

---

# LSB: Local Self-Balancing MCMC in Discrete Spaces

---

Emanuele Sansone<sup>1</sup>

## Abstract

We present the Local Self-Balancing sampler (LSB), a local Markov Chain Monte Carlo (MCMC) method for sampling in purely discrete domains, which is able to autonomously adapt to the target distribution and to reduce the number of target evaluations required to converge. LSB is based on (i) a parametrization of locally balanced proposals, (ii) a newly proposed objective function based on mutual information and (iii) a self-balancing learning procedure, which minimises the proposed objective to update the proposal parameters. Experiments on energy-based models and Markov networks show that LSB converges using a smaller number of queries to the oracle distribution compared to recent local MCMC samplers.

## 1. Introduction

Sampling from complex and intractable probability distributions is of fundamental importance for learning and inference (MacKay, 2003). In this regard, MCMC methods are promising solutions to tackle the intractability of sampling in high dimensions. They have been successfully applied in several domains, including Bayesian statistics and statistical physics (Neal, 1993; Robert & Casella, 2013), bioinformatics and computational biology (Altekar et al., 2004; Alexeev et al., 2020) as well as machine learning (Andrieu et al., 2003; Koller & Friedman, 2009; Nijkamp et al., 2020).

While MCMC is a general method, which can be used to sample from any target distribution, its efficiency strongly depends on the choice of the proposal. Indeed, the proposal must be adapted to the distribution we want to sample from (Andrieu & Thoms, 2008; Hoffman & Gelman, 2014) and machine learning can help to automate this process (Zhang et al., 2012; Pakman & Paninski, 2013; Af-

shar et al., 2015; Dinh et al., 2017; Nishimura et al., 2020). However, less effort has been devoted to devise strategies for adapting the proposal distribution to a discrete target. Most common solutions embed the discrete problem into a continuous one, thus allowing to reuse sampling strategies designed for the continuous domain. However, these strategies are not always optimal, as they either require specific analytic forms for the target (Zhang et al., 2012), or simply because the embeddings do not capture the topological properties of the original domain (Pakman & Paninski, 2013; Afshar et al., 2015; Dinh et al., 2017; Nishimura et al., 2020). We can avoid these issues by sampling directly in the discrete domain and using local MCMC strategies (Zanella, 2020).

In this work, we propose an adaptation strategy for local MCMC. Specifically, we introduce (i) a new parametrization for locally balanced proposals, (ii) a new objective function based on mutual information to compute the statistical dependence between consecutive samples, and (iii) a learning procedure to adapt the parameters of the proposal to the target distribution using our proposed objective. The resulting procedure, called the Local Self-Balancing sampler (LSB), automatically discovers a locally balanced proposal with the advantage of reducing the number of target evaluations required to converge. Furthermore, we show that our framework generalizes over two recent works (Zanella, 2020) and (Grathwohl et al., 2021). We conduct experiments on energy-based models and Markov network, and show that LSB queries the oracle distribution in a more efficient way compared to recent local MCMC samplers.

In summary, the key contributions of the work are:

- A parametrization of locally balanced proposals, which enable to convert the problem of proposal selection into a learning problem.
- The use of mutual information as a criterion to learn the proposal distribution and accelerate MCMC.
- An estimator to efficiently compute the mutual information objective and to enable efficient learning.

We start by providing some background on locally balanced proposal distributions (Section 2), we introduce LSB by describing the parametrizations, the objective and the learning

---

<sup>1</sup>Department of Computer Science, KU Leuven, Belgium. Correspondence to: Emanuele Sansone <emanuele.sansone@kuleuven.be>.

procedure (Section 3), we discuss the related work (Section 4) and the experiments (Section 5), and finally we conclude by highlighting the main limitations of LSB and the possible future directions (Section 6).

## 2. Background

We consider the problem of sampling from a distribution  $p$  with a support defined over a large finite discrete sample space  $\mathcal{X}$ , i.e.  $p(\mathbf{x}) = \tilde{p}(\mathbf{x}) / \sum_{\mathbf{x}'' \in \mathcal{X}} \tilde{p}(\mathbf{x}'')$ , where the normalization term cannot be tractably computed and only  $\tilde{p}$  can be evaluated. One solution to the problem consists of sampling using MCMC (Neal, 1993). The main idea of MCMC is to sequentially sample from a tractable surrogate distribution, alternatively called proposal, and to use an acceptance criterion to ensure that generated samples are distributed according to the original distribution. More formally, MCMC is a Markov chain with a transition probability of the form:<sup>1</sup>

$$T(\mathbf{x}'|\mathbf{x}) = A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x}) \quad (1)$$

where  $Q(\mathbf{x}'|\mathbf{x})$  is the probability of sampling  $\mathbf{x}'$  given a previously sampled  $\mathbf{x}$ , namely the proposal distribution, and  $A(\mathbf{x}', \mathbf{x})$  is the probability of accepting sample  $\mathbf{x}'$  given  $\mathbf{x}$ , e.g.  $A(\mathbf{x}', \mathbf{x}) = \min \left\{ 1, \frac{\tilde{p}(\mathbf{x}')Q(\mathbf{x}|\mathbf{x}')}{\tilde{p}(\mathbf{x})Q(\mathbf{x}'|\mathbf{x})} \right\}$ .<sup>2</sup> In this work, we consider the family of locally informed proposals (Zanella, 2020), which are characterized by the following expression:

$$Q(\mathbf{x}'|\mathbf{x}) = \frac{g\left(\frac{\tilde{p}(\mathbf{x}')}{\tilde{p}(\mathbf{x})}\right)1[\mathbf{x}' \in N(\mathbf{x})]}{Z(\mathbf{x})} \quad (2)$$

where  $N(\mathbf{x})$  is the neighborhood of  $\mathbf{x}$  based on the Hamming metric.<sup>3</sup>

Note that the choice of  $g$  has a dramatic impact on the performance of the Markov chain, as investigated in (Zanella, 2020). In fact, there is a family of functions called *balancing functions*, satisfying the relation  $g(t) = tg(1/t)$  (for all  $t > 0$ ), which have extremely desirable properties, briefly recalled hereunder.

**Acceptance rate.** The balancing property allows to rewrite the acceptance function in a form that depends only from the ratio  $\frac{Z(\mathbf{x})}{Z(\mathbf{x}'')}$ , specifically  $A(\mathbf{x}', \mathbf{x}) = \min \left\{ 1, \frac{Z(\mathbf{x})}{Z(\mathbf{x}'')} \right\}$ . Therefore if  $\tilde{p}(\mathbf{x})$  is smooth enough, then the ratio  $\frac{Z(\mathbf{x})}{Z(\mathbf{x}'')}$  will be close to 1, as  $\mathbf{x}, \mathbf{x}'$  are close to each other. However, a proper choice of  $g$  can additionally influence the value of this ratio.

<sup>1</sup>For all  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$  such that  $\mathbf{x} \neq \mathbf{x}'$ , see Eq. (7) in Appendix B

<sup>2</sup>Other choices are available (Neal, 1993) as well.

<sup>3</sup>In this work,  $N(\mathbf{x})$  is a set of points having Hamming distance 1 from  $\mathbf{x}$ .

**Detailed balance.** Note that for all  $\mathbf{x}' = \mathbf{x}$ , detailed balance trivially holds, viz.  $p(\mathbf{x})T(\mathbf{x}'|\mathbf{x}) = p(\mathbf{x}')T(\mathbf{x}|\mathbf{x}')$ . In all other cases, detailed balance can be proved, by exploiting the fact that  $T(\mathbf{x}'|\mathbf{x}) = A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})$  and by using the balancing property (see the Appendix A for more details). Detailed balance is a sufficient condition for invariance. Consequently, the target  $p$  is a fixed point of the Markov chain.

**Ergodicity.** Under mild assumptions, we have also ergodicity (we leave more detailed discussion to Appendix B). In other words, the Markov chain converges to the fixed point distribution  $p$  independently from its initialization.

**Efficiency.** The efficiency of MCMC is generally measured in terms of the resulting asymptotic variance for sample mean estimators. This is indeed a proxy to quantify the level of correlation between samples generated through MCMC. Higher levels of asymptotic variance correspond to higher levels of correlation, meaning that the Markov chain produces more dependent samples and it is therefore less efficient. Balancing functions are asymptotically optimal according to Peskun ordering (Zanella, 2020).

