Learning Markov Random Fields for Combinatorial Structures via Sampling through Lovász Local Lemma

Nan Jiang[*1], **Yi Gu**[*2], **Yexiang Xue**[1]

[1] Department of Computer Science, Purdue University, USA
[2] Department of Mathematics, Northwestern University, USA
{jiang631, yexiang}@purdue.edu, Yi.Gu@u.northwestern.edu

Abstract

Learning to generate complex combinatorial structures satisfying constraints will have transformative impacts in many application domains. However, it is beyond the capabilities of existing approaches due to the highly intractable nature of the embedded probabilistic inference. Prior works spend most of the training time learning to separate valid from invalid structures but do not learn the inductive biases of valid structures. We develop NEural Lovász Sampler (NELSON), which embeds the sampler through Lovász Local Lemma (LLL) as a fully differentiable neural network layer. Our NELSON-CD embeds this sampler into the contrastive divergence learning process of Markov random fields. NELSON allows us to obtain valid samples from the current model distribution. Contrastive divergence is then applied to separate these samples from those in the training set. NELSON is implemented as a fully differentiable neural net, taking advantage of the parallelism of GPUs. Experimental results on several real-world domains reveal that NELSON learns to generate 100% valid structures, while baselines either time out or cannot ensure validity. NELSON also outperforms other approaches in running time, log-likelihood, and MAP scores.

1 Introduction

In recent years, tremendous progress has been made in generative modeling (Hinton 2002; Tsochantaridis et al. 2005; Goodfellow et al. 2014; Kingma and Welling 2014; Germain et al. 2015; Larochelle and Murray 2011; van den Oord, Kalchbrenner, and Kavukcuoglu 2016; Arjovsky, Chintala, and Bottou 2017; Song and Ermon 2019; Song et al. 2021; Murphy, Weiss, and Jordan 1999; Yedidia, Freeman, and Weiss 2000; Wainwright and Jordan 2006).

Learning a generative model involves increasing the divergence in likelihood scores between the structures in the training set and those structures sampled from the current generative model distribution. While current approaches have achieved successes in *un-structured* domains such as vision or speech, their performance is degraded in the structured domain, because it is already computationally intractable to search for a valid structure in a combinatorial space subject to constraints, not to mention sampling, which has a higher complexity. In fact, when applied in a constrained domain, existing approaches spend most of their training time manipulating the likelihood of invalid structures, but not learning the difference between valid structures inside and outside of the training set. In the meantime, tremendous progress has been made in automated reasoning (Braunstein, Mézard, and Zecchina 2005; Andersen et al. 2007; Chavira, Darwiche, and Jaeger 2006; Van Hentenryck 1989; Gogate and Dechter 2012; Sang, Bearne, and Kautz 2005). Nevertheless, reasoning and learning have been growing independently for a long time. Only recently do ideas emerge exploring the role of reasoning in learning (Kusner, Paige, and Hernández-Lobato 2017; Jin, Barzilay, and Jaakkola 2018; Dai et al. 2018; Hu et al. 2017; Lowd and Domingos 2008; Ding et al. 2021).

The Lovász Local Lemma (LLL) (Erdős and Lovász 1973) is a classic gem in combinatorics, which at a high level, states that there exists a positive probability that none of a series of *bad* events occur, as long as these events are *mostly* independent from one another and are not too likely individually. Recently, Moser and Tardos (2010) came up with an algorithm, which samples from the probability distribution proven to exist by LLL. Guo, Jerrum, and Liu (2019) proved that the algorithmic-LLL is an unbiased sampler if those bad events satisfy the so-called "extreme" condition. The expected running time of the sampler is also shown to be polynomial. As one contribution of this paper, we offer proofs of the two aforementioned results using precise mathematical notations, clarifying a few descriptions not precisely defined in the original proof. While this line of research clearly demonstrates the potential of LLL in generative learning (generating samples that satisfy all hard constraints), it is not clear how to embed LLL-based samplers into learning and no empirical studies have been performed to evaluate LLL-based samplers in machine learning.

In this paper, we develop NEural Lovász Sampler (NELSON), which implements the LLL-based sampler as a fully differentiable neural network. Our NELSON-CD embeds NELSON into the contrastive divergence learning process of Markov Random Fields (MRFs). Embedding LLL-based sampler allows the contrastive learning algorithm to focus on learning the difference between the training data and the valid structures drawn from the current model distribution. Baseline approaches, on the other hand, spend most

[*]These authors contributed equally.

of their training time learning to generate valid structures. In addition, NELSON is fully differentiable, hence allowing for efficient learning harnessing the parallelism of GPUs.

Related to our NELSON are neural-based approaches to solve combinatorial optimization problems (Selsam et al. 2019; Duan et al. 2022; Li, Chen, and Koltun 2018). Machine learning is also used to discover better heuristics (Yolcu and Póczos 2019; Chen and Tian 2019). Reinforcement learning (Karalias and Loukas 2020; Bengio, Lodi, and Prouvost 2021) as well as approaches integrating search with neural nets (Mandi et al. 2020) are found to be effective in solving combinatorial optimization problems as well. Regarding probabilistic inference, there are a rich line of research on MCMC-type sampling (Neal 1993; Dagum and Chavez 1993; Ge, Xu, and Ghahramani 2018) and various versions of belief propagation (Murphy, Weiss, and Jordan 1999; Ihler, Fisher III, and Willsky 2005; Coja-Oghlan, Müller, and Ravelomanana 2020; Ding and Xue 2020). SampleSearch (Gogate and Dechter 2011) integrates importance sampling with constraint-driven search. Probabilistic inference based on hashing and randomization obtains probabilistic guarantees for marginal queries and sampling via querying optimization oracles subject to randomized constraints (Gomes, Sabharwal, and Selman 2006; Ermon et al. 2013b; Achlioptas and Theodoropoulos 2017; Chakraborty, Meel, and Vardi 2013).

We experiment NELSON-CD on learning preferences towards (i) random K-satisfiability solutions (ii) sink-free orientations of un-directed graphs and (iii) vehicle delivery routes. In all these applications, NELSON-CD (i) has the fastest training time due to seamless integration into the learning framework (shown in Tables 1(a), 3(a)). (ii) NELSON generates samples 100% satisfying constraints (shown in Tables 1(b), 3(b)), which facilitates effective contrastive divergence learning. Other baselines either cannot satisfy constraints or time out. (iii) The fast and valid sample generation allows NELSON to obtain the best learning performance (shown in Table 1(c), 2(a,b), 3(c,d)).

Our contributions can be summarized as follows: **(a)** We present NELSON-CD, a contrastive divergence learning algorithm for constrained MRFs driven by sampling through the Lovász Local Lemma (LLL). **(b)** Our LLL-based sampler (NELSON) is implemented as a fully differentiable multi-layer neural net, allowing for end-to-end training on GPUs. **(c)** We offer a mathematically sound proof of the sample distribution and the expected running time of the NELSON algorithm. **(d)** Experimental results reveal the effectiveness of NELSON in (i) learning models with high likelihoods (ii) generating samples 100% satisfying constraints and (iii) having high efficiency in training[1].

2 Preliminaries

Markov Random Fields (MRF) represent a Boltzmann distribution of the discrete variables $X = \{X_i\}_{i=1}^n$ over a Boolean hypercube $\mathcal{X} = \{0,1\}^n$. For $x \in \mathcal{X}$, we have:

$$P_\theta(X = x) = \frac{\exp\left(\phi_\theta(x)\right)}{Z(\theta)} = \frac{\exp\left(\sum_{j=1}^m \phi_{\theta,j}(x_j)\right)}{Z(\theta)}. \tag{1}$$

Here, $Z(\theta) = \sum_{x' \in \mathcal{X}} \exp\left(\phi_\theta(x')\right)$ is the partition function that normalizes the total probability to 1. The potential function is $\phi_\theta(x) = \sum_{j=1}^m \phi_{\theta,j}(x_j)$. Each $\phi_{\theta,j}$ is a *factor potential*, which maps a value assignment over a subset of variables $X_j \subseteq X$ to a real number. We use upper case letters, such as X_j to represent (a set of) random variables, and use lower case letters, such as x_j, to represent its value assignment. We also use $\texttt{var}(\phi_{\theta,j})$ to represent the domain of $\phi_{\theta,j}$, *i.e.*, $\texttt{var}(\phi_{\theta,j}) = X_j$. θ are the parameters to learn.

Constrained MRF is the MRF model subject to a set of hard constraints $\mathcal{C} = \{c_j\}_{j=1}^L$. Here, each constraint c_j limits the value assignments of a subset of variables $\texttt{var}(c_j) \subseteq X$. We write $c_j(x) = 1$ if the assignment x satisfies the constraint c_j and 0 otherwise. Note that x is an assignment to all random variables, but c_j only depends on variables $\texttt{var}(c_j)$. We denote $C(x) = \prod_{j=1}^L c_j(x)$ as the indicator function. Clearly, $C(x) = 1$ if all constraints are satisfied and 0 otherwise. The constrained MRF is:

$$P_\theta(X = x|\mathcal{C}) = \frac{\exp\left(\phi_\theta(x)\right) C(x)}{Z_\mathcal{C}(\theta)}, \tag{2}$$

where $Z_\mathcal{C}(\theta) = \sum_{x' \in \mathcal{X}} \exp\left(\phi_\theta(x)\right) C(x)$ sums over only valid assignments.

Learn Constrained MRF Given a data set $\mathcal{D} = \{x^k\}_{k=1}^N$, where each x^k is a valid assignment that satisfies all constraints, learning can be achieved via maximal likelihood estimation. In other words, we find the optimal parameters θ^* by minimizing the negative log-likelihood $\ell_\mathcal{C}(\theta)$:

$$\begin{aligned} \ell_\mathcal{C}(\theta) &= -\frac{1}{N}\sum_{k=1}^N \log P_\theta(X = x^k|\mathcal{C}) \\ &= -\frac{1}{N}\sum_{k=1}^N \phi_\theta(x^k) + \log Z_\mathcal{C}(\theta). \end{aligned} \tag{3}$$

The parameters θ can be trained using gradient descent: $\theta^{t+1} = \theta^t - \eta \nabla \ell_\mathcal{C}(\theta)$, where η is the learning rate. Let $\nabla \ell_\mathcal{C}(\theta)$ denotes the gradient of the objective $\ell_\mathcal{C}(\theta)$, that is calculated as:

$$\begin{aligned} \nabla \ell_\mathcal{C}(\theta) &= -\frac{1}{N}\sum_{k=1}^N \nabla \phi_\theta(x^k) + \nabla \log Z_\mathcal{C}(\theta) \\ &= -\mathbb{E}_{x \sim \mathcal{D}}\left(\nabla \phi_\theta(x)\right) + \mathbb{E}_{\tilde{x} \sim P_\theta(x|\mathcal{C})}\left(\nabla \phi_\theta(\tilde{x})\right). \end{aligned} \tag{4}$$

The first term is the expectation over all data in training set $\mathcal{D}$. During training, this is approximated using a mini-batch of data randomly drawn from the training set $\mathcal{D}$. The second term is the expectation over the current model distribution $P_\theta(X = x|\mathcal{C})$ (detailed in Appendix C.2). Because learning is achieved following the directions given by the divergence of two expectations, this type of learning is commonly known as contrastive divergence (CD) (Hinton 2002).

[1]Code is at: https://github.com/jiangnanhugo/nelson-cd. Please refer to the Appendix in the extended version (Jiang, Gu, and Xue 2022) for the whole proof and the experimental settings.

Estimating the second expectation is the bottleneck of training because it is computationally intractable to sample from this distribution subject to combinatorial constraints. Our approach, NELSON, leverages the sampling through Lovász Local Lemma to approximate the second term.

Factor Potential in Single Variable Form Our method requires each factor potential $\phi_{\theta,j}(x_j)$ in Eq. (1) to involve only one variable. This is NOT an issue as *all constrained MRF models can be re-written in single variable form* by introducing additional variables and constraints. Our transformation follows the idea in Sang, Bearne, and Kautz (2005). We illustrate the idea by transforming one-factor potential $\phi_{\theta,j}(x_j)$ into the single variable form. First, notice all functions including $\phi_{\theta,j}(x_j)$ over a Boolean hypercube $\{0,1\}^n$ have a (unique) discrete Fourier expansion:

$$\phi_{\theta,j}(x_j) = \sum_{S \in [\texttt{var}(\phi_{\theta,j})]} \hat{\phi}_{\theta,j,S}\, \chi_S(x). \tag{5}$$

Here $\chi_S(x) = \prod_{X_i \in S} X_i$ is the basis function and $\hat{\phi}_{\theta,j,S}$ are Fourier coefficients. $[\texttt{var}(\phi_{\theta,j})]$ denotes the power set of $\texttt{var}(\phi_{\theta,j})$. For example, if $\texttt{var}(\phi_{\theta,j}) = \{X_1, X_2\}$, then $[\texttt{var}(\phi_{\theta,j})] = \{\emptyset, \{X_1\}, \{X_2\}, \{X_1, X_2\}\}$. See Mansour (1994) for details of Fourier transformation. To transform $\phi_{\theta,j}(x_j)$ into single variable form, we introduce a new Boolean variable $\hat{\chi}_S$ for every $\chi_S(x)$. Because $\hat{\chi}_S$ and all X_i's are Boolean, we can use combinatorial constraints to guarantee $\hat{\chi}_S = \prod_{X_i \in S} X_i$. These constraints are incorporated into $\mathcal{C}$. Afterward, $\phi_{\theta,j}(x_j)$ is represented as the sum of several single-variable factors. Notice this transformation is only possible when the MRF is subject to constraints. We offer a detailed example in Appendix C.1 for further explanation. Equipped with this transformation, we assume all $\phi_{\theta,j}(x_j)$ are single variable factors for the rest of the paper.

