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.

Introduction

In recent years, tremendous progress has been made in generative modeling (**?????????????**).

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 (**??????**). Nevertheless, reasoning and learning have been growing independently for a long time. Only recently do ideas emerge exploring the role of reasoning in learning (**??????**).

The Lovász Local Lemma (LLL) (**?**) 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, **?** came up with an algorithm, which samples from the probability distribution proven to exist by LLL. **?** 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 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 (**???**). Machine learning is also used to discover better heuristics (**??**). Reinforcement learning (**??**) as well as approaches integrating search with neural nets (**?**) 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 (**???**) and various versions of belief propagation (**????**). SampleSearch (**?**) integrates importance sampling with constraint-driven search. Probabilistic

[*]These authors contributed equally.

inference based on hashing and randomization obtains probabilistic guarantees for marginal queries and sampling via querying optimization oracles subject to randomized constraints (**????**).

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].

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) (**?**). 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 **?**. 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

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

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 **?** 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 **?**.

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 $c_j(x) = 1$.*

Sampling Through Lovász Local Lemma

Lovász Local Lemma (LLL) (**?**) 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 $Z_{\mathcal{C}}(\theta) = \sum_{x' \in \mathcal{X}} \exp\left(\sum_{i=1}^n \theta_i x_i\right) C(x)$.

As shown in Algorithm 1, the LLL-based sampler (**?**) 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 (**?**).

Algorithm 1: Sampling Through Lovász Local Lemma.

Require: 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 $P_\theta(X = x|\mathcal{C})$.

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.

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.

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:*

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

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

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.

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 $\theta^{T_{\max}}$.

Algorithm 2: Learn Constrained MRFs via NELSON-CD.

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

```
NELSON(W, b, V) ← build(X, C).                    ▷ in Sec.
for t = 1 to T_max do
    {x^j}_{j=1}^m ∼ D.                            ▷ from data
    {x̃^j}_{j=1}^m ← NELSON(θ^t, m).               ▷ from model
    g^t ← (1/m) Σ_{j=1}^m ∇φ(x^j) − ∇φ(x̃^j)       ▷ divergence
    θ^{t+1} ← θ^t − η g^t.                        ▷ update parameters
return The converged MRF model θ^{T_max}.
```

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)).

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 (**?**), which is a special case of MCMC that is widely used in training MRF models.

• Weighted SAT samplers, including WAPS (**?**), WeightGen (**?**) and XOR sampler (**??**).

• Uniform SAT samplers, including CMSGen (**?**), QuickSampler (**?**), UniGen (**?**) and KUS (**?**). 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.

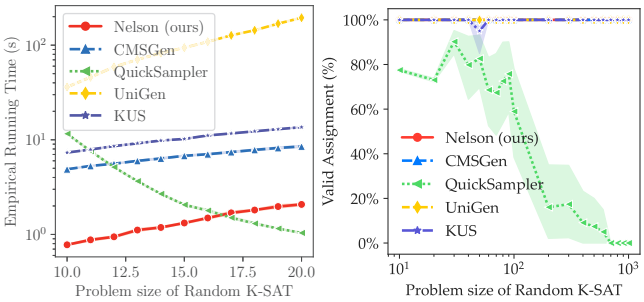

Figure 1: Running time and the percentage of valid structures sampled uniformly at random from solutions of K-SAT problems. Among all the problem sizes, NELSON always generate valid solutions and is the most efficient sampler.

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

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 $\mathcal{D}$ (**?**). To generate the training set $\mathcal{D}$, we use CNFGen (**?**) to generate the random K-SAT problem and use Glucose4 solver to generate random valid solutions (**?**).

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 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 **?**). In weighted sampling,

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 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	Gibbs	CMSGen	Quicksampler WeightGen,KUS XOR, WAPS
	(a) log-likelihood (↑)			
100	-49.16	$\mathbf{-36.36}$	-60.12	
300	$\mathbf{-52.61}$	-53.11	-128.39	
500	$\mathbf{-196.47}$	-197.21	-272.49	T.O.
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**	
300	**66.37**	64.42	64.50	
500	**90.03**	73.14	70.67	T.O.
700	**69.74**	**69.74**	48.10	
1000	**91.70**	77.56	78.72	

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.

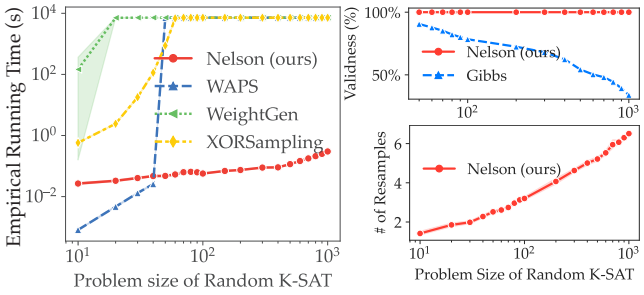

Figure 2: Running time, the percentage of valid solutions generated, and rounds of resampling for weighted sample generation of K-SAT solutions. Among all the problem sizes, NELSON scales the best among all approaches and always generates valid solutions.

the weights are specified 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.

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 (**?**). This task has wide applications in robotics routing and IoT network configuration (**?**). 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.

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,

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

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.

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.

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.

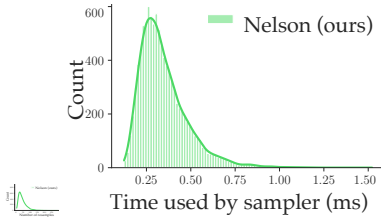

Figure 3: Frequency histograms for the number of resample and the total time of NELSON method for uniformly sampling visiting paths for vehicle routing problem.

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. Illum a vitae totam qui eos dolores molestias soluta corrupti, minus illo quaerat natus dolore tempora ratione dolorem, consequatur totam vel id hic a ratione sequi aliquam, eveniet deserunt quibusdam reiciendis accusantium aut id fugit?Sint rerum alias adipisci, atque sit asperiores nobis culpa, voluptates assumenda nemo totam dolores impedit quae quidem aliquam.Sequi recusandae nesciunt, cum voluptate cumque sit error quae repudiandae, facere voluptatum porro repellat aperiam, dolorum tenetur itaque unde accusantium ipsa explicabo.Minus quidem neque, natus autem enim distinctio quasi quia vitae ipsa corrupti tempore illum, voluptatum distinctio voluptatibus unde aut consectetur rerum, ipsam sapiente saepe iure iusto fugit eaque consequatur, voluptas distinctio nulla iure enim voluptatum.Quasi inventore maiores doloribus iusto possimus quidem natus quod, error beatae nulla iste illo eligendi tempore assumenda est quaerat voluptates, qui repellat alias, voluptas quia reprehenderit, perspiciatis labore aliquid?Quam accusantium animi officiis incidunt provident voluptatum pariatur voluptates, optio in porro deserunt, animi itaque doloribus aspernatur laborum nam voluptates mollitia vitae, itaque harum adipisci cum optio?Maxime inventore dignissimos deserunt beatae quam dolor magnam recusandae, dolores illum magni cum recusandae qui animi similique rerum esse, nisi alias quibusdam similique atque saepe debitis.Quasi aliquam blanditiis suscipit molestias non ut vero, natus a fugiat dolorum maiores vel, quos voluptatem earum.Soluta odit a impedit labore quisquam consequatur est nulla blanditiis, laudantium modi eos sed saepe repellen-

dus maiores, architecto quasi veritatis veniam illum consequuntur laboriosam