The work in (Zanella, 2020) proposes a pool of balancing functions with closed-form expression together with some general guidelines to choose one. However, this pool is only a subset of the whole family of balancing functions and several cases do not even have an analytical expression. Consequently, it is not clear which function to use in order to sample efficiently from the target distribution. Indeed, we will see in the experimental section that (i) the optimality of the balancing function depends on the target distribution and that (ii) in some cases the optimal balancing function may be different from the ones proposed in (Zanella, 2020). In the next sections, we propose a strategy to automatically learn the balancing function from the target distribution, thus reducing the number of target evaluations in order to achieve convergence compared to recent discrete MCMC samplers.

## 3. LSB: Local Self-Balancing Strategy

We start by introducing two different parametrizations for the family of balancing functions in increasing order of functional expressiveness. Then, we propose an objective criterion based on mutual information to learn the parametrization and to reduce the number of steps required to converge. Finally, we introduce an approximate strategy to further reduce the computational overhead of each step.

### 3.1. Parametrizations

We state the following proposition and then use it to devise the first parametrization.

**Proposition 3.1.** Given  $n$  balancing functions  $\mathbf{g}(t) = [g_1(t), \dots, g_n(t)]^T$  and any vector of scalar weights  $\mathbf{w} = [w_1, \dots, w_n]^T$ , the linear combination  $g(t) \doteq \mathbf{w}^T \mathbf{g}(t)$  satisfies the balancing property.

$$\text{Proof. } g(t) = \mathbf{w}^T \mathbf{g}(t) = \sum_{i=1}^n w_i g_i(t) = t \sum_{i=1}^n w_i g_i(1/t) = t \mathbf{w}^T \mathbf{g}(1/t) = t g(1/t) \quad \square$$

Despite its simplicity, the proposition has important implications. First of all, it allows to convert the problem of choosing the optimal balancing function into a learning problem. Secondly, the linear combination introduces only few parameters (in the experiments we consider  $n = 4$ ) and therefore the learning problem can be solved in an efficient way. Additionally, we consider only positive weights to avoid negative values in Eq. 2, thus having a valid proposal distribution.

The first parametrization (LSB 1) consists of the relations  $w_i = e^{\theta_i} / \sum_{j=1}^n e^{\theta_j}$  for all  $i = 1, \dots, n$ , where  $\boldsymbol{\theta} = [\theta_1, \dots, \theta_n] \in \mathbb{R}^n$ . Note that the softmax is used to smoothly select one among the  $n$  balancing functions. Therefore, we refer to this parametrization as learning to select among existing balancing functions.

The second parametrization (LSB 2) is obtained from the following proposition.

**Proposition 3.2.** Given  $g_{\boldsymbol{\theta}}(t) \doteq \frac{h_{\boldsymbol{\theta}}(t)}{2} + \frac{th_{\boldsymbol{\theta}}(1/t)}{2}$ , where  $h_{\boldsymbol{\theta}}$  is any real valued function parameterized by vector  $\boldsymbol{\theta} \in \mathbb{R}^k$  (e.g. a neural network),  $g_{\boldsymbol{\theta}}(t)$  satisfies the balancing property. Furthermore, for any balancing function  $\ell$ , there always exists  $\tilde{\boldsymbol{\theta}} \in \mathbb{R}^k$  such that  $g_{\tilde{\boldsymbol{\theta}}}(t) = \ell(t)$  for all  $t > 0$ . In other words, LSB 2 parameterizes all possible balancing functions.

*Proof.* The first part follows directly from the definition of the balancing property.

Given any balancing function  $\ell$ , we can always find a  $\tilde{\boldsymbol{\theta}}$  such that  $h_{\tilde{\boldsymbol{\theta}}}(t) = \ell(t)$  for all  $t > 0$  (because  $h_{\boldsymbol{\theta}}$  is a universal function approximator (Hornik et al., 1989)). This implies that  $h_{\tilde{\boldsymbol{\theta}}}$  satisfies the balancing property, i.e.  $h_{\tilde{\boldsymbol{\theta}}}(t) = th_{\tilde{\boldsymbol{\theta}}}(1/t)$  for all  $t > 0$ . Consequently, by the first part of the proposition, we have that  $g_{\tilde{\boldsymbol{\theta}}}(t) = h_{\tilde{\boldsymbol{\theta}}}(t)$ . Finally, we can conclude that  $g_{\tilde{\boldsymbol{\theta}}}(t) = \ell(t)$  for all  $t > 0$ .  $\square$

The proposition enables to parameterize the whole family of balancing functions, thus generalizing the result obtained for LSB 1. Note that the increased level of expressiveness of LSB 2 comes at the cost of a higher computation. However, in practice we can always trade-off expressiveness and computation by specifying the capacity of the function approximator (e.g. the hyperparameters of the neural network). We leave this discussion to the experimental section.

In the next paragraphs, we propose an objective and a learning strategy to train the parameters of LSB 1 and LSB 2.

### 3.2. Objective and Learning Algorithm

The goal here is to devise a criterion to find the balancing function reducing the number of target likelihood evaluations and increasing the convergence/mixing rate to distribution  $p$ .

As already mentioned in previous section, MCMC based on locally informed proposal converges to the true target distribution independently of its initialization. Furthermore, the rate of convergence and mixing is controlled by the balancing function. Importantly, we can speedup convergence and mixing by minimizing the amount of statistical dependence between consecutive samples. To this purpose, we introduce the following mutual information-based criterion:

$$\mathcal{I}_{\boldsymbol{\theta}} = KL\{p(\mathbf{x})T_{\boldsymbol{\theta}}(\mathbf{x}'|\mathbf{x})\|p(\mathbf{x})p(\mathbf{x}')\} \quad (3)$$

where  $KL$  is the Kullback Leibler divergence and  $T_{\boldsymbol{\theta}}$  is the transition probability with explicit dependence on parameter  $\boldsymbol{\theta}$ . Now, we are ready to highlight some properties of Eq. 3 with the following theorem (see Appendix C for the proof).

**Theorem 3.3.** Consider  $\mathcal{X} = \{0, 1\}^d$  and  $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/\Gamma$ , where  $\Gamma$  is a normalizing constant. Define two auxiliary distributions  $Q_1$  and  $Q_2$ , such that for all  $\mathbf{x} \in \mathcal{X}$ ,  $Q_1(\mathbf{x}) > 0$  whenever  $\tilde{p}(\mathbf{x}) > 0$ , and for all  $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ ,  $Q_2(\mathbf{x}') > 0$  whenever  $Q(\mathbf{x}'|\mathbf{x}) > 0$ . Therefore,

(a) if  $\mathcal{M}(\mathbf{x}) \doteq 1 - \sum_{\mathbf{x}'' \in \mathcal{N}(\mathbf{x})} A(\mathbf{x}'', \mathbf{x})Q(\mathbf{x}''|\mathbf{x})$ , then

$$\begin{aligned} \mathcal{L}_{\boldsymbol{\theta}} = & E_{\substack{\mathbf{x} \sim Q_1 \\ \mathbf{x}' \sim Q_2}} \left\{ \frac{\tilde{p}(\mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{Q_1(\mathbf{x})Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \cdot \right. \\ & \left. \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} \right\} \\ & + E_{\mathbf{x} \sim Q_1} \left\{ \frac{\tilde{p}(\mathbf{x})}{Q_1(\mathbf{x})} \mathcal{M}(\mathbf{x}) \log \frac{\mathcal{M}(\mathbf{x})}{\tilde{p}(\mathbf{x})} \right\} \quad (4) \end{aligned}$$

$$\text{and } \mathcal{I}_{\boldsymbol{\theta}} = \frac{\mathcal{L}_{\boldsymbol{\theta}}}{\Gamma} + \frac{\log \Gamma}{\Gamma}.$$