Extreme Condition The set of constraints $\mathcal{C}$ is called "extremal" if no variable assignment violates two constraints sharing variables, according to Guo, Jerrum, and Liu (2019).

Condition 1. *A set of constraints $\mathcal{C}$ is called extremal if and only if for each pair of constraints $c_i, c_j \in \mathcal{C}$, (i) either their domain variables do not intersect, i.e., $\texttt{var}(c_i) \cap \texttt{var}(c_j) = \emptyset$. (ii) or for all $x \in \mathcal{X}$, $c_i(x) = 1$ or $c_j(x) = 1$.*

3 Sampling Through Lovász Local Lemma

Lovász Local Lemma (LLL) (Erdős and Lovász 1973) is a fundamental method in combinatorics to show the existence of a valid instance that avoids all the bad events, if the occurrences of these events are "mostly" independent and are not very likely to happen individually. Since the occurrence of a bad event is equivalent to the violation of a constraint, we can use the LLL-based sampler to sample from the space of constrained MRFs. To illustrate the idea of LLL-based sampling, we assume the constrained MRF model is given in the single variable form (as discussed in the previous section):

$$P_\theta(X = x|\mathcal{C}) = \frac{\exp\left(\sum_{i=1}^n \theta_i x_i\right) C(x)}{Z_{\mathcal{C}}(\theta)}, \tag{6}$$

where $Z_{\mathcal{C}}(\theta) = \sum_{x' \in \mathcal{X}} \exp\left(\sum_{i=1}^n \theta_i x_i\right) C(x)$.

Algorithm 1: Sampling Through Lovász Local Lemma.

Input: Random variables $X = \{X_i\}_{i=1}^n$; Constraints $\mathcal{C} = \{c_j\}_{j=1}^L$; Parameters of the constrained MRF θ.

1: $x_i \sim \frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$, for $1 \le i \le n$. ▷ initialize
2: **while** $C(x) = 0$ **do**
3: Find all violated constraints $S \subseteq \mathcal{C}$ in x.
4: $x_k \sim \frac{\exp(\theta_k x_k)}{\sum_{x_k \in \{0,1\}} \exp(\theta_k x_k)}$, for $x_k \in \texttt{var}(S)$. ▷ resample

return A valid sample x drawn from $P_\theta(X = x|\mathcal{C})$.

As shown in Algorithm 1, the LLL-based sampler (Guo, Jerrum, and Liu 2019) takes the random variables $X = \{X_i\}_{i=1}^n$, the parameters of constrained MRF θ, and constraints $\mathcal{C} = \{c_j\}_{j=1}^L$ that satisfy Condition 1 as the inputs. In Line 1 of Algorithm 1, the sampler gives an initial random assignment of each variable following its marginal probability: $x_i \sim \frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$, for $1 \le i \le n$. Here we mean that x_i is chosen with probability mass $\frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$. Line 2 of Algorithm 1 checks if the current assignment satisfies all constraints in $\mathcal{C}$. If so, the algorithm terminates. Otherwise, the algorithm finds the set of violated constraints $S = \{c_j | c_j(x) = 0, c_j \in \mathcal{C}\}$ and re-samples related variables $X_k \in \texttt{var}(S)$ using the same marginal probability, *i.e.*, $x_k \sim \frac{\exp(\theta_k x_k)}{\sum_{x_k \in \{0,1\}} \exp(\theta_k x_k)}$. Here $\texttt{var}(S) = \cup_{c_j \in S} \texttt{var}(c_j)$. The algorithm repeatedly samples all those random variables violating constraints until all the constraints are satisfied.

Under Condition 1, Algorithm 1 guarantees each sample is from the constrained MRFs' distribution $P_\theta(X = x|\mathcal{C})$ (in Theorem 1). In Appendix A, we present the detailed proof and clarify the difference to the original descriptive proof (Guo, Jerrum, and Liu 2019).

Theorem 1 (Probability Distribution)**.** *Given random variables $X = \{X_i\}_{i=1}^n$, constraints $\mathcal{C} = \{c_j\}_{j=1}^L$ that satisfy Condition 1, and the parameters of the constrained MRF in the single variable form θ. Upon termination, Algorithm 1 outputs an assignment x that is randomly drawn from the constrained MRF distribution: $x \sim P_\theta(X = x|\mathcal{C})$.*

Sketch of Proof. We first show that in the last round, the probability of obtaining two possible assignments conditioning on all previous rounds in Algorithm 1 has the same ratio as the probability of those two assignments under distribution $P_\theta(X = x|\mathcal{C})$. Then we show when Algorithm 1 ends, the set of all possible outputs is equal to the domain of non-zero probabilities of $P_\theta(X = x|\mathcal{C})$. Thus we conclude the execution of Algorithm 1 produces a sample from $P_\theta(X = x|\mathcal{C})$ because of the identical domain and the match of probability ratios of any two valid assignments. □

The expected running time of Algorithm 1 is determined by the number of rounds of re-sampling. In the uniform case that $\theta_1 = \ldots = \theta_n$, the running time is linear in the size of the constraints $\mathcal{O}(L)$. The running time for the weighted case has a closed form. We leave the details in Appendix B.

4 Neural Lovász Sampler

We first present the proposed Neural Lovász Sampler (NELSON) that implements the LLL-based sampler as a neural network, allowing us to draw multiple samples in parallel on GPUs. We then demonstrate how NELSON is embedded in CD-based learning for constrained MRFs.

4.1 NELSON: Neural Lovász Sampler

Represent Constraints as CNF NELSON obtains samples from the constrained MRF model in single variable form (Eq. 6). To simplify notations, we denote $P_\theta(X_i = x_i) = \frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$. Since our constrained MRF model is defined on the Boolean hyper-cube $\{0,1\}^n$, we assume all constraints $\{c_j\}_{j=1}^L$ are given in the Conjunctive Normal Form (CNF). Note that all propositional logic can be reformulated in CNF format with at most a polynomial-size increase. A formula represented in CNF is a conjunction ($\wedge$) of clauses. A clause is a disjunction ($\vee$) of literals, and a literal is either a variable or its negation ($\neg$). Mathematically, we use c_j to denote a clause and use $l_{j,k}$ to denote a literal. In this case, a CNF formula would be:

$$c_1 \wedge \ldots \wedge c_L, \quad \text{where } c_j = l_{j,1} \vee \ldots \vee l_{j,K} \tag{7}$$

A clause is true if and only if at least one of the literals in the clause is true. The whole CNF is true if all clauses are true.

We transform each step of Algorithm 1 into arithmetic operations, hence encoding it as a multi-layer neural network. To do that, we first need to define a few notations:

• Vector of assignment $x^t = (x_1^t, \ldots, x_n^t)$, where x_i^t is the assignment of variable X_i in the t-th round of Algorithm 1. $x_i^t = 1$ denotes variable X_i takes value 1 (or true).

• Vector of marginal probabilities $P = (P_1, \ldots, P_n)$, where P_i is the probability of variable X_i taking value 0 (false): $P_i = P_\theta(X_i = 0) = \exp(0)/(\exp(0) + \exp(\theta_i))$.

• Tensor $W \in \{-1, 0, 1\}^{L \times K \times n}$ and matrix $b \in \{0,1\}^{L \times n}$, that are used for checking constraint satisfaction:

$$W_{jki} = \begin{cases} 1 & \text{if } k\text{-th literal of clause } c_j \text{ is } X_i, \\ -1 & \text{if } k\text{-th literal of clause } c_j \text{ is } \neg X_i, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

$$b_{jk} = \begin{cases} 1 & \text{if } k\text{-th literal of clause } c_j \text{ is negated,} \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

• Matrix $V \in \{0,1\}^{L \times n}$, denoting the mapping from clauses to variables in the CNF form for constraints $\mathcal{C}$:

$$V_{ji} = \begin{cases} 1 & \text{if clause } c_j \text{ contains a literal involving } X_i \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

• Vector of resampling indicators A^t, where $A_i^t = 1$ indicates variable X_i needs to be resampled at round t.

Given these defined variables, we represent each step of Algorithm 1 using arithmetic operations as follows:

Initialization To complete line 1 of Algorithm 1, given the marginal probability vector P, the first step is sampling an initial assignment of X, $x^1 = (x_1^1, \ldots, x_n^1)$. It is accomplished by: for $1 \le i \le n$,

$$x_i^1 = \begin{cases} 1 & \text{if } u_i > P_i, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Here u_i is sampled from the uniform distribution in $[0, 1]$.

Check Constraint Satisfaction To complete line 2 of Algorithm 1, given an assignment x^t at round $t \ge 1$, tensor W and matrix b, we compute Z^t as follows:

$$Z^t = W \circledast x^t + b, \tag{12}$$

where $\circledast$ represents a special multiplication between tensor and vector: $(W \circledast x)_{jk} = \sum_{i=1}^n W_{jki} x_i^t$. Note that $Z_{jk}^t = 1$ indicates the k-th literal of j-th clause is true (takes value 1). Hence, we compute S_j^t as:

$$S_j^t = 1 - \max_{1 \le k \le K} Z_{jk}, \quad \text{for } 1 \le j \le L. \tag{13}$$

Here $S_j^t = 1$ indicates x^t violates j-th clause. We check $\sum_{j=1}^L S_j^t \neq 0$ to see if any clause is violated, which corresponds to $C(x) = 0$ and is the continuation criteria of the while loop.

Extract Variables in Violated Clauses To complete line 3 of Algorithm 1, we extract all the variables that require resampling based on vector S^t computed from the last step. The vector of resampling indicator A^t can be computed as:

$$A_i^t = \mathbf{1}\left(\sum_{j=1}^L S_j^t V_{ji} \ge 1\right), \quad \text{for } 1 \le i \le n \tag{14}$$

where $\sum_{j=1}^L S_j^t V_{ji} \ge 1$ implies X_i requires resampling.

Resample To complete line 4 of Algorithm 1, given the marginal probability vector P, resample indicator vector A^t and assignment x^t, we draw a new random sample x^{t+1}. This can be done using this update rule: for $1 \le i \le n$,

$$x_i^{t+1} = \begin{cases} (1 - A_i^t) x_i^t + A_i^t & \text{if } u_i > P_i, \\ (1 - A_i^t) x_i^t & \text{otherwise.} \end{cases} \tag{15}$$

Again, u_i is drawn from the uniform distribution in $[0, 1]$. Drawing multiple assignments in parallel is attained by extending x^t with a new dimension (See implementation in Appendix D.1). Example 1 show the detailed steps of NELSON (See more examples in Appendix A.5).

Example 1. *Assume we have random variables X_1, X_2, X_3 with $n = 3$, Constraints $\mathcal{C} = (X_1 \vee X_2) \wedge (\neg X_1 \vee X_3)$ in the CNF form with $L = 2, K = 2$. Tensor W is:*

$$W = \begin{bmatrix} w_{11} = [w_{111}, w_{112}, w_{113}], & w_{12} = [w_{121}, w_{122}, w_{123}] \\ w_{21} = [w_{211}, w_{212}, w_{213}], & w_{22} = [w_{221}, w_{222}, w_{223}] \end{bmatrix},$$

$w_{11} = [1, 0, 0], w_{12} = [0, 1, 0], w_{21} = [-1, 0, 0], w_{22} = [0, 0, 1]$.

Algorithm 2: Learn Constrained MRFs via NELSON-CD.

Input: Dataset $\mathcal{D}$; Constraints $\mathcal{C}$; #Samples m; Learning Iterations $T_{\max}$; Parameters of Constrained MRFs θ.

1: NELSON$(W, b, V) \leftarrow$ build$(X, \mathcal{C})$. ▷ in Sec. 4
2: **for** $t = 1$ **to** $T_{\max}$ **do**
3: $\{x^j\}_{j=1}^m \sim \mathcal{D}$. ▷ from data
4: $\{\tilde{x}^j\}_{j=1}^m \leftarrow$ NELSON(θ^t, m). ▷ from model
5: $g^t \leftarrow \frac{1}{m}\sum_{j=1}^m \nabla\phi(x^j) - \nabla\phi(\tilde{x}^j)$ ▷ divergence
6: $\theta^{t+1} \leftarrow \theta^t - \eta g^t$. ▷ update parameters
return The converged MRF model $\theta^{T_{\max}}$.

Note that $w_{111} = 1$ means X_1 is the 1st literal in the 1st clause and $w_{211} = -1$ means $\neg X_1$ is the 1st literal in the 2nd clause. Matrix b and the mapping matrix V are:

$$b = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},$$

$b_{21} = 1$ indicates the 1st literal in the 2nd clause is negated. For the mapping matrix, $V_{11} = V_{12} = 1$ implies the 1st clause contains X_1 and X_2. For $t = 1$, suppose we have an initialized assignment $x^1 = [0\ 0\ 1]^\top$, meaning $X_1 = X_2 = 0, X_3 = 1$. The intermediate results of Z^1, S^1, A^1 become:

$$Z^1 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \quad S^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad A^1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},$$

where $S_1^1 = 1$ implies the 1st clause is violated. $A_1^1 = A_2^1 = 1$ denotes variables X_1, X_2 require resampling.

4.2 Contrastive Divergence-based Learning

The whole learning procedure is shown in Algorithm 2. At every learning iteration, we call NELSON to draw assignments $\{\tilde{x}^j\}_{j=1}^m$ from constrained MRF's distribution $P_\theta(X|\mathcal{C})$. Then we pick m data points at random from the training set $\{x^j\}_{j=1}^m \sim \mathcal{D}$. The divergence g^t in line 5 of Algorithm 2 is an estimation of $\nabla\ell_{\mathcal{C}}(\theta)$ in Eq. (4). Afterward, the MRFs' parameters are updated, according to line 6 of Algorithm 2. After $T_{\max}$ learning iterations, the algorithm outputs the constrained MRF model with parameters $\theta^{T_{\max}}$.

