vee X_v, \quad \forall (u, v) \in E. \qquad \text{(soft clause)}$$