(b) if  $\mathcal{M}(\mathbf{x}) \doteq 1 - A(\mathbf{x}^*, \mathbf{x})Q(\mathbf{x}^*|\mathbf{x})$ , where  $\mathbf{x}^*$  is randomly sampled according to a uniform distribution over  $\mathcal{N}(\mathbf{x})$ , then

$$\begin{aligned} \mathcal{L}_{\boldsymbol{\theta}} = & E_{\substack{\mathbf{x} \sim Q_1 \\ \mathbf{x}' \sim Q_2}} \left\{ \frac{\tilde{p}(\mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{Q_1(\mathbf{x})Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \cdot \right. \\ & \left. \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} \right\} \\ & + E_{\mathbf{x} \sim Q_1} \left\{ \frac{\mathcal{M}(\mathbf{x})}{Q_1(\mathbf{x})} [\eta \mathcal{M}(\mathbf{x}) - \tilde{p}(\mathbf{x})(\log \eta + 1)] \right\} \quad (5) \end{aligned}$$

$$\text{and } \mathcal{I}_{\boldsymbol{\theta}} \leq \frac{\mathcal{L}_{\boldsymbol{\theta}}}{\Gamma} + \frac{\log \Gamma}{\Gamma}, \text{ for } \eta \in \mathbb{R}^+.$$

**Algorithm 1** Local Self-Balancing (LSB)

**Input:** Learning rate  $\gamma = 1e - 2$ ,  $\pi = 1e - 8$ , initial parameter  $\theta_0$ , burn-in iterations  $K$  and batch of samples  $N$   
 $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^N \sim U_{\mathcal{X}}$   
**for**  $k = 1$  **to**  $K$  **do**  
 $\{\mathbf{x}^{(i)}\}_{i=1}^N \sim Q_1$   
 $\{\mathbf{x}'^{(i)}\}_{i=1}^N \sim Q_2$   
 $\hat{\mathcal{L}}_{\theta} \leftarrow$  Estimate  $\mathcal{L}_{\theta}$  using  $\{\mathbf{x}^{(i)}\}_{i=1}^N, \{\mathbf{x}'^{(i)}\}_{i=1}^N$   
 $\theta \leftarrow \theta - \frac{\gamma}{N} \nabla_{\theta} \hat{\mathcal{L}}_{\theta}$   
 Update  $\eta$   
 Accept/Reject samples  
 Update  $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^N$  with accepted samples  
 $\theta_0 \leftarrow \theta$   
**end for**

The theorem tells that for case (a),  $\mathcal{L}_{\theta}$  is equal to  $\mathcal{I}_{\theta}$  up to a constant, whereas, for case (b)  $\mathcal{L}_{\theta}$  is an upper bound of  $\mathcal{I}_{\theta}$ . In both cases, we can use  $\mathcal{L}_{\theta}$  as a surrogate objective to minimize Eq. 3. However, note that computing the two expectations in Eq. 4 or Eq. 5 is generally intractable, but one can estimate these two quantities by using Monte Carlo, for instance by sampling first from  $Q_1$  and then sampling from  $Q_2$ . Also, note that the conditions on  $Q_1$  and  $Q_2$  in the theorem require that  $Q_1$  and  $Q_2$  are positive on the support of  $p(\mathbf{x})$  and  $Q(\mathbf{x}'|\mathbf{x})$ , respectively. These conditions can be met by choosing  $Q_1(\mathbf{x}) = \pi U_{\mathcal{X}}(\mathbf{x}) + (1-\pi)\delta(\mathbf{x}-\hat{\mathbf{x}})$ , where  $U$  is a uniform distribution over the whole space  $\mathcal{X}$ ,  $\delta$  is the Dirac delta function,  $\hat{\mathbf{x}}$  is the last accepted sample and  $\pi$  defines the proportions of the mixture, and  $Q_2(\mathbf{x}') = Q_{\theta_0}(\mathbf{x}'|\mathbf{x})$ , where  $Q_{\theta_0}(\mathbf{x}'|\mathbf{x})$  is the proposal distribution with parameter vector  $\theta_0$ . Importantly, while the choice of  $Q_2$  is quite natural, we choose to use a mixture distribution including a deterministic and a uniform distribution for  $Q_1$ , as the deterministic distribution remembers and exploits the previous iterate in the current sampling step, whereas the uniform distribution discards this information and acts as an uninformative prior on the support of  $p(\mathbf{x})$ , thus enabling the exploration of the whole input domain. Therefore, the mixing proportion parameter  $\pi$  balances between exploration and exploitation. In order to learn a balancing function, there is no real need to have exploration ( $\pi$  is set to  $1e - 8$  in all our experiments). However, the mixing parameter and the exploration term might play a more important role in other learning settings, for instance when learning "global" proposal distributions. Once  $\mathcal{L}_{\theta}$  is estimated, we can update the parameters of the two parametrizations by using an off-the-shelf gradient-based optimizer.

In our training algorithm, we choose case (b) as our surrogate objective to minimize the amount of target likelihood evaluations. Indeed, note that the estimate for case (a) requires  $O(d^2)$  evaluations, because we need to com-

**Algorithm 2** Fast Local Self-Balancing (FLSB).

**Input:** Learning rate  $\gamma = 1e - 2$ ,  $\pi = 1e - 8$ , initial parameter  $\theta_0$ , burn-in iterations  $K$  and batch of samples  $N$   
 $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^N \sim U_{\mathcal{X}}$   
**for**  $k = 1$  **to**  $K$  **do**  
 $\{\mathbf{x}^{(i)}\}_{i=1}^N \sim Q_1$   
 $\{\mathbf{x}'^{(i)}\}_{i=1}^N \sim Q_2$  (using the approximation)  
 $\hat{\mathcal{L}}_{\theta} \leftarrow$  Estimate  $\mathcal{L}_{\theta}$  using  $\{\mathbf{x}^{(i)}\}_{i=1}^N, \{\mathbf{x}'^{(i)}\}_{i=1}^N$   
 $\theta \leftarrow \theta - \frac{\gamma}{N} \nabla_{\theta} \hat{\mathcal{L}}_{\theta}$   
 Update  $\eta$   
 Accept/Reject samples (exact accept. score)  
 Update  $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^N$  with accepted samples  
 $\theta_0 \leftarrow \theta$   
**end for**

pute  $Q(\mathbf{x}''|\mathbf{x})$  for all  $\mathbf{x}'' \in N(\mathbf{x})$ , whereas the one for case (b) requires only  $O(d)$  evaluations. Finally, we learn the additional parameter  $\eta$  in case (b) using standard gradient descent. The whole learning procedure is shown in Algorithm 1.

### 3.3. Constant Target Evaluation

Without loss of generality, we can express  $\tilde{p}(\mathbf{x}) = e^{f(\mathbf{x})}$  for some  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  and consider  $\mathcal{X} = \{0, 1\}^d$ .<sup>4</sup> By definition of  $N(\mathbf{x})$ , the proposal distribution in Eq. 2 can be written in an equivalent form as  $Q(\mathbf{x}'|\mathbf{x}) = \sum_{i=1}^d Q(\mathbf{x}'|\mathbf{x}, i)Q(i|\mathbf{x})$ , where  $Q(\mathbf{x}'|\mathbf{x}, i) = \delta(\mathbf{x}' - \mathbf{x}_{-i})$  with  $\mathbf{x}_{-i}$  obtained by flipping  $i$ -bit in  $\mathbf{x}$ , and

$$Q(i|\mathbf{x}) = \text{Cat} \left\{ \text{Norm} \left[ g(e^{df_1(\mathbf{x})}), \dots, g(e^{df_d(\mathbf{x})}) \right] \right\}_i$$

where Cat stands for a categorical distribution, Norm is a normalization operator acting on a  $d$ -dimensional vector and  $df_i(\mathbf{x}) = f(\mathbf{x}_{-i}) - f(\mathbf{x})$ . It's clear that computing previous equation requires to evaluate  $f$  for  $O(d)$  times. However, if we assume that  $f$  is known and differentiable (which is true for instance in energy-based models), we can use Taylor expansion to approximate the difference  $df_i(\mathbf{x})$ . Indeed, we have that

$$f(\mathbf{x}_{-i}) - f(\mathbf{x}) \approx \nabla_{\mathbf{x}} f(\bar{\mathbf{x}})|_{\bar{\mathbf{x}}=\mathbf{x}}^T (\mathbf{x}_{-i} - \mathbf{x}) \doteq \bar{df}_i(\mathbf{x})$$

which allows to evaluate the target only  $O(1)$  times. Therefore, our new proposal distribution is defined as follows:

$$Q(i|\mathbf{x}) = \text{Cat} \left\{ \text{Norm} \left[ g(e^{\bar{df}_1(\mathbf{x})}), \dots, g(e^{\bar{df}_d(\mathbf{x})}) \right] \right\}_i$$

Interestingly, if we choose  $g(t) = \sqrt{t}$ , we recover the exact same proposal distribution of a recent sampler, called Gibbs-With-Gradients (Grathwohl et al., 2021). Thanks to this

<sup>4</sup> $\tilde{p}(\mathbf{x})$  is obviously defined over a discrete sample space. However, we can always identify a real-valued function which coincides with  $\log \tilde{p}(\mathbf{x})$  on the discrete support.