5 Experiments

We show the efficiency of the proposed NELSON on learning MRFs defined on the solutions of three combinatorial problems. Over all the tasks, we demonstrate that NELSON outperforms baselines on learning performance, i.e., generating structures with high likelihoods and MAP@10 scores (Table 1(c), 2(a,b), 3(c,d)). NELSON also generates samples which 100% satisfy constraints (Tables 1(b), 3(b)). Finally, NELSON is the most efficient sampler. Baselines either time out or cannot generate valid structures (Tables 1(a), 3(a)).

5.1 Experimental Settings

Baselines We compare NELSON with other contrastive divergence learning algorithms equipped with other sampling approaches. In terms of baseline samplers, we consider:

- Gibbs sampler (Carter and Kohn 1994), which is a special case of MCMC that is widely used in training MRF models.
- Weighted SAT samplers, including WAPS (Gupta et al. 2019), WeightGen (Chakraborty et al. 2014) and XOR sampler (Ermon et al. 2013a; Ding and Xue 2021).
- Uniform SAT samplers, including CMSGen (Golia et al. 2021), QuickSampler (Dutra et al. 2018), UniGen (Soos, Gocht, and Meel 2020) and KUS (Sharma et al. 2018). Notice these samplers cannot sample SAT solutions from a non-uniform distribution. We include them in the learning experiments as a comparison, and exclude them in the weighted sampling experiment (in Fig. 2).

Metrics In terms of evaluation metrics, we consider:

- Training time per iteration, which computes the average time for every learning method to finish one iteration.
- Validness, that is the percentage of generated solutions that satisfy the given constraints $\mathcal{C}$.
- Mean Averaged Precision (MAP@10), which is the percentage that the solutions in the training set $\mathcal{D}$ reside among the top-10 *w.r.t.* likelihood score. The higher the MAP@10 scores, the better the model generates structures closely resembling those in the training set.
- log-likelihood of the solutions in the training set $\mathcal{D}$ (in Eq. 3). The model that attains the highest log-likelihood learns the closest distribution to the training set.
- Approximation error of $\nabla \log Z_{\mathcal{C}}(\theta)$, which is the L_1 distance between the exact value $\nabla \log Z_{\mathcal{C}}(\theta)$ and the approximated value given by the sampler.

See Appendix D for detailed settings of baselines and evaluation metrics, as well as the following task definition, dataset construction, and potential function definition.

5.2 Random K-SAT Solutions with Preference

Task Definition & Dataset This task is to learn to generate solutions to a K-SAT problem. We are given a training set $\mathcal{D}$ containing solutions to a corresponding CNF formula $c_1 \wedge \ldots \wedge c_L$. Note that not all solutions are equally likely to be presented in $\mathcal{D}$. The *learning* task is to maximize the log-likelihood of the assignments seen in the training set $\mathcal{D}$. Once learning is completed, the *inference* task is to generate valid solutions that closely resemble those in $\mathcal{D}$ (Dodaro and Previti 2019). To generate the training set $\mathcal{D}$, we use CNFGen (Lauria et al. 2017) to generate the random K-SAT problem and use Glucose4 solver to generate random valid solutions (Ignatiev, Morgado, and Marques-Silva 2018).

Sampler's Efficiency and Accuracy Table 1 shows the proposed NELSON is an efficient sampler that generates valid assignments, in terms of the training time for learning constrained MRF, approximation error for the gradient and validness of the generated assignments. In Table 1(a), NELSON takes much less time for sampling against all the samplers and can train the model with the dataset of problem size 1000 within an hour. In Table 1(b), NELSON always generates valid samples. The performance of QuickSampler and Gibbs methods decreases when the problem size becomes larger. In Table 1(c), NELSON, XOR and WAPS are

Table 1: Sampling efficiency and accuracy for learning K-SAT solutions with preferences. The proposed NELSON is the most efficient (see "Training Time Per Epoch") and always generates valid assignments (see "Validness") with a small approximation error (see "Approximation Error of Gradient") against all baselines. T.O. means time out.

Problem size	NELSON	XOR	WAPS	WeightGen	CMSGen	KUS	QuickSampler	Unigen	Gibbs
	(a) Training time per iteration (Mins) (↓)								
10	**0.13**	26.30	1.75	0.64	0.22	0.72	0.40	0.66	0.86
20	**0.15**	134.50	3.04	T.O.	0.26	0.90	0.30	2.12	1.72
30	**0.19**	1102.95	6.62	T.O.	0.28	2.24	0.32	4.72	2.77
40	**0.23**	T.O.	33.70	T.O.	0.31	19.77	0.39	9.38	3.93
50	**0.24**	T.O.	909.18	T.O.	0.33	1532.22	0.37	13.29	5.27
500	**5.99**	T.O.	T.O.	T.O.	34.17	T.O.	T.O.	T.O.	221.83
1000	**34.01**	T.O.	T.O.	T.O.	177.39	T.O.	T.O.	T.O.	854.59
	(b) Validness of generated solutions (%) (↑)								
$10-50$	**100**	**100**	**100**	**100**	**100**	**100**	82.65	**100**	90.58
500	**100**	T.O.	T.O.	T.O.	**100**	T.O.	7.42	**100**	54.27
1000	**100**	T.O.	T.O.	T.O.	**100**	T.O.	0.00	**100**	33.91
	(c) Approximation error of $\nabla \log Z_{\mathcal{C}}(\theta)$ (↓)								
10	**0.10**	0.21	0.12	3.58	3.96	4.08	3.93	4.16	0.69
12	**0.14**	0.19	0.16	5.58	5.50	5.49	5.55	5.48	0.75
14	**0.15**	0.25	0.19	T.O.	6.55	6.24	7.79	6.34	1.30
16	0.16	0.25	**0.15**	T.O.	9.08	9.05	9.35	9.03	1.67
18	**0.18**	0.30	0.23	T.O.	10.44	10.30	11.73	10.20	1.90

Table 2: The quality of learning outcomes for learning random K-SAT solutions with preferences. NELSON achieves the best likelihood and MAP@10 scores. T.O. is time out.

Problem size	NELSON	Gibbs	CMSGen	Quicksampler WeightGen,KUS XOR, WAPS
	(a) log-likelihood (↑)			
100	-49.16	$\mathbf{-36.36}$	-60.12	T.O.
300	$\mathbf{-52.61}$	-53.11	-128.39	
500	$\mathbf{-196.47}$	-197.21	-272.49	
700	$\mathbf{-238.60}$	-238.75	-389.44	
1000	$\mathbf{-294.22}$	-296.33	-532.85	
	(b) MAP@10 (%) (↑)			
100	82.13	83.32	**86.34**	T.O.
300	**66.37**	64.42	64.50	
500	**90.03**	73.14	70.67	
700	**69.74**	**69.74**	48.10	
1000	**91.70**	77.56	78.72	

the three algorithms that can effectively estimate the gradient while the other algorithms incur huge estimation errors. Also, the rest methods are much slower than NELSON.

Learning Quality Table 2 demonstrates NELSON-CD learns a more accurate constrained MRF model by measuring the log-likelihood and MAP@10 scores. Note that baselines including Quicksampler, Weightgen, KUS, XOR and WAPS timed out for the problem sizes we considered. Compared with the remaining baselines, NELSON attains the best log-likelihood and MAP@10 metric.

Abalation Study We also evaluated the samplers' efficiency in isolation (not embedded in learning). The sampling cases we considered are uniform and weighted (mainly following the experiment setting in Chakraborty and Meel (2019)). In weighted sampling, the weights are specified

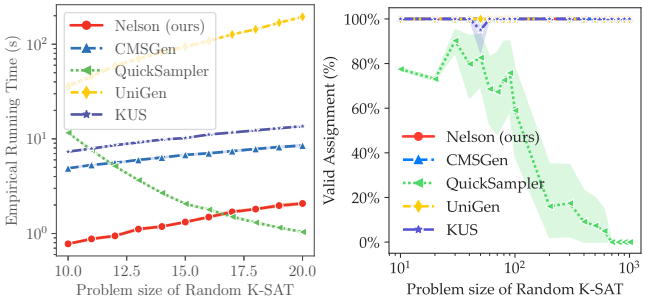


Figure 1: Running time and the percentage of valid structures sampled uniformly at random from solutions of K-SAT problems. Among all the problem sizes, NELSON always generate valid solutions and is the most efficient sampler.

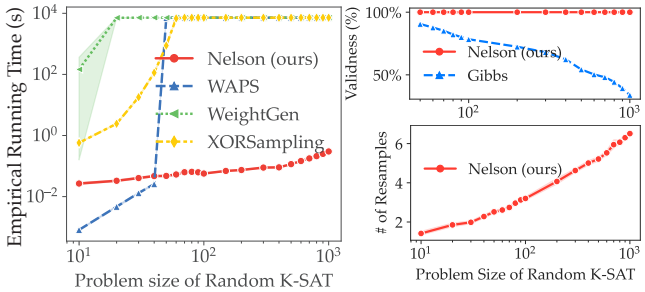


Figure 2: Running time, the percentage of valid solutions generated, and rounds of resampling for weighted sample generation of K-SAT solutions. Among all the problem sizes, NELSON scales the best among all approaches and always generates valid solutions.

Table 3: Sample efficiency and learning performance of the sink-free orientation task. NELSON is the most efficient (see Training Time Per Epoch) and always generates valid assignments (see Validness), has the smallest error approximating gradients, and has the best learning performance (see MAP@10) among all baselines.

Problem size	(a) Training Time Per Epoch (Mins) (↓) NELSON	Gibbs	CMSGen
10	**0.53**	9.85	0.69
20	**0.53**	80.12	1.93
30	**0.72**	256.38	3.65
40	**0.93**	777.01	5.99
50	**1.17**	T.O.	9.08
	(b) Validness of Orientations (%) (↑)		
7	**100**	50.16	**100**
8	**100**	64.63	**100**
9	**100**	47.20	**100**
10	**100**	62.60	**100**
11	**100**	84.95	**100**
	(c) Approximation Error of $\nabla \log Z_{\mathcal{C}}(\theta)$ (↓)		
5	**0.01**	0.09	0.21
7	**0.05**	0.08	2.37
9	**0.03**	0.11	2.37
11	**0.04**	0.17	8.62
13	**0.05**	0.28	11.27
	(d) MAP@10 (%) (↑)		
10	61.14	60.01	**64.56**
20	**55.26**	55.20	47.79
30	**100.00**	96.29	**100.00**
40	**40.01**	39.88	38.90
50	**46.12**	T.O.	42.11

by fixed values to the single factors in Eq. (6). In the uniform sampling case in Fig. 1, NELSON and Quicksampler require much less time to draw samples compared to other approaches. However, the solutions generated by Quicksampler rarely satisfy constraints. In the weighted sampling case in Fig. 2, NELSON scales better than all the competing samplers as the sizes of the K-SAT problems increase.

5.3 Sink-Free Orientation in Undirected Graphs

Task Definition & Dataset A *sink-free* orientation of an undirected graph is a choice of orientation for each arc such that every vertex has at least one outgoing arc (Cohn, Pemantle, and Propp 2002). This task has wide applications in robotics routing and IoT network configuration (Takahashi et al. 2009). Even though finding a sink-free orientation is tractable, sampling a sink-free orientation from the space of all orientations is still #P-hard. Given a training set of preferred orientations $\mathcal{D}$ for the graph, the *learning* task is to maximize the log-likelihood of the orientations seen in the training set. The *inference* task is to generate valid orientations that resemble those in the training set. To generate the training set, we use the Erdős-Rényi random graph from the NetworkX library. The problem size is characterized by the number of vertices in the graph. The baselines we consider are CD-based learning with Gibbs sampling and CMSGen.

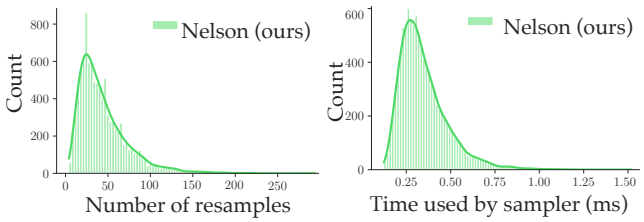


Figure 3: Frequency histograms for the number of resample and the total time of NELSON method for uniformly sampling visiting paths for vehicle routing problem.

Learning Quality In Table 3(a), we show the proposed NELSON method takes much less time to train MRF for one epoch than the competing approaches. Furthermore, in Table 3(b), NELSON and CMSGen generate 100% valid orientations of the graph while the Gibbs-based model does not. Note the constraints for this task satisfy Condition 1, hence NELSON sampler's performance is guaranteed by Theorem 1. In Table 3(c), NELSON attains the smallest approximation error for the gradient (in Eq. 4) compared to baselines. Finally, NELSON learns a higher MAP@10 than CMSGen. The Gibbs-based approach times out for problem sizes larger than 40. In summary, our NELSON is the best-performing algorithm for this task.

5.4 Learn Vehicle Delivery Routes

Task Definition & Dataset Given a set of locations to visit, the task is to generate a sequence to visit these locations in which each location is visited once and only once and the sequence closely resembles the trend presented in the training data. The training data are such routes collected in the past. The dataset is constructed from TSPLIB, which consists of 29 cities in Bavaria, Germany. The constraints for this problem do not satisfy Condition 1. We still apply the proposed method to evaluate if the NELSON algorithm can handle those general hard constraints.

In Fig. 3, we see NELSON can obtain samples of this delivery problem efficiently. We measure the number of resamples taken as well as the corresponding time used by the NELSON method. NELSON takes roughly 50 times of resamples with an average time of 0.3 seconds to draw a batch (the batch size is 100) of valid visiting sequences.