Table 1. Summary comparing locally balanced proposals (LB) of (Zanella, 2020), Gibbs-With-Gradients (GWG) (Grathwohl et al., 2021) and our samplers (LSB and FLSB).

Name	$g(t)$	$df_i$	Eval./step
LB	Fixed	Exact	$O(d)$
GWG	Fixed ( $g(t) = \sqrt{t}$ )	Approx.*	$O(1)$
LSB	Learnt	Exact	$O(d)$
FLSB	Learnt	Approx.*	$O(1)$

\* Assumption:  $f$  is known and differentiable.

approximation, we can propose a more efficient version of Algorithm 1, called Fast Local Self-Balancing procedure (FLSB), shown in Algorithm 2.

Finally, we can summarize the main differences among locally balanced proposals of (Zanella, 2020), Gibbs-With-Gradients (Grathwohl et al., 2021) and our samplers in Table 1.

## 4. Related Work

It’s important to devise strategies, which enable the automatic adaption of proposals to target distributions, not only to reduce user intervention, but also to increase the efficiency of MCMC samplers (Andrieu & Thoms, 2008; Hoffman & Gelman, 2014). Recently, there has been a surge of interest in using machine learning and in particular deep learning to learn proposals directly from data, especially in the continuous domain. Here, we provide a brief overview of recent integrations of machine learning and MCMC samplers according to different parametrizations and training objectives.

**Parametrizations and objectives in the continuous domain.** The work in (Wang et al., 2018) proposes a strategy based on block Gibbs sampling, where blocks are large motifs of the underlying probabilistic graphical structure. It parameterizes the conditional distributions of each block using mixture density networks and trains them using meta-learning on a log-likelihood-based objective. The work in (Song et al., 2017) considers a global sampling strategy, where the proposal is parameterized by a deep generative model. The model is learnt through adversarial training, where a neural discriminator is used to detect whether or not generated samples are distributed according to the target distribution. Authors in (Habib & Barber, 2018) propose a global sampling strategy based on MCMC with auxiliary variables (Higdon, 1998). The proposals are modelled as Gaussian distributions parameterized by neural networks and are trained on a variational bound of a log-likelihood-based objective. The works in (Levy et al., 2018; Gong et al., 2019) propose a gradient-based MCMC (Duane et al., 1987; Grenander & Miller, 1994), where neural models are used to

learn the hyperparameters of the equations governing the dynamics of the sampler. Different objectives are used during training. In particular, the work in (Gong et al., 2019) uses a log-likelihood based objective, whereas the work in (Levy et al., 2018) considers the expected squared jump distance, namely a tractable proxy for the lag-1 autocorrelation function (Pasarica & Gelman, 2010). The work in (Zhu, 2019) proposes a global two-stage strategy, which consists of (i) sampling according to a Gaussian proposal and (ii) updating its parameters using the first- and second-order statistics computed from a properly maintained pool of samples. The parameter update can be equivalently seen as finding the solution maximizing a log-likelihood function defined over the pool of samples. Finally, the work in (Pompe, E. and Holmes, C. and Łatuszyński, K. and others, 2020) extends this last strategy to the case of Gaussian mixture proposals. All these works differ from the current one in at least two aspects. Firstly, it is not clear how these parametrizations can be applied to sampling in the discrete domain. Secondly, the proposed objectives compute either a distance between the proposal distribution and the target one, namely using an adversarial objective or a variational bound on the log-likelihood, or a proxy on the correlation between consecutive generated samples, namely the expected squared jump distance. Instead, our proposed objective is more general in the sense that it reduces the statistical dependence between consecutive samples, as being closely related to mutual information.

**Sampling in the discrete domain.** Less efforts have been devoted to devise sampling strategies for a purely discrete domain. Most of the works consider problem relaxations by embedding the discrete domain into a continuous one, applying existing strategies like Hamiltonian Monte Carlo (Zhang et al., 2012; Pakman & Paninski, 2013; Afshar et al., 2015; Dinh et al., 2017; Nishimura et al., 2020) on it and then moving back to the original domain. These strategies are suboptimal, either because they consider limited settings, where the target distribution has specific analytic forms (Zhang et al., 2012), or because they make strong assumptions on the properties of the embeddings, thus not preserving the topological properties of the discrete domain (Pakman & Paninski, 2013; Afshar et al., 2015; Dinh et al., 2017; Nishimura et al., 2020).<sup>5</sup> The work in (Zanella, 2020) provides an extensive experimental comparison between several discrete sampling strategies, including the ones based on embeddings, based on stochastic local search (Hans et al., 2007) and the Hamming ball sampler (Titsias & Yau, 2017), which can be regarded as a more efficient version of block Gibbs sampling. Notably, the sampling strategy based on locally informed proposals and balancing functions proposed in (Zanella,

<sup>5</sup>For example by considering transformations that are bijective and/or by proposing transformations which allow to tractably compute the marginal distribution on the continuous domain.



Figure 1. Examples of  $\alpha$  in different settings of the Ising model ( $30 \times 30$ ), i.e noisy  $\mu = 1, \sigma = 3$  and clean  $\mu = 3, \sigma = 3$ .

2020) can be considered as the current state of the art for discrete MCMC. Our work builds and extends upon this sampler by integrating it with a machine learning strategy.

It’s important to mention that there are a couple of neural approaches applied to the discrete domain. The first one (Jaini et al., 2021) proposes to use normalizing flows to (i) learn a continuous relaxation of the discrete domain by dequantizing input data, and to (ii) learn a latent embedding more amenable to MCMC sampling. Learning the input-latent transformation is performed by maximizing the log-likelihood computed on data sampled from the latent space. The second one (Dai et al., 2020) proposes a strategy to learn an initializing distribution for a fixed local discrete MCMC sampler in the context of energy-based models. This is achieved by forcing the distribution to be close enough (in terms of KL divergence) to the fixed point of the MCMC kernel. Both works differ from ours in at least three aspects. Indeed, our work parameterizes the kernel of a local MCMC sampler, while the others consider a more global approach. Secondly, we are learning a one-dimensional real valued function through a simple neural network, instead of learning a more complex deep latent variable model which must transform input data. Finally, we are providing a mutual information objective, which directly tackle the problem of reducing the statistical dependence, and therefore also correlation, of consecutive samples.

## 5. Experiments

Firstly, we analyze samplers’ performance on energy-based models, including the 2D Ising model and Restricted Boltzmann Machines. Then, we perform experiments on additional UAI benchmarks. Code to replicate the experiments is available at <https://github.com/emsansone/LSB.git>.