6 Conclusion

In this research, we present NELSON, which embeds a sampler based on Lovász Local Lemma into the contrastive divergence learning of Markov random fields. The embedding is fully differentiable. This approach allows us to learn generative models over constrained domains, which presents significant challenges to other state-of-the-art models. We also give sound proofs of the performance of the LLL-based sampler. Experimental results on several real-world domains reveal that NELSON learns to generate 100% valid structures, while baselines either time out or cannot generate valid structures. NELSON also outperforms other approaches in the running times and in various learning metrics.

Acknowledgments

We thank all the reviewers for their constructive comments. This research was supported by NSF grants IIS-1850243, CCF-1918327.

References

Achlioptas, D.; and Theodoropoulos, P. 2017. Probabilistic Model Counting with Short XORs. In *SAT*, volume 10491 of *Lecture Notes in Computer Science*, 3–19. Springer.

Andersen, H. R.; Hadzic, T.; Hooker, J. N.; and Tiedemann, P. 2007. A Constraint Store Based on Multivalued Decision Diagrams. In *CP*, volume 4741 of *Lecture Notes in Computer Science*, 118–132. Springer.

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein Generative Adversarial Networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, 214–223. PMLR.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.*, 290(2): 405–421.

Braunstein, A.; Mézard, M.; and Zecchina, R. 2005. Survey propagation: an algorithm for satisfiability. *Random Struct. Algorithms*, 27: 201–226.

Carter, C. K.; and Kohn, R. 1994. On Gibbs sampling for state space models. *Biometrika*, 81(3): 541–553.

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S. A.; and Vardi, M. Y. 2014. Distribution-Aware Sampling and Weighted Model Counting for SAT. In *AAAI*, 1722–1730. AAAI Press.

Chakraborty, S.; and Meel, K. S. 2019. On Testing of Uniform Samplers. In *AAAI*, 7777–7784.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A Scalable and Nearly Uniform Generator of SAT Witnesses. In *CAV*, volume 8044, 608–623. Springer.

Chavira, M.; Darwiche, A.; and Jaeger, M. 2006. Compiling relational Bayesian networks for exact inference. *Int. J. Approx. Reason.*, 42(1-2): 4–20.

Chen, X.; and Tian, Y. 2019. Learning to Perform Local Rewriting for Combinatorial Optimization. In *NeurIPS*, 6278–6289.

Cohn, H.; Pemantle, R.; and Propp, J. G. 2002. Generating a Random Sink-free Orientation in Quadratic Time. *Electron. J. Comb.*, 9(1).

Coja-Oghlan, A.; Müller, N.; and Ravelomanana, J. B. 2020. Belief Propagation on the random k-SAT model. arXiv:2011.02303.

Dagum, P.; and Chavez, R. M. 1993. Approximating Probabilistic Inference in Bayesian Belief Networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(3): 246–255.

Dai, H.; Tian, Y.; Dai, B.; Skiena, S.; and Song, L. 2018. Syntax-Directed Variational Autoencoder for Structured Data. In *ICLR (Poster)*. OpenReview.net.

Ding, F.; Ma, J.; Xu, J.; and Xue, Y. 2021. XOR-CD: Linearly Convergent Constrained Structure Generation. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, 2728–2738. PMLR.

Ding, F.; and Xue, Y. 2020. Contrastive Divergence Learning with Chained Belief Propagation. In *PGM*, volume 138 of *Proceedings of Machine Learning Research*, 161–172. PMLR.

Ding, F.; and Xue, Y. 2021. XOR-SGD: provable convex stochastic optimization for decision-making under uncertainty. In *UAI*, volume 161 of *Proceedings of Machine Learning Research*, 151–160. AUAI Press.

Dodaro, C.; and Previti, A. 2019. Minipref: A Tool for Preferences in SAT (short paper). In *RCRA + RiCeRcA*, volume 2538 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Duan, H.; Vaezipoor, P.; Paulus, M. B.; Ruan, Y.; and Maddison, C. J. 2022. Augment with Care: Contrastive Learning for Combinatorial Problems. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, 5627–5642. PMLR.

Dutra, R.; Laeufer, K.; Bachrach, J.; and Sen, K. 2018. Efficient sampling of SAT solutions for testing. In *ICSE*, 549–559. ACM.

Erdős, P.; and Lovász, L. 1973. Problems and results on 3-chromatic hypergraphs and some related questions. In *Colloquia Mathematica Societatis Janos Bolyai 10. Infinite and Finite Sets, Keszthely (Hungary)*. Citeseer.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013a. Embed and Project: Discrete Sampling with Universal Hashing. In *NIPS*, 2085–2093.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013b. Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization. In *ICML (2)*, volume 28 of *JMLR Workshop and Conference Proceedings*, 334–342. JMLR.org.

Fichte, J. K.; Hecher, M.; and Zisser, M. 2019. An Improved GPU-Based SAT Model Counter. In *CP*, 491–509.

Ge, H.; Xu, K.; and Ghahramani, Z. 2018. Turing: A Language for Flexible Probabilistic Inference. In *AISTATS*, volume 84, 1682–1690. PMLR.

Germain, M.; Gregor, K.; Murray, I.; and Larochelle, H. 2015. MADE: Masked Autoencoder for Distribution Estimation. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, 881–889. JMLR.org.

Gogate, V.; and Dechter, R. 2011. SampleSearch: Importance sampling in presence of determinism. *Artif. Intell.*, 175(2): 694–729.

Gogate, V.; and Dechter, R. 2012. Importance sampling-based estimation over AND/OR search spaces for graphical models. *Artificial Intelligence*, 184-185: 38 – 77.

Golia, P.; Soos, M.; Chakraborty, S.; and Meel, K. S. 2021. Designing Samplers is Easy: The Boon of Testers. In *FMCAD*, 222–230. IEEE.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Near-Uniform Sampling of Combinatorial Spaces Using XOR Constraints. In *NIPS*, 481–488. MIT Press.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A. C.; and Bengio, Y. 2014. Generative Adversarial Nets. In *NIPS*, 2672–2680.

Guo, H.; Jerrum, M.; and Liu, J. 2019. Uniform Sampling Through the Lovász Local Lemma. *J. ACM*, 66(3): 18:1–18:31.

Gupta, R.; Sharma, S.; Roy, S.; and Meel, K. S. 2019. WAPS: Weighted and Projected Sampling. In *TACAS*, volume 11427, 59–76.

Hinton, G. E. 2002. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Comput.*, 14(8): 1771–1800.

Hu, Z.; Yang, Z.; Liang, X.; Salakhutdinov, R.; and Xing, E. P. 2017. Toward Controlled Generation of Text. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, 1587–1596. PMLR.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, volume 10929 of *Lecture Notes in Computer Science*, 428–437. Springer.

Ihler, A. T.; Fisher III, J. W.; and Willsky, A. S. 2005. Loopy Belief Propagation: Convergence and Effects of Message Errors. *J. Mach. Learn. Res.*, 6: 905–936.

Jerrum, M. 2021. Fundamentals of Partial Rejection Sampling. arXiv:2106.07744.

Jiang, N.; Gu, Y.; and Xue, Y. 2022. Learning Markov Random Fields for Combinatorial Structures with Sampling through Lovász Local Lemma. arXiv:2212.00296.

Jin, W.; Barzilay, R.; and Jaakkola, T. S. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 2328–2337. PMLR.

Karalias, N.; and Loukas, A. 2020. Erdos Goes Neural: an Unsupervised Learning Framework for Combinatorial Optimization on Graphs. In *NeurIPS*, 6659–6672.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

Kusner, M. J.; Paige, B.; and Hernández-Lobato, J. M. 2017. Grammar Variational Autoencoder. In *ICML*, volume 70, 1945–1954. PMLR.

Larochelle, H.; and Murray, I. 2011. The Neural Autoregressive Distribution Estimator. In *AISTATS*, volume 15 of *JMLR Proceedings*, 29–37. JMLR.org.

Lauria, M.; Elffers, J.; Nordström, J.; and Vinyals, M. 2017. CNFgen: A Generator of Crafted Benchmarks. In *SAT*, volume 10491, 464–473. Springer.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. In *NeurIPS*, 537–546.

Lowd, D.; and Domingos, P. M. 2008. Learning Arithmetic Circuits. In *UAI*, 383–392. AUAI Press.

Mahmoud, M. 2022. *GPU Enabled Automated Reasoning*. Ph.D. thesis, Mathematics and Computer Science.

Mandi, J.; Demirovic, E.; Stuckey, P. J.; and Guns, T. 2020. Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems. In *AAAI*, 1603–1610. AAAI Press.

Mansour, Y. 1994. Learning Boolean functions via the Fourier transform. In *Theoretical advances in neural computation and learning*, 391–424. Springer.

Moser, R. A.; and Tardos, G. 2010. A constructive proof of the general lovász local lemma. *J. ACM*, 57(2): 11:1–11:15.

Murphy, K. P.; Weiss, Y.; and Jordan, M. I. 1999. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *UAI*, 467–475. Morgan Kaufmann.

Neal, R. M. 1993. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, ON, Canada.

Prevot, N.; Soos, M.; and Meel, K. S. 2021. Leveraging GPUs for Effective Clause Sharing in Parallel SAT Solving. In *SAT*, 471–487.

Rosa, E. D.; Giunchiglia, E.; and O'Sullivan, B. 2011. Optimal stopping methods for finding high quality solutions to satisfiability problems with preferences. In *SAC*, 901–906. ACM.

Sang, T.; Bearne, P.; and Kautz, H. 2005. Performing Bayesian Inference by Weighted Model Counting. In *AAAI*, AAAI'05, 475–481.

Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill, D. L. 2019. Learning a SAT Solver from Single-Bit Supervision. In *ICLR (Poster)*. OpenReview.net.

Sharma, S.; Gupta, R.; Roy, S.; and Meel, K. S. 2018. Knowledge Compilation meets Uniform Sampling. In *LPAR*, volume 57 of *EPiC Series in Computing*, 620–636.

Song, Y.; and Ermon, S. 2019. Generative Modeling by Estimating Gradients of the Data Distribution. In *NeurIPS*, 11895–11907.

Song, Y.; Sohl-Dickstein, J.; Kingma, D. P.; Kumar, A.; Ermon, S.; and Poole, B. 2021. Score-Based Generative Modeling through Stochastic Differential Equations. In *ICLR*. OpenReview.net.

Soos, M.; Gocht, S.; and Meel, K. S. 2020. Tinted, Detached, and Lazy CNF-XOR Solving and Its Applications to Counting and Sampling. In *CAV (1)*, volume 12224 of *Lecture Notes in Computer Science*, 463–484. Springer.

Takahashi, J.; Yamaguchi, T.; Sekiyama, K.; and Fukuda, T. 2009. Communication timing control and topology reconfiguration of a sink-free meshed sensor network with mobile robots. *IEEE/ASME transactions on mechatronics*, 14(2): 187–197.

Tsochantaridis, I.; Joachims, T.; Hofmann, T.; and Altun, Y. 2005. Large Margin Methods for Structured and Interdependent Output Variables. *J. Mach. Learn. Res.*, 6: 1453–1484.

van den Oord, A.; Kalchbrenner, N.; and Kavukcuoglu, K. 2016. Pixel Recurrent Neural Networks. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, 1747–1756. JMLR.org.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*. Cambridge, MA, USA: MIT Press. ISBN 0-262-08181-4.

Wainwright, M. J.; and Jordan, M. I. 2006. Log-determinant relaxation for approximate inference in discrete Markov random fields. *IEEE Trans. Signal Process.*, 54(6-1): 2099–2109.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2000. Generalized Belief Propagation. In *NIPS*, 689–695. MIT Press.

Yolcu, E.; and Póczos, B. 2019. Learning Local Search Heuristics for Boolean Satisfiability. In *NeurIPS*, 7990–8001.

A Probability Distribution of Algorithm 1

A.1 Definitions and Notations

This section is for the proofs related to the probability distribution in the proposed Algorithm 1. For convenience, commonly used notations are listed in Table 4. We make some slight changes to some notations that appear in the main paper to make sure they are consistent and well-defined in this proof.

Similar to the previous analysis (Guo, Jerrum, and Liu 2019; Jerrum 2021), we begin by introducing the concept "dependency graph" (in Definition 1) for the constraints $\mathcal{C}$.

Definition 1 (Dependency Graph)**.** *The dependency graph $G = (\mathcal{C}, E)$, where the vertex set is the set of constraints $\mathcal{C}$. Two vertices c_i and c_j are connected with an edge $(c_i, c_j) \in E$ if and only if they are defined on at least one common random variable, i.e., $\texttt{var}(c_i) \cap \texttt{var}(c_j) \neq \emptyset$.*

For keeping track of the whole sampling procedure, we need the concept "sampling record"(in Definition 2) (Guo, Jerrum, and Liu 2019), which are the broken constraints at every round in Algorithm 1. It is also known as the witness tree in Moser and Tardos (2010). This allows us to check the constraint satisfaction of the assignment at every round.

Under Condition 1, for any edge in the dependency graph $(c_i, c_j) \in E$, either $\mathbf{1}(x, c_i) = 0$ or $\mathbf{1}(x, c_j) = 0$ for all $x \in \mathcal{X}$. In other words, two constraints with shared related variables, representing two *adjacent vertices* in the dependency graph G, are not broken simultaneously. Thus, the constraints in record S_t form an *independent set*[2] over the dependency graph under Condition 1.

Definition 2 (Sampling Record)**.** *Given dependency graph $G(\mathcal{C}, E)$, let $X_t = x$ be one possible assignment obtained at round t of Algorithm 1. Let $S_t \subseteq \mathcal{C}$ be the set of vertices in graph G (subset of constraints) that x violates*

$$S_t = \{c_i | c_i \in \mathcal{C} \text{ and } \mathbf{1}(x, c_i) = 0\}, \tag{16}$$

where indicator function $\mathbf{1}(x, c_i) = 0$ implies x violates constraint c_i at round t. Define the sampling record as the sequence of violated constraints $S_1, \ldots, S_t$ throughout the execution.