### 5.1. 2D Ising Model

We consider the Ising model applied to image segmentation, to identify an object from its background. Consider a binary state space  $\mathcal{X} = \{-1, 1\}^V$ , where  $(V, E)$  defines a square lattice graph of the same size of the analyzed image, namely  $n \times n$ . For each state configuration  $\mathbf{x} = (x_i)_{i \in V} \in \mathcal{X}$ ,

define a prior distribution

$$p_{prior}(\mathbf{x}) \propto \exp \left\{ \lambda \sum_{(i,j) \in E} x_i x_j \right\}$$

where  $\lambda$  is a non-negative scalar used to weight the dependence among neighboring variables in the lattice. Then, consider that each pixel  $y_i$  is influenced only by the corresponding hidden variable  $x_i$  and generated according to a Gaussian density with mean  $\mu x_i$  and variance  $\sigma^2$ . Note that each variable in the lattice tells whether the corresponding pixel belongs to the object or to the background (1 or -1, respectively). The corresponding posterior distribution of a hidden state  $\mathbf{x}$  given an observed image is defined as follows:

$$p(\mathbf{x}) = \frac{1}{Z} \exp \left\{ \sum_{i \in V} \alpha_i x_i + \lambda \sum_{(i,j) \in E} x_i x_j \right\} \quad (6)$$

where  $\alpha_i = y_i \mu / \sigma^2$  is a coefficient biasing  $x_i$  towards either 1 or  $-1$ . Therefore,  $\alpha = (\alpha_i)_{i \in V}$  contains information about the observed image. Figure 1 shows two synthetically generated examples of  $\alpha$ . We evaluate the sampling performance on the distribution defined in Eq. 6. Importantly, the topological graph structure of the lattice and the exponential form of the posterior distribution allows to compute a locally balanced proposals in  $O(1)$  target evaluations, without the need of using a gradient-based approximation (cf. Section 3.3). Therefore, we consider only comparisons between LB and LSB

**Learning the balancing function.** We consider the balancing functions proposed in (Zanella, 2020), namely  $g(t) = t/(1+t)$  (a.k.a Barker function),  $\sqrt{t}$ ,  $\min\{1, t\}$  and  $\max\{1, t\}$ .<sup>6</sup> We compare these four balancing functions with our two parametrizations on four different settings of the Ising model, namely independent and noisy  $(\lambda, \mu, \sigma) = (0, 1, 3)$ , independent and clean  $(\lambda, \mu, \sigma) = (0, 3, 3)$ , dependent and noisy  $(\lambda, \mu, \sigma) = (1, 1, 3)$  and dependent and clean  $(\lambda, \mu, \sigma) = (1, 3, 3)$  cases and show the corresponding performance in Figure 2 and Table 2. We leave additional details and results to the Appendices D and E.

From Figure 2, we can see that our first parametrization LSB 1 is able to always ”select” an unbounded balancing function during burn-in, while when approaching convergence it is able to adapt to preserve fast mixing, as measured by the effective sample size (ESS) in Table 2. It’s interesting to mention also that the softmax nonlinearity used in LSB 1 can sometimes slow down the adaptation due to vanishing

<sup>6</sup>As discussed in Section 3.3, the case  $g(t) = \sqrt{t}$  corresponds to the exact version of GWG. Here there is no need for using the approximation, as we can exploit the structure of the lattice and the exponential form of the distribution to achieve constant target evaluation.

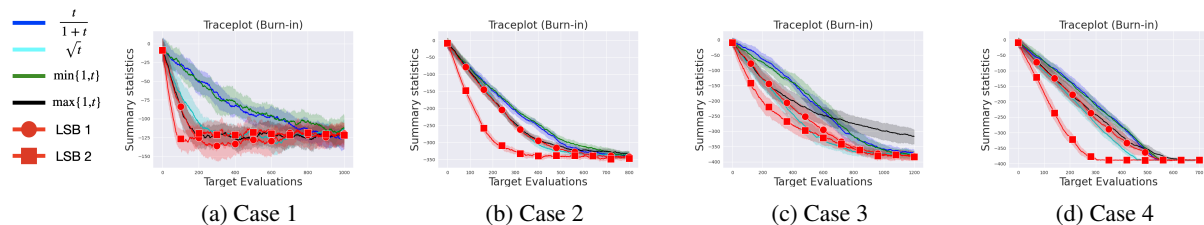


Figure 2. Samplers’ performance on four cases of the Ising model ( $30 \times 30$ ) for the burn-in phase. (a) Case 1: Independent-noisy, (b) case 2: Independent-clean, (c) case 3: Dependent-noisy, (d) case 4: Dependent-clean

Table 2. Quantitative performance for mixing measured by effective sample size on the four cases of the Ising model ( $30 \times 30$ ).  $\max\{1, t\}$  is performing significantly worse in statistical terms than the other functions.

Setting	$\frac{t}{1+t}$	$\sqrt{t}$	$\min\{1, t\}$	$\max\{1, t\}$	LSB 1	LSB 2
Case 1	<b><math>2.55 \pm 0.30</math></b>	<b><math>2.29 \pm 0.23</math></b>	<b><math>2.43 \pm 0.22</math></b>	$1.71 \pm 0.13$	<b><math>2.32 \pm 0.27</math></b>	<b><math>2.34 \pm 0.23</math></b>
Case 2	<b><math>3.30 \pm 0.36</math></b>	<b><math>2.89 \pm 0.28</math></b>	<b><math>2.96 \pm 0.30</math></b>	$1.68 \pm 0.11$	<b><math>2.85 \pm 0.28</math></b>	$2.30 \pm 0.29$
Case 3	<b><math>2.39 \pm 0.98</math></b>	<b><math>1.83 \pm 0.56</math></b>	<b><math>2.31 \pm 0.92</math></b>	$1.20 \pm 0.10$	<b><math>2.01 \pm 0.70</math></b>	<b><math>2.44 \pm 0.96</math></b>
Case 4	$2.10 \pm 0.91$	$7.08 \pm 4.07$	$1.74 \pm 0.26$	$1.74 \pm 0.51$	$2.52 \pm 1.11$	<b><math>20.11 \pm 15.58</math></b>

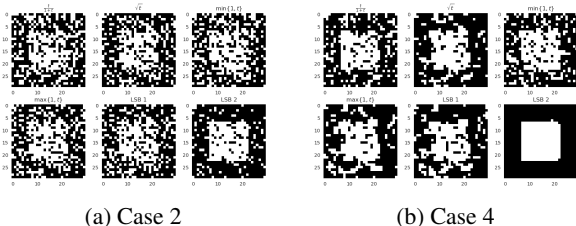


Figure 3. Realizations obtained after 300 burn-in iterations on the Ising model.

gradients. This can be observed by looking at the case 4 of Figure 2, where for a large part of the burn-in period the strategy prefers  $\max\{1, t\}$  over  $\sqrt{t}$ . Nevertheless, it is still able to recover a solution different from  $\max\{1, t\}$  at the end of burn-in, as confirmed by the larger effective sample size in Table 2 compared to the one achieved by  $\max\{1, t\}$ .

Furthermore, we observe that our second parametrization LSB 2, which is functionally more expressive compared to LSB 1, allows to outperform all previous cases in terms of number of target evaluations required to converge, as shown in Figure 2 and Table 2. This provides evidence that the optimality of the balancing function depends on the target distribution and that exploiting information about the target can lead to significant improvements (e.g. in case 3 of Figure 2, LSB 2 converges twice time faster as the best balancing function  $\sqrt{t}$ ). Figure 3 provides some realizations obtained by the samplers for the cases with dependent variables  $\lambda = 1$ . We clearly see from these pictures that convergence for LSB 2 occurs at an earlier stage than the other balancing functions and therefore the latent variables in the Ising model converge faster to their

ground truth configuration.

## 5.2. Sampling in Restricted Boltzmann Machines

We also evaluate the performance on the challenging task of sampling from Restricted Boltzmann Machines, using the experimental setup of (Grathwohl et al., 2021). In particular, we train a RBM with 250 hidden units on the binary MNIST dataset using contrastive divergence and use this model as our base distribution for evaluating the samplers. Notably, since the distribution has an analytic differentiable form, we can exploit the gradient-based approximation explained in Section 3.3. Therefore, we compare our sampler (FLSB) against Gibbs-With-Gradients (GWG) (Grathwohl et al., 2021), block Gibbs sampling (Gibbs 2) and the Hamming Ball sampler (HB-10-1) (Titsias & Yau, 2017).<sup>7</sup> We use two scores to measure the performance. The first one consists of the maximum mean discrepancy (MMD) distance between the generated samples and the ground truth ones, obtained through the block-Gibbs sampling procedure available in RBMs. The second one consists of the effective sample size (ESS) of a summary statistics computed on the sampling trajectories (more details about the experiments are available in Appendix D). We also provide the effective sample size normalized over time (ESS/sec).