At round t ($t \geq 1$) of Algorithm 1, suppose the violated constraints is $S_t \subseteq \mathcal{C}$. The constraints that are not adjacent to S_t in the dependency graph are still satisfied after re-sample. The only possible constraints that might be broken after the re-sample operation are among S_t itself, or those constraints directly connected to S_t in the dependency graph. Therefore,

$$S_{t+1} \subset \Gamma(S_t), \qquad \text{for all } t \geq 1.$$

where $\Gamma(S_t)$ is the set of vertices of S_t and its adjacent neighbors in the dependency graph G (see Table 4). When Algorithm 1 terminates at round $T + 1$, no constraints are violated anymore, i.e., $s_{T+1} = \emptyset$. To summarize the above discussion on a sampling record by Algorithm 1, we have the following Claim 1.

Claim 1. *Under Condition 1, a potential sampling record of length $T + 1$ by the Algorithm 1 is a sequence of independent sets: $S_1, S_2, \ldots, S_T, \emptyset$ with*

1. *$S_{t+1} \subseteq \Gamma(S_t)$ and $S_t \neq \emptyset$, for $1 \leq t \leq T$;*
2. *$s_{T+1} = \emptyset$.*

Extra Notations related to Constrained MRF The constrained MRF model over constraints set $\mathcal{C}$ is defined as:

$$P_\theta(X = x|\mathcal{C}) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C})}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}(x', \mathcal{C})}$$

where the partition function only sums over valid assignments in $\mathcal{X}$. Note that $C(x)$ in Equation (6) is the same as $\mathbf{1}(x', \mathcal{C})$ in the above equation. We slightly change the notations for consistency in this proof. Also notice that the output distribution can no longer be factorized after constraints are enforced, since the partition function cannot be factorized. Our task is to draw samples from this distribution.

To analyze the intermediate steps in Algorithm 1, we further need to define the following notations.

Definition 3. *The constrained MRF distribution for constraints $\mathcal{C} \backslash \Gamma(S_t)$ is*

$$P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t)) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C}\backslash\Gamma(S_t))}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}(x', \mathcal{C}\backslash\Gamma(S_t))}$$

Definition 4. *At round t of Algorithm 1, assume $S_t \subseteq \mathcal{C}$ are the set of broken constraints, Define $\mathbb{P}(X_{t+1} = x|S_1, \ldots, S_t)$ to be the probability of obtaining a new assignment x after we re-sample random variables indexed by $\texttt{var}(S_t)$.*

[2] A set of vertices with no two adjacent vertices in the graph.

Table 4: Summary of all the notations used in the theoretical analysis of Algorithm 1.

Notation	Definition
$X = \{X_i\}_{i=1}^n$	set of discrete random variables
$x \in \mathcal{X}$	possible assignments for variables X
$x_i \in \mathcal{X}_i$	variable X_i can take all values in $\mathcal{X}_i$
$\mathcal{C} = \{c_j\}_{j=1}^m$	given constraints
$S_t \subseteq \mathcal{C}$	subset of constraints violated at round t of Algorithm 1
$G(\mathcal{C}, E)$	the dependency graph (in Definition 1)
$\texttt{var}(c_j)$	the indices of domain variables that are related to constraint c_i
$\texttt{var}(S_t)$	the indices for domain variables that are related to constraints S_t
$\Gamma(c_j)$	c_j and its direct neighbors in the dependency graph
$\Gamma(S_t)$	S_t and direct neighbors of S_t in the dependency graph
$\mathcal{C}\backslash\Gamma(S_t)$	all constraints in $\mathcal{C}$ but not in $\Gamma(S_t)$
$S_1, \ldots, S_T, \emptyset$	a sampling record of Algorithm 1 (in Definition 2)
$\mathbf{1}(x, c_i)$	indicator function that evaluates if assignment x satisfies constraint c_i
$\mathbf{1}(x, S_t)$	indicator function that evaluates if assignment x satisfies constraints in S_t
$\mathbf{1}(x, \mathcal{C})$	indicator function that evaluates if assignment x satisfies all constraints $\mathcal{C}$
$P_\theta(X\|\mathcal{C}\backslash\Gamma(S_t))$	see Definition 3
$\mathbb{P}(X\|S_t)$	see Definition 4

A.2 Ratio Property Lemma

Lemma 1 (Ratio Property)**.** *Under Condition 1, assume Algorithm 1 is at round t. Conditioning on observing one possible sampling record $S_1, \ldots, S_t$, Algorithm 1 step 4 will re-sample variables in $\texttt{var}(S_t)$ at round $t+1$. Let $x, x' \in \mathcal{X}$ be two possible assignments after this re-sample. The probability ratio of obtaining these two results equals that under constrained MRF $P_\theta(x|\mathcal{C}\backslash\Gamma(S_t))$:*

$$\frac{\mathbb{P}(X_{t+1} = x|S_1, \ldots, S_t)}{\mathbb{P}(X_{t+1} = x'|S_1, \ldots, S_t)} = \frac{P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t))}{P_\theta(X = x'|\mathcal{C}\backslash\Gamma(S_t))}, \tag{17}$$

where $\mathbb{P}(X_{t+1} = x|S_1, \ldots, S_t)$ is the probability of Algorithm 1 step 4 produces assignment x at round $t+1$, conditioning on the observed record $S_1, \ldots, S_t$ and re-sample $\texttt{var}(S_t)$. $P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t))$ is the constrained MRF (for the constraints $\mathcal{C}\backslash\Gamma(S_t)$) probability on assignment x.

Proof. During the intermediate step of the algorithm, assume the set of constraints S_t are violated. We want to re-sample variables indexed by $\texttt{var}(S_t)$, so variables indexed by $\texttt{var}(\mathcal{C}\backslash\Gamma(S_t))$ won't change assignments. Also, because $\Gamma(S_t)$ is the largest possible set of constraints that can be infected by the re-sample, constraints $\mathcal{C}\backslash\Gamma(S_t)$ are still satisfied after the re-sample.

At round t, we re-sample variables in $\texttt{var}(S_t)$ according to step 4 in Algorithm 1, we thus have:

$$\mathbb{P}(X^{t+1}_{\texttt{var}(S_t)} = x_{\texttt{var}(S_t)}|S_1 \ldots S_t) = \prod_{i \in \texttt{var}(S_t)} \frac{\exp(\theta_i x_i)}{\sum_{x_i' \in \mathcal{X}_i} \exp(\theta_i x_i')}.$$

Here the notation $X^{t+1}_{\texttt{var}(S_t)} = x_{\texttt{var}(S_t)}$ means $X_i = x_i$ for $i \in \texttt{var}(S_t)$ at round t. For any two possible assignments x, x' after the re-sample,

$$x_i = x_i', \quad \text{for } i \in \{1, \ldots, n\}\backslash\texttt{var}(S_t)$$

since the rest variable's assignments are kept the same after re-sample. Thus, we can have ratio:

$$\frac{\mathbb{P}(X_{t+1} = x|S_1, \ldots, S_t)}{\mathbb{P}(X_{t+1} = x'|S_1, \ldots, S_t)} = \frac{\exp(\sum_{i \in \texttt{var}(S_t)} \theta_i x_i)}{\exp(\sum_{i \in \texttt{var}(S_t)} \theta_i x_i')} = \frac{\exp(\sum_{i \in \texttt{var}(\Gamma(S_t))} \theta_i x_i)}{\exp(\sum_{i \in \texttt{var}(\Gamma(S_t))} \theta_i x_i')}. \tag{18}$$

For the last step, since every assignment outside $\texttt{var}(S_t)$ is not changed, we can enlarge the index set of summation to $\Gamma(S_t)$ by multiplying

$$1 = \frac{\exp(\sum_{i \in \texttt{var}(\Gamma(S_t))\backslash\texttt{var}(S_t)} \theta_i x_i)}{\exp(\sum_{i \in \texttt{var}(\Gamma(S_t))\backslash\texttt{var}(S_t)} \theta_i x_i')}.$$

After re-sample, we knows that x must satisfy the constraints $\mathcal{C}\backslash\Gamma(S_t)$. Thus, the probability of this x conditioned on constraints $\mathcal{C}\backslash\Gamma(S_t)$ holding in the constrained MRF model is:

$$P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t)) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C}\backslash\Gamma(S_t))}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x_i')\mathbf{1}(x', \mathcal{C}\backslash\Gamma(S_t))} = \frac{\exp(\sum_{i=1}^n \theta_i x_i)}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x_i')\mathbf{1}(x', \mathcal{C}\backslash\Gamma(S_t))}.$$

In the constrained MRF model (for constraints $\mathcal{C}\backslash\Gamma(S_t)$), the ratio of these two probabilistic assignments x, x' is:

$$\frac{P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t))}{P_\theta(X = x'|\mathcal{C}\backslash\Gamma(S_t))} = \frac{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t))}\theta_i x_i)}{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t))}\theta_i x'_i)}, \tag{19}$$

because the x_i outside $\texttt{var}(\Gamma(S_t))$ remains the same. Note that x, x' are two possible assignments produced according to to step 4 in Algorithm 1 at round t. Combining Equation (18) and Equation (19), we conclude that:

$$\frac{\mathbb{P}(X_{t+1} = x|S_1, \ldots, S_t)}{\mathbb{P}(X_{t+1} = x'|S_1, \ldots, S_t)} = \frac{P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t))}{P_\theta(X = x'|\mathcal{C}\backslash\Gamma(S_t))}.$$

The proof is finished. □

A.3 Proof of Theorem 1

Suppose the re-sampling process terminates at round $T+1$ and we obtain a valid sample x. Upon the termination of Algorithm 1, all the constraints are satisfied. So we have: $S_{T+1} = \emptyset$. In other words, $\mathbf{1}(x, \mathcal{C}) = 1$.

Let x, x' be two possible valid assignments produced at round $T + 1$ by the Algorithm 1. Using the analysis in Lemma 1, we can still have:

$$\frac{\mathbb{P}(X_{T+1} = x|S_1, \ldots, S_T)}{\mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)} = \frac{\exp(\sum_{i\in\texttt{var}(S_T)}\theta_i x_i)}{\exp(\sum_{i\in\texttt{var}(S_T)}\theta_i x'_i)}.$$

The probability of this x in the constrained MRF model (for constraints $\mathcal{C}$) is:

$$P_\theta(X = x|\mathcal{C}) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C})}{\sum_{x'\in\mathcal{X}}\exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}\,(x', \mathcal{C})} = \frac{\exp(\sum_{i=1}^n \theta_i x_i)}{\sum_{x'\in\mathcal{X}}\exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}\,(x', \mathcal{C})}.$$

Then we conclude that:

$$\frac{\mathbb{P}(X_{T+1} = x|S_1, \ldots, S_T)}{\mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)} = \frac{P_\theta(X = x|\mathcal{C})}{P_\theta(X = x'|\mathcal{C})}.$$

Note that this ratio property holds for all the possible sampling records $S_1, \ldots, S_T, \emptyset$.

Summation of All Possible Sampling Records Define $\mathbb{P}(S_1, \ldots, S_T)$ to be the probability of observing record $S_1, \ldots, S_T$ by Algorithm 1. For any possible sampling record $S_1, \ldots, S_T, \emptyset$, the ratio property still holds:

$$\frac{\mathbb{P}(X_{T+1} = x|S_1, \ldots S_T)\mathbb{P}(S_1, \ldots, S_T)}{\mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)\mathbb{P}(S_1, \ldots, S_T)} = \frac{P_\theta(X = x|\mathcal{C})}{P_\theta(X = x'|\mathcal{C})}$$

where the term $\mathbb{P}(S_1, \ldots, S_T)$ on the Left-hand-side (LHS) is actually the same. After we summarize over all possible sampling records $S_1, \ldots, S_T, \emptyset$, the ratio property still holds. Let $\mathbb{P}(X_{T+1} = x)$ be the probability of obtaining one valid assignment x by the execution of Algorithm 1.

$$\frac{\mathbb{P}(X_{T+1} = x)}{\mathbb{P}(X_{T+1} = x')} = \frac{\sum_{S_1,\ldots,S_T}\mathbb{P}(X_{T+1} = x|S_1, \ldots, S_T)\mathbb{P}(S_1, \ldots, S_T)}{\sum_{S_1,\ldots,S_T}\mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)\mathbb{P}(S_1, \ldots, S_T)} = \frac{P_\theta(X = x|\mathcal{C})}{P_\theta(X = x'|\mathcal{C})} \tag{20}$$

Sample Space Analysis At Termination We need one more statement to show Theorem 1 holds. Let $\mathcal{X}_{\text{LLL}}$ be the set of all possible assignments x that can be generated by Algorithm 1:

$$\mathcal{X}_{\text{LLL}} = \bigcup_{S_1\ldots S_T}\{x|\mathbb{P}(X_{T+1} = x|S_1 \ldots S_T) \neq 0 \text{ and } \mathbb{P}(S_1 \ldots S_T) \neq 0\}.$$

where $\mathbb{P}(S_1 \ldots S_T) \neq 0$ means $S_1, \ldots, S_T$ is a possible record. $\mathbb{P}(X_{T+1} = x|S_1 \ldots S_T) \neq 0$ means it is possible to obtain x given the record $S_1, \ldots, S_T$.