In Figure 4, we see that FLSB 1 recovers the performance of GWG. FLSB 2 is able to adapt to the target distribution and to converge using a much smaller number of target likelihood evaluations. Furthermore, we observe that FLSB 2 performs well also in terms of ESS. These experiments

<sup>7</sup>Gibbs-X refers to Gibbs sampling with block size of X, whereas HB-X-Y refers to Hamming Ball sampler with block size of X and a Hamming ball of size Y.

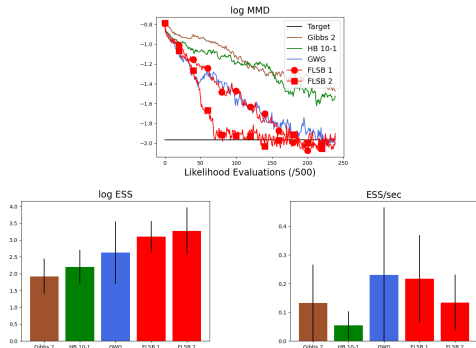


Figure 4. Samplers’ performance on RBMs. On the top, burn-in performance computed using MMD (in logarithmic scale). On the bottom, mixing performance computed after burn-in using  $\log(ESS)$  (on the left) and ESS per seconds (on the right).

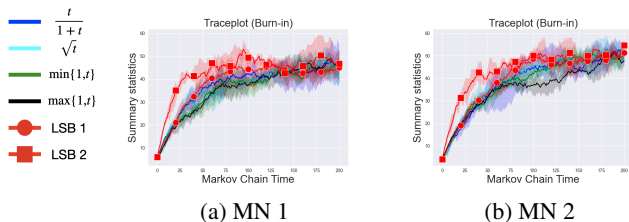


Figure 5. Samplers’ performance on Markov networks from UAI competition (100 variables). (a)-(b) are the traceplots for the burn-in phase.

provide further evidence that there is a clear advantage on learning the balancing function. However, it is important to mention that the improved performance comes with a computational overhead. Indeed, when comparing the samplers based on ESS/sec (Figure 4), we observe that, while FLSB 1 and GWG achieve comparable results, the performance of FLSB 2 are inferior on average. This is explained by the fact that each sampling iteration requires to evaluate a more complex balancing function (i.e. a multilayer perceptron network with one 10-neuron hidden layer, corresponding to 31 parameters vs. 4 parameters for FLSB 1). Clearly, there is a distinction between the number of evaluations of the likelihood and the number of evaluations for the balancing function. Our proposed strategy reduces the first ones and the introduced computational overhead affects only the second kind of evaluations. This can be mitigated for example by looking at more efficient block-wise implementations. However, this is left to future work. Additional comparisons of FLSB2 against block-wise sampling strategies together with a summary of its benefits are provided in Appendix F.

### 5.3. Markov Networks: UAI data

Lastly, we evaluate how our strategy generalizes to different graph topologies compared to the one of the Ising model.

In particular, we consider two Markov networks, with 100 discrete variables each and near-deterministic dependencies, from the 2006 UAI competition.<sup>8</sup> In this setting, we can’t leverage the gradient-based approximation of Section 3.3, as the distribution is specified in tabular form. Therefore, we compare our sampler LSB with the four balancing functions proposed in (Zanella, 2020). Similarly to previous experiments, we analyze the convergence of the burn-in phase (using traceplots) and the mixing performance according to ESS. Further details about the simulations are available in the Appendices D and G.

Also in this case, we observe that the proposed strategy is able to adapt to the target distribution and reduce the number of target likelihood evaluations required to converge (Figure 5). In this case, we observe similar performance in terms of ESS (cf. Table 3).

## 6. Conclusions and Future Work

We have presented a strategy to learn locally balanced proposals for MCMC in discrete spaces. The strategy consists of (i) a new parametrization of balancing functions and (ii) a learning procedure adapting the proposal to the target distribution. This allows to reduce the number of target likelihood evaluations required to converge. We believe that the proposed strategy can play an important role on applications where querying an oracle distribution is very expensive (like in deep energy-based models). We will investigate this in the future.

Note that the LSB sampler belongs to the family of local sampling strategies, thus inheriting their limitations. The locality assumption can be quite restrictive, for example when sampling from discrete distributions with deterministic dependencies among variables. In such situations, local sampling might fail to correctly sample from the target in a finite amount of time, as being required to cross regions with zero probability mass. The locality assumption can be relaxed for instance by leveraging a recent extension of the works in (Zanella, 2020; Grathwohl et al., 2021), called Path Auxiliary MCMC (Sun et al., 2022), which uses a trajectory of samples generated by the repeated application of the same base locally balanced proposal, and by adapting our proposed strategy to this new setting. Alternatively, we can consider block-wise implementations of locally balanced proposals, as also suggested in (Zanella, 2020). However, we leave all these extensions to future work.

To the best of our knowledge, we are the first to consider using mutual information to accelerate MCMC. We have focused on a local MCMC sampler for discrete domains. In the future, it will be interesting to see the application of the mutual information criterion to more global settings as well

<sup>8</sup><http://sli.ics.uci.edu/~ihler/uai-data/>



Table 3. Quantitative performance for mixing measured by effective sample size on two Markov networks from UAI competition.

Dataset	$\frac{t}{1+t}$	$\sqrt{t}$	$\min\{1, t\}$	$\max\{1, t\}$	LSB 1	LSB 2
MN 1	<b>2.90 ± 0.76</b>	<b>3.41 ± 0.77</b>	2.54 ± 0.32	<b>2.70 ± 0.63</b>	<b>3.19 ± 0.46</b>	<b>3.22 ± 0.38</b>
MN 2	<b>3.43 ± 0.75</b>	<b>3.92 ± 0.94</b>	<b>3.78 ± 0.50</b>	<b>3.63 ± 0.67</b>	<b>3.52 ± 0.42</b>	<b>3.44 ± 0.44</b>

as its extension to the continuous case.

Lastly, further theoretical investigation is required to establish the local convergence of our training algorithm. Indeed, standard results of gradient-based optimization do not apply in this setting, as the iterates are correlated. This remains an open challenge for future work.

### Acknowledgements

This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. The author would like to thank Luc de Raedt for the financial support.

### References

Afshar, H. M., Domke, J., et al. Reflection, Refraction, and Hamiltonian Monte Carlo. In *NeurIPS*, pp. 3007–3015, 2015.

Alexeev, N., Isomurodov, J., Sukhov, V., Korotkevich, G., and Sergushichev, A. Markov Chain Monte Carlo for Active Module Identification Problem. *BMC Bioinformatics*, 21(6):1–20, 2020.

Altekar, G., Dwarkadas, S., Huelsenbeck, J. P., and Ronquist, F. Parallel Metropolis Coupled Markov Chain Monte Carlo for Bayesian Phylogenetic Inference. *Bioinformatics*, 20(3):407–415, 2004.

Andrieu, C. and Thoms, J. A Tutorial on Adaptive MCMC. *Statistics and Computing*, 18(4):343–373, 2008.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. An Introduction to MCMC for Machine Learning. *Machine learning*, 50(1):5–43, 2003.

Dai, H., Singh, R., Dai, B., Sutton, C., and Schuurmans, D. Learning Discrete Energy-based Models via Auxiliary-variable Local Exploration. In *NeurIPS*, pp. 10443–10455, 2020.

Dinh, V., Bilge, A., Zhang, C., and Matsen I. V., F. A. Probabilistic Path Hamiltonian Monte Carlo. In *ICML*, pp. 1009–1018, 2017.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

Gong, W., Li, Y., and Hernández-Lobato, J. M. Meta-Learning for Stochastic Gradient MCMC. In *ICLR*, 2019.

Grathwohl, W., Swersky, K., Hashemi, M., Duvenaud, D., and Maddison, C. J. Oops I Took A Gradient: Scalable Sampling for Discrete Distributions. In *ICML*, 2021.

Grenander, U. and Miller, M. I. Representations of Knowledge in Complex Systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(4):549–581, 1994.

Habib, R. and Barber, D. Auxiliary Variational MCMC. In *ICLR*, 2018.

Hans, C., Dobra, A., and West, M. Shotgun Stochastic Search for “Large P” Regression. *Journal of the American Statistical Association*, 102(478):507–516, 2007.

Higdon, D. M. Auxiliary Variable Methods for Markov Chain Monte Carlo with Applications. *Journal of the American statistical Association*, 93(442):585–595, 1998.