Let $\mathcal{X}_\mathcal{C}$ be the set of assignments x that satisfy all the constraints in the constrained MRF (for constraints $\mathcal{C}$):

$$\mathcal{X}_\mathcal{C} = \{x|P_\theta(X = x|\mathcal{C}) \neq 0, \text{ for all } x \in \mathcal{X}\}.$$

Lemma 2. *$\mathcal{X}_{LLL} \subseteq \mathcal{X}_\mathcal{C}$ and $\mathcal{X}_\mathcal{C} \subseteq \mathcal{X}_{LLL}$, thus $\mathcal{X}_{LLL} = \mathcal{X}_\mathcal{C}$.*

Proof. When Algorithm 1 terminates, it only produces valid assignments; thus, we must have: $\mathcal{X}_{\text{LLL}} \subseteq \mathcal{X}_\mathcal{C}$. On the other hand, there is always a non-zero probability that Algorithm 1 will generate every valid assignment $x \in \mathcal{X}_\mathcal{C}$, which implies that $\mathcal{X}_\mathcal{C} \subseteq \mathcal{X}_{\text{LLL}}$. Therefore we can conclude that $\mathcal{X}_{\text{LLL}} = \mathcal{X}_\mathcal{C}$. □

Lemma 2 show that the two distributions have the same sample space when Algorithm 1 terminates. What's more, Equation (20) shows they have the same probability ratio for any possible valid assignments x, x'. This shows that the execution of the Algorithm 1 is a random draw from the constrained MRF distribution $P_\theta(X = x|\mathcal{C})$. The proof of Theorem 1 is finished.

A.4 Difference to the Original Proof

The main difference in the above proof to the existing proof in (Guo, Jerrum, and Liu 2019, Lemma 7) is that: We show Lemma 1 that characterizes the proportional ratio of getting different assignments of variables, which is more general than the descriptive proof for Guo, Jerrum, and Liu (2019, Lemma 7).

A.5 A Running Example in View of Markov Chain

We dedicate this section to demonstrate the execution of Algorithm 1 with Example 1. Algorithm 1 can be viewed as a Markov chain, so we will show the probability of obtaining valid samples is unbiased by running thousands of steps of the Markov chain. The constraints are $\mathcal{C} = \{c_1 = (X_1 \vee X_2), c_2 = (\neg X_1 \vee X_3)\}$. We use the assignment of all the variables as the states $s_1, \ldots, s_8$ in the rounds of Algorithm 1.

$$\begin{aligned} s_1 &= (X_0 = 0, X_1 = 0, X_2 = 0) \\ s_2 &= (X_0 = 0, X_1 = 0, X_2 = 1) \\ s_3 &= (X_0 = 0, X_1 = 1, X_2 = 0) \\ s_4 &= (X_0 = 0, X_1 = 1, X_2 = 1) \\ s_5 &= (X_0 = 1, X_1 = 0, X_2 = 0) \\ s_6 &= (X_0 = 1, X_1 = 0, X_2 = 1) \\ s_7 &= (X_0 = 1, X_1 = 1, X_2 = 0) \\ s_8 &= (X_0 = 1, X_1 = 1, X_2 = 1) \end{aligned} \tag{21}$$

Here s_1, s_2, s_3, s_4 correspond to *valid* assignments of variables with respect to the constraints $\mathcal{C}$ and s_5, s_6, s_7, s_8 correspond to *invalid* assignments of variables, that requires resampling.

For simplicity, we consider the uniform setting where $\theta_1 = \theta_2 = \theta_3$. The goal is to sample every valid assignment with equal probability. Therefore, the probability for every variable is:

$$P(X_i) = \begin{cases} \frac{1}{2} & \text{for variable } X_i \text{ taking value } 1 \\ \frac{1}{2} & \text{for variable } X_i \text{ taking value } 0 \end{cases}$$

for $i = 1, 2, 3$. Based on Algorithm 1, we know the probability of transferring from s_i to s_j ($1 \leq i, j \leq 8$). Thus we can construct the transition matrix between every state:

$$T = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix} \end{array} \tag{22}$$

where $T_{ij} = T(s_i, s_j)$ denotes the transition probability from state s_i to state s_j.

Taking state s_5 as an example, it violates constraint C_2 thus X_2, X_3 will be resampled. There are 4 possible assignments of X_2, X_3, which corresponds to states $\{s_1, s_2, s_3, s_5\}$. Since each variable is resampled uniformly at random, the probability of transition from state s_5 to the states $\{s_1, s_2, s_3, s_5\}$ are $1/4$. The Algorithm 1 will terminate once it reaches states $\{s_1, s_2, s_3, s_4\}$, which corresponds to the (valid) state only transit to itself with probability 1. Thus we find $T(s_i, s_i) = 1$ for $i = 1, 2, 3, 4$.

For a randomly initialized assignment:

$$x = \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \end{bmatrix} \end{array} \tag{23}$$

that has an equal probability of being any state. After executing Algorithm 1 for 2000 steps, we have:

$$T^{2000} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 & \frac{1}{4} \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{2} & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}, \quad xT^{2000} = \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{24}$$

This implies Algorithm 1 outputs every valid assignment with the same probability in the uniform setting, which follows the result in Theorem 1.

B Running Time Analysis of Algorithm 1

We dedicate this section to showing the running time of Algorithm 1 on a general weighted case. The expected running time of Algorithm 1 is determined by the number of rounds of re-sampling. Algorithm 1 re-sample all the related random variables simultaneously in every single round. However, it is hard to get an estimation of the exact total running time over the *random variables*. Instead, we can only have a loose upper bound of the expected running time over the *sequence of sampling record* (the sequence of violated constraints).

The overall structure of the proof is similar to the proof in Guo, Jerrum, and Liu (2019, Theorem 13). We show the difference in our proof at the end of this section.

B.1 Definitions and Notations

We define the following terms to simplify our notations.

Definition 5. *Let S_t be a subset of vertices in a dependency graph. 1) Define p_{S_t} as the probability of constraints in S_t being violated:*

$$p_{S_t} = \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i\right) \tag{25}$$

where we use $\neg c_i$ to indicate the constraint c_i is violated. 2) Define q_{S_t} as the probability that only the constraints in S_t are violated and nothing else.

$$q_{S_t} = \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j\right) \tag{26}$$

where $\bigwedge_{c_i \in S_t} \neg c_i$ corresponds to only the constraints in S_t are violated and $\bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j$ corresponds to all the rest constraints are satisfied. So $q_{\{c_i\}}$ is the probability that only constraint c_i is broken and all the rest still hold. Similarly, $q_\emptyset$ denotes the probability that all the constraints are satisfied.

Lemma 3. *Given Definition 5, we can further expand q_{S_t} under Condition 1:*

$$q_{S_t} = p_{S_t} \mathbb{P}\left(\wedge_{c_j \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right)$$

Proof. We can split q_{S_t} into the probability of two independent events:

$$\begin{aligned}
q_{S_t} &= \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j\right) && \text{By definition of } q_{S_t} \text{ in Equation (26)} \\
&= \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right) \\
&= \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i\right) \mathbb{P}\left(\bigwedge_{c_n \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right) \\
&= p_{S_t} \mathbb{P}\left(\wedge_{c_j \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right). && \text{By definition of } p_{S_t} \text{ in Equation (25)}
\end{aligned}$$

The second equality holds because under Condition 1, adjacent vertices have zero probability. In other words, when we observe that constraints in S_t are violated, constraints in $\Gamma(S_t) \setminus S_t$ cannot be violated. The third equality holds because the random variables in $\texttt{var}(S_t)$ are *independent to* those variables in $\texttt{var}(\mathcal{C} \setminus \Gamma(S_t))$. So we can apply $P(AB) = P(A)P(B)$ when the events A, B are independent to each other. □

Remark 1 (Equivalence of Record)**.** *At round t of Algorithm 1, it finds all the constraints S_t that are broken ($\bigwedge_{c_i \in S_t} \neg c_i$), which implies the rest of the constraints $\mathcal{C} \setminus \Gamma(S_t)$ are satisfied ($\bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j$). Thus the probability of observing S_t in the record is equivalent to the following:*

$$\mathbb{P}(S_t) = \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j\right) \tag{27}$$

Lemma 4. *Given a possible sampling record $S_1 \ldots S_{t-1}$,S_t by Algorithm 1, the following equality holds for the pair (S_{t-1}, S_t):*

$$\sum_{S_t} q_{S_t} = \mathbb{P}(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i)$$

Proof. By Definition 2 of the sampling record, we have $S_t \subset \Gamma(S_{t-1})$. The relationship of its complement would be:

$$\mathcal{C} \backslash \Gamma(S_{t-1}) \subset \mathcal{C} \backslash S_t.$$

Using the above result, we have:

$$\mathbb{P}\left(\bigwedge_{c_j \in \mathcal{C} \backslash S_t} c_j \wedge \bigwedge_{c_k \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_k \right) = \mathbb{P}\left(\bigwedge_{c_j \in \mathcal{C} \backslash S_t} c_j \right) \tag{28}$$

Based on Remark 1 and Baye's theorem, we have:

$$\begin{aligned}
\mathbb{P}(S_t | \wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i) &= \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \backslash S_t} c_j \Big| \wedge_{c_k \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_k \right) && \text{By Equation (27)} \\
&= \frac{\mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \backslash S_t} c_j \wedge \bigwedge_{c_k \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_k \right)}{\mathbb{P}\left(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i \right)} && \text{By Bayes's formula} \quad (29) \\
&= \frac{\mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_k \in \mathcal{C} \backslash S_t} c_k \right)}{\mathbb{P}\left(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i \right)} && \text{By Equation (28)} \\
&= \frac{q_{S_t}}{\mathbb{P}\left(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i \right)}. && \text{By definition of } p_{S_t} \text{ in Equation (26)}
\end{aligned}$$

Since LHS of Equation (29) sums over all possible S_t is one: $\sum_{S_t} \mathbb{P}(S_t | \wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i) = 1$. Thus, summing over S_t for the RHS of Equation (29), we have:

$$1 = \sum_{S_t} \frac{q_{S_t}}{\mathbb{P}\left(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i \right)} = \frac{\sum_{S_t} q_{S_t}}{\mathbb{P}\left(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i \right)} \tag{30}$$

In the second equality, the reason that we can move the summation operator to the numerator is that the denominator is a constant *w.r.t.* all possible S_t. To be specific, given $S_t \subseteq \Gamma(S_{t-1})$, we have S_t is independent to $\mathcal{C} \backslash \Gamma(S_{t-1})$. Based on Equation (30), we finally obtain:

$$\sum_{S_t} q_{S_t} = \mathbb{P}(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i).$$

The proof is finished. □

Lemma 5. *The probability of observing the sampling record $S_1, \ldots, S_T$ by Algorithm 1 under Condition 1 is:*

$$\mathbb{P}(S_1, \ldots, S_T) = q_{S_T} \prod_{t=1}^{T-1} p_{S_t} \tag{31}$$

Proof. Given sampling record $S_1, \ldots, S_{t-1}$, the conditional probability of observing the next record S_t, S'_t can be expanded based on Lemma 1,

$$\frac{\mathbb{P}(S_t | S_1, \ldots, S_{t-1})}{\mathbb{P}(S'_t | S_1, \ldots, S_{t-1})} = \frac{\mathbb{P}(S_t | \wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i)}{\mathbb{P}(S'_t | \wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i)}$$

Based on Equation (29), we can simplify the RHS of the above ratio equality and] obtain:

$$\frac{\mathbb{P}(S_t | S_1, \ldots, S_{t-1})}{\mathbb{P}(S'_t | S_1, \ldots, S_{t-1})} = \frac{q_{S_t}}{q_{S'_t}}$$

Because of $\sum_{S_t} \mathbb{P}(S_t | S_1, \ldots, S_{t-1}) = 1$ and Equation (30), we can get:

$$\mathbb{P}(S_t | S_1, \ldots, S_{t-1}) = \frac{q_{S_t}}{\mathbb{P}(\wedge_{c_i \in \mathcal{C} \backslash \Gamma(S_{t-1})} c_i)} \tag{32}$$

We finally compute the probability of observing the sampling record $S_1 \ldots, S_T$ by:

$$\begin{aligned}
\mathbb{P}(S_1 \ldots, S_T) =& \mathbb{P}(S_1) \prod_{t=2}^{T} \mathbb{P}(S_t | S_1, \ldots, S_{t-1}) && \text{By Chain rule} \\
=& q_{S_1} \prod_{t=2}^{T} \frac{q_{S_t}}{\mathbb{P}(\wedge_{c_i \in \mathcal{C} \setminus \Gamma(S_{t-1})} c_i)} && \text{By Equation (32)} \\
=& q_{S_T} \prod_{t=2}^{T} \frac{q_{S_{t-1}}}{\mathbb{P}(\wedge_{c_i \in \mathcal{C} \setminus \Gamma(S_{t-1})} c_i)} && \text{Left shift the numerator from } S_t \text{ to } S_{t-1} \\
=& q_{S_T} \prod_{t=1}^{T-1} p_{S_t} && \text{Plugin Lemma 3}
\end{aligned}$$

The proof is finished. □

B.2 An Upper Bound on Expected Running Time

Suppose the expected number of samplings of constraints c_i is $\mathbb{E}(T_i)$, then the total running time will be:

$$\mathbb{E}(T) \leq \sum_{i=1}^{n} \mathbb{E}(T_i)$$

Since each random variable has equal status, then the question comes down to the computation of individual T_i's expectation. Let $S_1, \ldots, S_T$ be any record of the algorithm that successfully terminates, and $T_i(S_1, \ldots, S_T)$ be the total number of sampling related to constraint c_i throughout this record. Based on Lemma 5, we have:

$$\mathbb{E}(T_i) = \sum_{S_1, \ldots, S_T} \mathbb{P}(S_1, \ldots, S_T) T_i(S_1, \ldots, S_T)$$

By far, we have shown the original proof of our work. We leave the difference between our proof with the existing one in Appendix B.3.

The rest of the computation can be done in the same way as the proof in Guo, Jerrum, and Liu (2019). Thus we cite the necessary intermediate steps in the existing work and finish the proof logic for the coherence of the whole running time analysis.

Lemma (Guo, Jerrum, and Liu (2019) Lemma 12)**.** *Let $q_\emptyset$ be a non-zero probability of all the constraints are satisfied. Let $q_{\{c_j\}}$ denote the probability that only constraint c_j is broken and the rest all hold. If $q_\emptyset > 0$, then $\mathbb{E}(T_i) = q_{\{c_j\}}/q_\emptyset$.*

After incorporating our fix, we can conclude the upper bound on the expected running time in Theorem 2.

Theorem 2 (Guo, Jerrum, and Liu (2019) Theorem 13)**.** *Under Condition 1, the total number of re-samplings throughout the algorithm is then $\frac{1}{q_\emptyset} \sum_{j=1}^{L} q_{\{c_j\}}$.*

B.3 Difference to the Existing Proof

The main difference in the above proof to the existing proof in (Guo, Jerrum, and Liu 2019, Theorem 13) is that: based on Equation (32) and (29), we show

$$\mathbb{P}(S_t | S_1, \ldots, S_{t-1}) = \mathbb{P}(S_t | \wedge_{c_i \in \mathcal{C} \setminus \Gamma(S_{t-1})} c_i)$$

In Guo, Jerrum, and Liu (2019)'s Equation (9), the first step cannot holds without the above equality. The original paper uses this result directly without providing enough justification.

C Constrained MRF Model

C.1 Single Variable Form of Constrained MRF

Here we provide an example of transforming MRF with pairwise and single potential functions into a single potential form by introducing extra variables. Given random variables X_1, X_2, X_3, we have the following example MRF model:

$$\phi_\theta(x_1, x_2, x_3) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2$$
$$P_\theta(x) = \frac{\exp(\phi_\theta(x_1, x_2, x_3))}{Z(\theta)}$$

X_1	X_2	$\hat{X}_{00}, \hat{X}_{01}, \hat{X}_{10}, \hat{X}_{11}$
0	0	$1, 0, 0, 0$
0	1	$0, 1, 0, 0$
1	0	$0, 0, 1, 0$
1	1	$0, 0, 0, 1$

Table 5: 4 constraints for converting pairwise terms in the potential function into single variable form.

In the above formula, we have a cross term x_1x_2. Two Boolean variables can have 4 different assignments in total. Therefore we can construct 4 extra Boolean variables to encode all these assignments. To illustrate, we introduce extra random variables $\hat{X}_{00}$, $\hat{X}_{01}$, $\hat{X}_{10}$, $\hat{X}_{11}$. We further introduce extra constraints: When $X_1 = 0, X_2 = 0$, the extra variable must take values: $\hat{X}_{00} = 1$, $\hat{X}_{01} = 0$, $\hat{X}_{10} = 0$, $\hat{X}_{11} = 0$. See the rest constraints in Table 5.

Then the new potential function, including extended variables and pairwise to single variable constraints $\mathcal{C}$, is reformulated as:

$$\hat{\phi}_\theta(x_1, x_2, x_3, \hat{x}_{00}, \hat{x}_{01}, \hat{x}_{10}, \hat{x}_{11}) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 \hat{x}_{00} + \theta_3 \hat{x}_{01} + \theta_3 \hat{x}_{10} + \theta_3 \hat{x}_{11}$$

$$P_\theta(x|\mathcal{C}) = \frac{\exp(\hat{\phi}_\theta(x_1, x_2, x_3, \hat{x}_{00}, \hat{x}_{01}, \hat{x}_{10}, \hat{x}_{11}))}{Z_{\mathcal{C}}(\theta)}$$

For clarity, the newly added constraints do not impact Condition 1. Since the single variable transformation in the MRFs model originates from Sang, Bearne, and Kautz (2005), thus is not considered as our contribution.

C.2 Gradient of $\log$-Partition Function $\nabla \log Z_{\mathcal{C}}(\theta)$

We use the Chain rule of the gradient to give a detailed deduction of Equation (4).

$$\begin{aligned}\nabla \log Z_{\mathcal{C}}(\theta) = \frac{\nabla Z_{\mathcal{C}}(\theta)}{Z_{\mathcal{C}}(\theta)} = \frac{1}{Z_{\mathcal{C}}(\theta)} \nabla \sum_{x \in \mathcal{X}} \exp\left(\phi_\theta(x)\right) C(x) = \sum_{x \in \mathcal{X}} \frac{\exp(\phi_\theta(x)) C(x)}{Z_{\mathcal{C}}(\theta)} \nabla \phi_\theta(x) = \sum_{x \in \mathcal{X}} P_\theta(x|\mathcal{C}) \nabla \phi_\theta(x) \\ = \mathbb{E}_{x \sim P_\theta(\tilde{x}|\mathcal{C})} \left(\nabla \phi_\theta(x)\right)\end{aligned} \tag{33}$$

The above result shows the gradient of the constrained partition function is equivalent to the expectation of the gradient of the potential function $\nabla \phi_\theta$ over the model's distribution (*i.e.*, $P_\theta(\tilde{x}|\mathcal{C})$). Therefore, we transform the gradient estimation problem into the problem of sampling from the current MRF model.

D Experiment Settings and Configurations

D.1 Implementation Details

Implementation of NELSON The proposed sampler can be implemented with Numpy, Pytorch, or Jax. We further offer a "batch version" implementation, that draws a batch of samples in parallel on GPU. The batched sampler is useful for those tasks that require a huge number of samples to estimate the gradient with a small approximation error.

In Section 4.1, we define vector of assignment $x^t = (x_1^t, \ldots, x_n^t)$, where x_i^t is the assignment of variable X_i in the t-th round of Algorithm 1. $x_i^t = 1$ denotes variable X_i takes value 1 (or true). In the batch version, we define the matrix for a batch of assignments. Let b be the batch size, we have

$$x^t = \begin{bmatrix} x_{11}^t & \ldots & x_{1n}^t \\ \vdots & \ddots & \vdots \\ x_{b1}^t & \ldots & x_{bn}^t \end{bmatrix}$$

In the following part, we provide the detailed computation pipeline for the batch version of the proposed algorithm.

Initialization The first step is to sample an initial assignment of X from the given the marginal probability vector P:

$$x_{li}^1 = \begin{cases} 1 & \text{if } u_{li} > P_i, \\ 0 & \text{otherwise.} \end{cases}, \quad \text{if} 1 \leq i \leq n, 1 \leq l \leq b \tag{34}$$

Here u_{li} is sampled from the uniform distribution over $[0, 1]$.

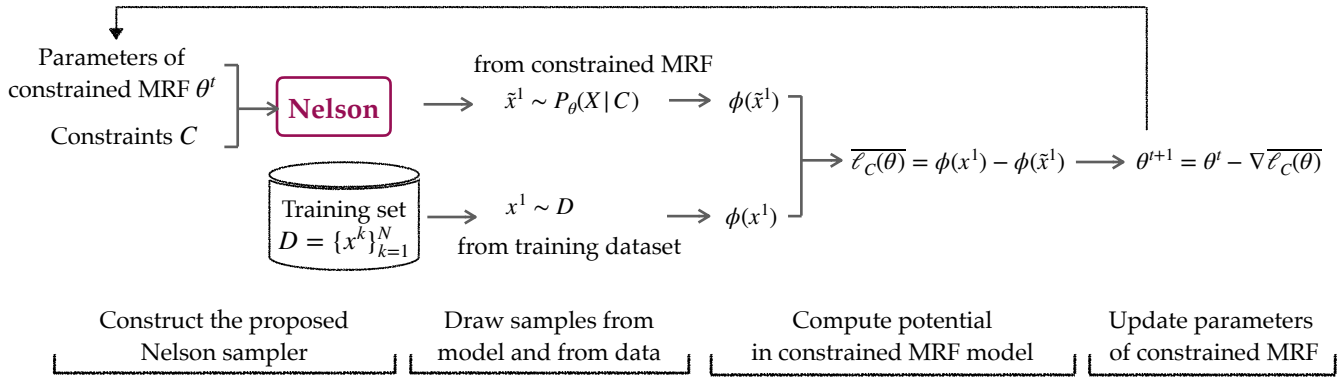


Figure 4: Implementation pipeline of the NELSON-CD algorithm with $m = 1$. The proposed NELSON can be efficiently adapted to a Pytorch-based machine learning library and enforces constraint satisfaction during learning.

Check Constraint Satisfaction The second step extract which constraint is violated. Given an assignment x^t at round $t \geq 1$, tensor W and matrix b, the computation of tensor Z^t is:

$$Z^t_{lik} = \sum_{i=1}^{n} W_{ikj} x^t_{lj} + b_{lik},$$

The special multiplication between tensor and matrix can be efficiently implemented with the Einstein summation[3]. Note that $Z^t_{ljk} = 1$ indicates for l-th batched assignment x_l, the k-th literal of j-th clause is true (takes value 1). Next, we compute S^t_{lj} as:

$$S^t_{lj} = 1 - \max_{1 \leq k \leq K} Z_{ljk}, \quad \text{for } 1 \leq j \leq L, 1 \leq l \leq b$$

Here $S^t_{lj} = 1$ indicates x^t_l violates j-th clause. We can check $\sum_{l=1}^{b} \sum_{j=1}^{L} S^t_{lj} \neq 0$ to see if any clause is violated for the current batch of assignments, which corresponds to $\sum_{i=1}^{b} C(x_l) = 0$.

Extract Variables in Violated Clauses We extract all the variables that require resampling based on vector S^t computed from the last step. The vector of the resampling indicator matrix A^t can be computed as:

$$A^t_{li} = \mathbf{1}\left(\sum_{j=1}^{L} S^t_{lj} V_{ji} \geq 1\right), \quad \text{for } 1 \leq i \leq n, 1 \leq l \leq b$$

where $\sum_{j=1}^{L} S^t_{lj} V_{ji} \geq 1$ implies X_{li} requires resampling.

Resample Given the marginal probability vector P, resample indicator matrix A^t and assignment matrix x^t, we draw a new random sample x^{t+1}.

$$x^{t+1}_{li} = \begin{cases} (1 - A^t_{li}) x^t_{li} + A^t_{li} & \text{if } u_{li} > P_i, \\ (1 - A^t_{li}) x^t_{li} & \text{otherwise.} \end{cases} \quad \text{for } 1 \leq i \leq n, 1 \leq l \leq b$$

where u_{li} is drawn from the uniform distribution in $[0, 1]$.

Since GPUs are more efficient at computing tensor, matrix, and vector operations but are slow at processing for loops. Drawing a batch of samples using the above extended computational pipeline is much faster than using a for loop over the computational pipeline in Section 4.1.

The sampler is involved with one hyper-parameter T_{tryout}. NELSON would terminate when it reaches T_{tryout} number of re-samples. This practice is commonly used to handle randomized programs that might run forever in rare cases.

Implementation of Algorithm 2 We first use the Constraints $\mathcal{C}$ and parameters θ^t to build the current NELSON module. Then we draw m samples from NELSON module $\{\tilde{x}^j\}_{j=1}^m$ and draw from dataset randomly m samples $\{x^j\}_{j=1}^m$. Continuing from that point, we compute the potential value from the two sets of inputs, i.e., $\{\phi_\theta(\tilde{x}^j)\}_{j=1}^m$ and $\{\phi_\theta(\tilde{x}^j)\}_{j=1}^m$. Pytorch would be slow if we compute each potential's gradient using a for-loop. To bypass this problem, we instead compute the following:

$$\overline{\ell_C(\theta)} = \frac{1}{m} \sum_{j=1}^{m} \phi_\theta(x^j) - \frac{1}{m} \sum_{j=1}^{m} \phi_\theta(\tilde{x}^j). \tag{35}$$

[3] https://github.com/dgasmith/opt_einsum

Following that, we call the PyTorch library's gradient function, which computes exactly

$$\nabla\overline{\ell_{\mathcal{C}}(\theta)} = \nabla\left(\frac{1}{m}\sum_{j=1}^{m}\phi_\theta(x^j) - \frac{1}{m}\sum_{j=1}^{m}\phi_\theta(\tilde{x}^j)\right) = \frac{1}{m}\sum_{j=1}^{m}\nabla\phi_\theta(x^j) - \frac{1}{m}\sum_{j=1}^{m}\nabla\phi_\theta(\tilde{x}^j)$$

Note that $\nabla\overline{\ell_{\mathcal{C}}(\theta)}$ recovers the result in Equation (4). Finally, we update the parameters θ. The proposed NELSON module and the neural network are computed on the same GPU device. This allows us to exploit the parallel computing power of modern GPUs and remove time for the data transfer from CPU to GPU or vice versa. See Figure 4 for a visualized overview of the implementation with Pytroch.

D.2 Learn Random K-SAT Solutions with Preference

Task Definition We are given a training set $\mathcal{D}$ containing some preferred assignments $\mathcal{D} = \{x^j\}_{j=1}^N$ for the corresponding CNF formula $c_1 \wedge \ldots \wedge c_L$. We require the CNF formula to be true. This means, by the definition of CNF formulas, that every clause has to be satisfied. These clauses become our set of constraints. Under the constrained MRF model, the learning task is to maximize the log-likelihood of the assignments seen in the training set $\mathcal{D}$. The inference task is to generate valid solutions from the learned model's distribution (Dodaro and Previti 2019; Rosa, Giunchiglia, and O'Sullivan 2011).

Dataset We denote the Boolean variables' size in K-SAT as the "problem size". We consider several datasets of different problem sizes generated from CNFGen[4] (Lauria et al. 2017) random K-SAT functions. K is fixed as 5; the number of variables and clauses are kept the same, ranging from 10 to 1500. We generate 100 different CNF formulas for every problem size. To generate the training set $\mathcal{D}$, we use the Glucose4 solver from PySAT[5] library (Ignatiev, Morgado, and Marques-Silva 2018) to generate 200 assignments randomly as the preferred assignments for every formula.