Hoffman, M. D. and Gelman, A. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Hornik, K., Stinchcombe, M., and White, H. Multilayer Feedforward Networks are Universal Approximators. *Neural networks*, 1989.

Jaini, P., Nielsen, D., and Welling, M. Sampling in Combinatorial Spaces with SurVAE Flow Augmented MCMC. In *AISTATS*, pp. 3349–3357, 2021.

Koller, D. and Friedman, N. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

Levy, D., H., M. D., and Sohl-Dickstein, J. Generalizing Hamiltonian Monte Carlo with Neural Networks. In *ICLR*, 2018.

MacKay, D. J. C. *Information Theory, Inference and Learning Algorithms*. Cambridge university press, 2003.

Neal, R. *Probabilistic Inference Using Markov Chain Monte Carlo Methods*. Department of Computer Science, University of Toronto Toronto, Ontario, Canada, 1993.

Nijkamp, E., Hill, M., Han, T., Zhu, S. C., and Wu, Y. N. On the Anatomy of MCMC-Based Maximum Likelihood Learning of Energy-Based Models. In *AAAI*, pp. 5272–5280, 2020.

- Nishimura, A., Dunson, D. B., and Lu, J. Discontinuous Hamiltonian Monte Carlo for Discrete Parameters and Discontinuous Likelihoods. *Biometrika*, 107(2):365–380, 2020.
- Pakman, A. and Paninski, L. Auxiliary-Variable Exact Hamiltonian Monte Carlo Samplers for Binary Distributions. In *NeurIPS*, 2013.
- Pasarica, C. and Gelman, A. Adaptively Scaling the Metropolis Algorithm Using Expected Squared Jumped Distance. *Statistica Sinica*, pp. 343–364, 2010.
- Pompe, E. and Holmes, C. and Łatuszyński, K. and others. A Framework for Adaptive MCMC Targeting Multimodal Distributions. *Annals of Statistics*, 48(5):2930–2952, 2020.
- Robert, C. and Casella, G. *Monte Carlo Statistical Methods*. Springer Science & Business Media, 2013.
- Song, J., Zhao, S., and Ermon, S. A-NICE-MC: Adversarial Training for MCMC. In *NeurIPS*, 2017.
- Sun, H., Dai, H., Xia, W., and Ramamurthy, A. Path Auxiliary Proposal for MCMC in Discrete Space. In *ICLR*, 2022.
- Titsias, M. K. and Yau, C. The Hamming Ball Sampler. *Journal of the American Statistical Association*, 112(520): 1598–1611, 2017.
- Wang, T., Wu, Y., Moore, D. A., and Russell, S. J. Meta-Learning MCMC Proposals. In *NeurIPS*, pp. 4150–4160, 2018.
- Zanella, G. Informed Proposals for Local MCMC in Discrete Spaces. *Journal of the American Statistical Association*, 115(530):852–865, 2020.
- Zhang, Y., Ghahramani, Z., Storkey, A. J., and Sutton, C. Continuous Relaxations for Discrete Hamiltonian Monte Carlo. *NeurIPS*, 25:3194–3202, 2012.
- Zhu, M. Sample Adaptive MCMC. In *NeurIPS*, volume 32, 2019.

## A. Detailed Balance

We want to prove that  $p(\mathbf{x})T(\mathbf{x}'|\mathbf{x}) = p(\mathbf{x}')T(\mathbf{x}|\mathbf{x}')$  for all  $\mathbf{x}' \neq \mathbf{x}$ . We have that

$$\begin{aligned} p(\mathbf{x})A(\mathbf{x}', \mathbf{x}) \frac{g\left(\frac{\tilde{p}(\mathbf{x}')}{\tilde{p}(\mathbf{x})}\right)1[\mathbf{x}' \in N(\mathbf{x})]}{Z(\mathbf{x})} &= \\ p(\mathbf{x}')A(\mathbf{x}, \mathbf{x}') \frac{g\left(\frac{\tilde{p}(\mathbf{x})}{\tilde{p}(\mathbf{x}')}\right)1[\mathbf{x} \in N(\mathbf{x}')] }{Z(\mathbf{x}')} & \end{aligned}$$

By observing that  $1(\mathbf{x}' \in N(\mathbf{x})) = 1(\mathbf{x} \in N(\mathbf{x}'))$  and using the balancing property, we can simplify previous equality to obtain the following relation:

$$\begin{aligned} p(\mathbf{x})A(\mathbf{x}', \mathbf{x}) \frac{g\left(\frac{\tilde{p}(\mathbf{x}')}{\tilde{p}(\mathbf{x})}\right)}{Z(\mathbf{x})} &= \\ p(\mathbf{x}')A(\mathbf{x}, \mathbf{x}') \frac{\tilde{p}(\mathbf{x})}{\tilde{p}(\mathbf{x}')} \frac{g\left(\frac{\tilde{p}(\mathbf{x}')}{\tilde{p}(\mathbf{x})}\right)}{Z(\mathbf{x}')} & \end{aligned}$$

Therefore, we can apply standard algebra to simplify even more

$$\tilde{p}(\mathbf{x})A(\mathbf{x}', \mathbf{x}) = \tilde{p}(\mathbf{x}')A(\mathbf{x}, \mathbf{x}') \frac{Z(\mathbf{x})}{Z(\mathbf{x}')}$$

Finally, recall that for balancing functions  $A(\mathbf{x}, \mathbf{x}') = \min \left\{ 1, \frac{\tilde{p}(\mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')Q(\mathbf{x}|\mathbf{x}')} \right\} = \min \left\{ 1, \frac{Z(\mathbf{x}')}{Z(\mathbf{x})} \right\}$  and therefore previous equality becomes an identity, namely:

$$\tilde{p}(\mathbf{x})A(\mathbf{x}', \mathbf{x}) = \tilde{p}(\mathbf{x}')A(\mathbf{x}', \mathbf{x})$$

thus proving detailed balance.

## B. Ergodicity

Let's consider a Markov chain, namely

$$T(\mathbf{x}'|\mathbf{x}) = A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x}) + 1[\mathbf{x}' = \mathbf{x}] \sum_{\mathbf{x}'' \in \mathcal{X}} (1 - A(\mathbf{x}'', \mathbf{x}))Q(\mathbf{x}''|\mathbf{x}) \quad (7)$$

with a proposal of the following form:

$$Q(\mathbf{x}'|\mathbf{x}) = \frac{g\left(\frac{\tilde{p}(\mathbf{x}')}{\tilde{p}(\mathbf{x})}\right)1[\mathbf{x}' \in N(\mathbf{x})]}{Z(\mathbf{x})} \quad (8)$$

We can prove the ergodicity of the Markov chain for the case where the fixed-point distribution  $p(\mathbf{x}) > 0$  for every  $\mathbf{x} \in \mathcal{X}$  and then extend it to a general distribution  $p$ .

Now, assume that  $p(\mathbf{x}) > 0$  for any point  $\mathbf{x} \in \mathcal{X}$ ,  $g(t) > 0$  for any  $t > 0$  and  $\mathcal{X}$  is a  $d$ -dimensional discrete space. Then, the Markov chain in Eq. 7 with proposal defined according to Eq. 8 can reach any state  $\mathbf{x}'$  from any state  $\mathbf{x}$  in  $d$  steps with non-zero probability. More formally, we can construct a new Markov chain by applying  $d$  times the original one and identify its transition probability with  $T^d(\mathbf{x}'|\mathbf{x})$ . We can easily check, thanks to our assumptions, that  $T^d(\mathbf{x}'|\mathbf{x}) > 0$  for any  $\mathbf{x}, \mathbf{x}'$ . In other words, the original Markov chain is regular. This is sufficient to satisfy the assumptions of the fundamental theorem of homogeneous Markov chains (Neal, 1993), thus proving ergodicity.

We can extend the previous result to any arbitrary  $p$  (namely considering cases where  $p(\mathbf{x}) = 0$  for some  $\mathbf{x} \in \mathcal{X}$ ). This can be achieved by modifying our assumptions on  $g$ , namely considering that  $g(t) > 0$  for any  $t \geq 0$  and reusing the same proof strategy.