It should be pointed out that we don't consider datasets like SATLIB and SAT competitions. It is mainly because these datasets are hard instances with a much larger input space but a limited number of solutions. NELSON would generally take exponential time to find these solutions, just like finding needles in a haystack. The other reasons are that using neural networks to learn these limited assignments is straightforward since we can simply hard-wire the network to memorize all the valid assignments. The main purpose of this work is to let a constrained MRF learn a representation for the underlying preference pattern, not create a neural solver that can generate valid assignments for any CNF formula. Thus, we conform to the settings of the easy formula where obtaining valid solutions is easy.

D.3 Learn Sink-Free Orientation in Undirected Graphs

Task Definition In graph theory, a *sink-free* orientation of an undirected graph is a choice of orientation for each edge such that every vertex has at least one outgoing edge (Cohn, Pemantle, and Propp 2002). It has wide applications in robotics routing and IoT network configuration (Takahashi et al. 2009). The Constraints for this problem are that every vertex has at least one outgoing edge after orientation. As stated in (Guo, Jerrum, and Liu 2019), these constraints satisfy Condition 1.

See Figure 5 for an example graph and one possible sink-free edge orientation. We define binary variables $X_1, \ldots, X_m$, and associate variable X_i to edge e_i for $1 \leq i \leq m$. Variable X_i takes value 1 if the edge orientation is $v_i \to v_j$ where $i < j$. Otherwise, X_i takes value 0. The constraints are:

$$\mathcal{C} = (X_1 \vee X_2) \wedge (\neg X_1 \vee X_3 \vee \neg X_4) \wedge (\neg X_2 \vee \neg X_3 \vee X_5) \wedge (X_4 \vee \neg X_5)$$

where the single constraint $c_1 = (X_1 \vee X_2)$ corresponds to vertex v_1, constraint $c_2 = (\neg X_1 \vee X_3 \vee \neg X_4)$ corresponds to vertex v_2, constraint $c_3 = (\neg X_2 \vee \neg X_3 \vee X_5)$ corresponds to vertex v_3, and constraint $c_4 = (X_4 \vee \neg X_5)$ corresponds to vertex v_4. The orientation assignment matrix x shown in Figure 5(b) implies: $X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 0, X_5 = 1$.

Notations Let graph $G(V, E)$ be an un-directed graph; its adjacency matrix A that represents graph connectivity is:

$$A_{ij} = \begin{cases} 1 & \text{If } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{36}$$

A possible assignment for the orientation of every edge can be represented as a matrix $x \in \{0,1\}^{|V|\times|V|}$:

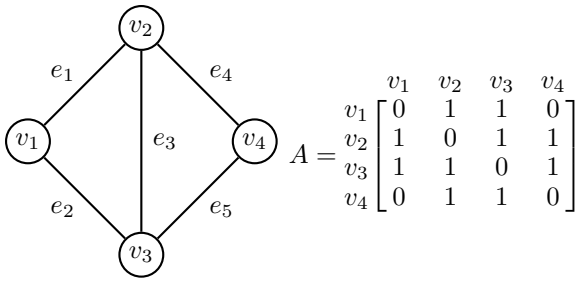
$$x_{ij} = \begin{cases} 1 & \text{if the edge orientation is } v_i \to v_j \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

In the constrained MRF model defined in Eq. (6), the potential function of one orientation of all edges is

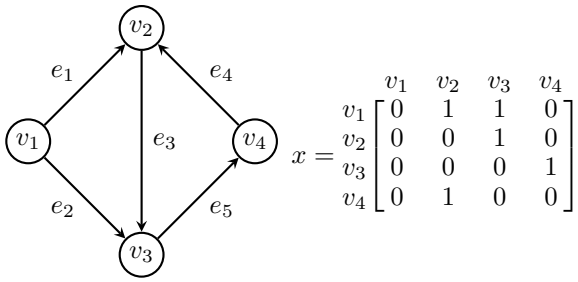
$$\phi_\theta(x) = \sum_{i=1}^{|V|}\sum_{j=1}^{|V|}\theta_{ij}A_{ij}x_{ij}$$

[4]https://github.com/MassimoLauria/cnfgen

[5]https://pysathq.github.io/



(a) An undirected graph G with its adjacency matrix A



(b) An orientation of the edges and the orientation matrix x

Figure 5: **(a)** An un-directed graph $G(V, E)$ where the vertices are $V = \{v_1, v_2, v_3, v_4\}$ and the un-directed edges are $E = \{e_1 = (v_1, v_2), e_2 = (v_1, v_3), e_3 = (v_2, v_3), e_4 = (v_2, v_4), e_5 = (v_3, v_4)\}$. **(b)** A possible sink-free orientation of the edges in the graph and its matrix representation x, where every vertex has at least one outgoing edge.

A single constraint for vertex v_k is $c_k(x) = \mathbf{1}\left(\sum_{j=1}^{n} A_{k,j}x_{k,j} = 1\right)$. If there is no ongoing edge of vertex v_k. The constraint function $C(x)$ is defined as: $\prod_{i=1}^{n} c_k(x)$. In Algorithm 1 step 1, edge (v_i, v_j) will pick the orientation $v_i \to v_j$ with probability:

$$\frac{\exp(\theta_{ij}A_{ij}x_{ij})}{\exp(\theta_{ji}A_{ji}x_{ji}) + \exp(\theta_{ij}A_{ij}x_{ij})}$$

Dataset We use the NetworkX[6] package to generate random Erdos Renyi graph with edge probability 0.55. The problem size refers to the number of vertices in the graph, we range the problem size from 10 to 100. For each problem size, we generate 100 different random undirected graphs. We then convert the graph into CNF form using the above edge-variable conversion rule. Afterward, we follow the same processing steps as the previous problem that learn preferential solution distribution for random K-SAT. .

D.4 Learn Vehicle Delivery Routes

Given a set of locations to visit, the task is to generate a sequence to visit these locations in which each location is visited once and only once and the sequence closely resembles the trend presented in the training data. The training data are such routes collected in the past. The dataset is constructed from TSPLIB, which consists of 29 cities in Bavaria, Germany. In Figure 3, we see NELSON can obtain samples of this delivery problem highly efficiently.

A possible travel plan can be represented as a matrix $x \in \{0, 1\}^{|V|\times|V|}$:

$$x_{ij} = \begin{cases} 1 & \text{if edge } v_i \to v_j \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

The constraints are that every routing plan should visit every location once and only once.

Similarly, in the constrained MRF model defined in Eq. (6), the potential function of the vehicle routing plan is

$$\phi_\theta(x) = \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} \theta_{ij}A_{ij}x_{ij}$$

D.5 Detailed Baselines Configuation

In terms of sampling-based methods, we consider:

- Gibbs sampler (Carter and Kohn 1994), a special case of MCMC that is widely used in training MRF models. In each step, the Gibbs algorithm samples one dimension based on a conditional marginal distribution. We follow this implementation[7].
- Weighted SAT samplers, including WAPS[8] (Gupta et al. 2019), WeightGen[9] (Chakraborty et al. 2014) and XOR sampler[10] (Ermon et al. 2013a; Ding and Xue 2021).

[6]https://networkx.org/

[7]https://github.com/Fading0924/BPChain-CD/blob/master/mrf.py

[8]https://github.com/meelgroup/waps

[9]https://bitbucket.org/kuldeepmeel/weightgen/src/master/

[10]https://cs.stanford.edu/~ermon/code/srcPAWS.zip

- Uniform SAT samplers, including UniGen[11] (Soos, Gocht, and Meel 2020), QuickSampler[12] (Dutra et al. 2018), CMS-Gen[13] (Golia et al. 2021) and KUS[14] (Sharma et al. 2018).

Currently, there are only GPU-based SAT solvers (Prevot, Soos, and Meel 2021; Mahmoud 2022) and model counters (Fichte, Hecher, and Zisser 2019), GPU-based SAT samplers are not available by far.

D.6 Detailed Definition of Evaluation Metrics

In terms of evaluation metrics, we consider

- Training time per epoch. The average time for the whole learning method to finish one epoch with each sampler.
- Validness. The learned model is adopted to generate assignments and we evaluate the percentage of generated assignments that satisfy the constraints.
- Mean Averaged Precision (MAP@10). This is a ranking-oriented metric that can evaluate the closeness of the learned MRF distribution to the goal distribution. If the model learns the goal distribution in the training set, then it would assign a higher potential value to those assignments in the training set than all the rest unseen assignments. Based on this principle, we randomly pick two sets of inputs in those valid assignments: seen assignments from the training set and unseen assignments that are randomly generated. We use the value of factor potential $\phi(x)$ to rank those assignments in ascending order. Next, we check how many preferred solutions can fall into the Top-10 by computing the following

$$\text{MAP@10} = \sum_{k=1}^{10} \frac{\#\text{preferred assignments among top-}k}{k}$$

- log-likelihood of assignments in the training set $\mathcal{D}$. The model that attains the highest log-likelihood learns the closest distribution to the training set. Specifically, given a training set $\mathcal{D} = \{x^k\}_{k=1}^N$ and parameters θ, the log-likelihood value is:

$$\frac{1}{N}\sum_{k=1}^{N} \log P_\theta(X = x^k|\mathcal{C}) = \frac{1}{N}\sum_{k=1}^{N} \phi_\theta(x^k) - \log Z_\mathcal{C}(\theta) \tag{39}$$

  We use the ACE algorithm to compute the approximated value of $\log Z_\mathcal{C}(\theta)$[15].

- Approximation Error of $\nabla \log Z_\mathcal{C}(\theta)$, that is the L_1 distance between the exact gradient of $\log Z_\mathcal{C}(\theta)$ in Eq. (4) and the empirical gradient from the sampler. For small problem sizes, we enumerate all $x \in \mathcal{X}$ to get the exact gradient and draw samples $\{\tilde{x}^j\}_{j=1}^m$ with $m = 2000$ from every sampler for approximation.

$$\Big| \underbrace{\sum_{x\in\mathcal{X}} \frac{\exp\left(\sum_{j=1}^n \theta_j x_j\right) C(x)}{Z_\mathcal{C}(\theta)} x_i}_{\text{Exact gradient term}} - \underbrace{\sum_{j=1}^{m} \tilde{x}_i^j}_{\text{Estimated gradient with sampler}} \Big|$$

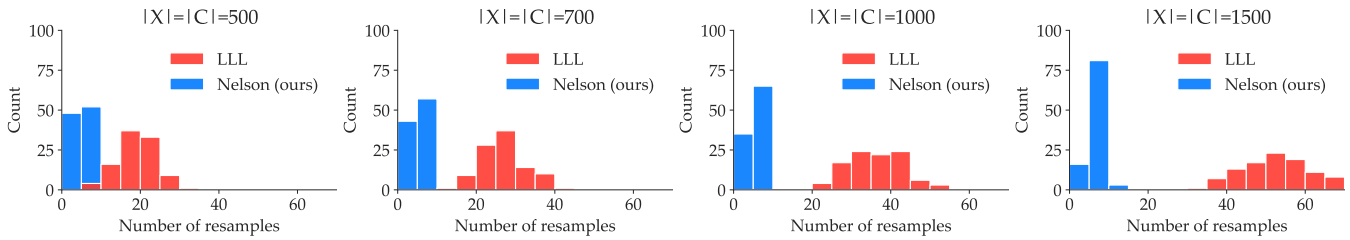
  For fixed parameter θ, the best sampler would attain the smallest approximation error.

D.7 Hyper-parameter Settings

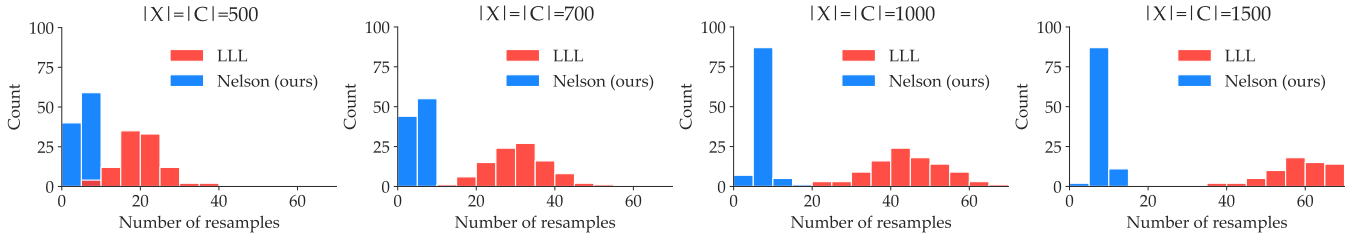
In the implementation of NELSON, we set the maximum tryout of resampling as $T_{tryout} = 1000$ for all the experiments and all the datasets.

For the hyper-parameters used in learning the constrained MRF, we set the number of samples from the model to be $m = 200$, the learning rate η is configured as 0.1 and the total learning iterations are $T_{\max} = 1000$.

[11] https://github.com/meelgroup/unigen
[12] https://github.com/RafaelTupynamba/quicksampler
[13] https://github.com/meelgroup/cmsgen
[14] https://github.com/meelgroup/KUS
[15] http://reasoning.cs.ucla.edu/ace/moreInformation.html



(a) Uniform Case.



(b) Weighted Case.

Figure 6: The distribution of resampling steps in the NELSON and Algorithmic-LLL (Moser and Tardos 2010). Both of them get a valid sample within $T_{tryouts}$. NELSON takes much fewer resamples than Algorithmic-LLL because it resamples all the violated clauses at every iteration while Algorithmic-LLL only resamples one of them.