## C. Proof of Theorem 1

We want to prove the following theorem.

**Theorem C.1.** Consider  $\mathcal{X} = \{0, 1\}^d$  and  $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/\Gamma$ , where  $\Gamma$  is a normalizing constant. Define two auxiliary distributions  $Q_1$  and  $Q_2$ , such that for all  $\mathbf{x} \in \mathcal{X}$ ,  $Q_1(\mathbf{x}) > 0$  whenever  $\tilde{p}(\mathbf{x}) > 0$ , and for all  $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ ,  $Q_2(\mathbf{x}') > 0$  whenever  $Q(\mathbf{x}'|\mathbf{x}) > 0$ . Therefore,

(a) if  $\mathcal{M}(\mathbf{x}) = 1 - \sum_{\mathbf{x}'' \in \mathcal{N}(\mathbf{x})} A(\mathbf{x}'', \mathbf{x})Q(\mathbf{x}''|\mathbf{x})$ , then

$$\mathcal{L}_\theta = E_{\substack{\mathbf{x} \sim Q_1 \\ \mathbf{x}' \sim Q_2}} \left\{ \frac{\tilde{p}(\mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{Q_1(\mathbf{x})Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} \right\} + E_{\mathbf{x} \sim Q_1} \left\{ \frac{\tilde{p}(\mathbf{x})}{Q_1(\mathbf{x})} \mathcal{M}(\mathbf{x}) \log \frac{\mathcal{M}(\mathbf{x})}{\tilde{p}(\mathbf{x})} \right\} \quad (9)$$

$$\text{and } \mathcal{I}_\theta = \frac{\mathcal{L}_\theta}{\Gamma} + \frac{\log \Gamma}{\Gamma}.$$

(b) if  $\mathcal{M}(\mathbf{x}) = 1 - A(\mathbf{x}^*, \mathbf{x})Q(\mathbf{x}^*|\mathbf{x})$ , where  $\mathbf{x}^*$  is randomly sampled according to a uniform distribution over  $\mathcal{N}(\mathbf{x})$ , then

$$\mathcal{L}_\theta = E_{\substack{\mathbf{x} \sim Q_1 \\ \mathbf{x}' \sim Q_2}} \left\{ \frac{\tilde{p}(\mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{Q_1(\mathbf{x})Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} \right\} + E_{\mathbf{x} \sim Q_1} \left\{ \frac{\mathcal{M}(\mathbf{x})}{Q_1(\mathbf{x})} \left[ \eta \mathcal{M}(\mathbf{x}) - \tilde{p}(\mathbf{x}) \log \eta - \tilde{p}(\mathbf{x}) \right] \right\} \quad (10)$$

$$\text{and } \mathcal{I}_\theta \leq \frac{\mathcal{L}_\theta}{\Gamma} + \frac{\log \Gamma}{\Gamma}, \text{ for } \eta \in \mathbb{R}^+.$$

*Proof.* Let's start from the definition of  $\mathcal{I}_\theta$  and use the fact that  $T_\theta(\mathbf{x}'|\mathbf{x}) = A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x}) + 1[\mathbf{x}' = \mathbf{x}] \sum_{\mathbf{x}'' \in \mathcal{X}} (1 - A(\mathbf{x}'', \mathbf{x}))Q(\mathbf{x}''|\mathbf{x})$ .

$$\begin{aligned} \mathcal{I}_\theta &= KL\{p(\mathbf{x})T_\theta(\mathbf{x}'|\mathbf{x})\|p(\mathbf{x})p(\mathbf{x}')\} \\ &= \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{X}} p(\mathbf{x})T_\theta(\mathbf{x}'|\mathbf{x}) \log \frac{p(\mathbf{x})T_\theta(\mathbf{x}'|\mathbf{x})}{p(\mathbf{x})p(\mathbf{x}')} \\ &= \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{X}} p(\mathbf{x})T_\theta(\mathbf{x}'|\mathbf{x}) \log \frac{T_\theta(\mathbf{x}'|\mathbf{x})}{p(\mathbf{x}')} \\ &= \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x}) \cup \{\mathbf{x}\}} p(\mathbf{x})T_\theta(\mathbf{x}'|\mathbf{x}) \log \frac{T_\theta(\mathbf{x}'|\mathbf{x})}{p(\mathbf{x}')} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x}) \cup \{\mathbf{x}\}} \tilde{p}(\mathbf{x})T_\theta(\mathbf{x}'|\mathbf{x}) \log \frac{T_\theta(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \tilde{p}(\mathbf{x})T_\theta(\mathbf{x}'|\mathbf{x}) \log \frac{T_\theta(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \tilde{p}(\mathbf{x})T_\theta(\mathbf{x}|\mathbf{x}) \log \frac{T_\theta(\mathbf{x}|\mathbf{x})}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \tilde{p}(\mathbf{x})A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \tilde{p}(\mathbf{x})T_\theta(\mathbf{x}|\mathbf{x}) \log \frac{T_\theta(\mathbf{x}|\mathbf{x})}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \tilde{p}(\mathbf{x})A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \tilde{p}(\mathbf{x}) \sum_{\mathbf{x}'' \in \mathcal{X}} [1 - A(\mathbf{x}'', \mathbf{x})]Q(\mathbf{x}''|\mathbf{x}) \log \frac{\sum_{\mathbf{x}'' \in \mathcal{X}} [1 - A(\mathbf{x}'', \mathbf{x})]Q(\mathbf{x}''|\mathbf{x})}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \tilde{p}(\mathbf{x})A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \tilde{p}(\mathbf{x}) \left[ 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x})Q(\mathbf{x}''|\mathbf{x}) \right] \log \frac{[1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x})Q(\mathbf{x}''|\mathbf{x})]}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} Q_1(\mathbf{x})Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{Q_1(\mathbf{x})Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x})Q(\mathbf{x}'|\mathbf{x})}{\tilde{p}(\mathbf{x}')} + \end{aligned}$$



$$+ \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{\tilde{p}(\mathbf{x})}{Q_1(\mathbf{x})} \left[ 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x}) \right] \log \frac{[1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})]}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \quad (11)$$

Now, by defining  $\mathcal{M}(\mathbf{x}) = 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})$ , we obtain case (a). Indeed, we have that

$$\begin{aligned} \mathcal{I}_\theta &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in N(\mathbf{x})} Q_1(\mathbf{x}) Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{Q_1(\mathbf{x}) Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{\tilde{p}(\mathbf{x})}{Q_1(\mathbf{x})} \left[ 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x}) \right] \log \frac{[1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})]}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in N(\mathbf{x})} Q_1(\mathbf{x}) Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{Q_1(\mathbf{x}) Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{\tilde{p}(\mathbf{x})}{Q_1(\mathbf{x})} \mathcal{M}(\mathbf{x}) \log \frac{\mathcal{M}(\mathbf{x})}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{\mathcal{L}_\theta}{\Gamma} + \frac{\log \Gamma}{\Gamma} \end{aligned} \quad (12)$$

For case (b), we can exploit the relation  $\log y \leq \eta y - \log \eta - 1$  for all  $y > 0$  we have that

$$\begin{aligned} \mathcal{I}_\theta &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in N(\mathbf{x})} Q_1(\mathbf{x}) Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{Q_1(\mathbf{x}) Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{\tilde{p}(\mathbf{x})}{Q_1(\mathbf{x})} \left[ 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x}) \right] \log \frac{[1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})]}{\tilde{p}(\mathbf{x})} + \frac{\log \Gamma}{\Gamma} \\ &\leq \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in N(\mathbf{x})} Q_1(\mathbf{x}) Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{Q_1(\mathbf{x}) Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{\tilde{p}(\mathbf{x})}{Q_1(\mathbf{x})} \left[ 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x}) \right] \left\{ \eta \frac{[1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})]}{\tilde{p}(\mathbf{x})} - \log \eta - 1 \right\} + \frac{\log \Gamma}{\Gamma} \\ &= \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in N(\mathbf{x})} Q_1(\mathbf{x}) Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{Q_1(\mathbf{x}) Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{[1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})]}{Q_1(\mathbf{x})} \left\{ \eta \left[ 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x}) \right] - \tilde{p}(\mathbf{x}) \log \eta - \tilde{p}(\mathbf{x}) \right\} + \frac{\log \Gamma}{\Gamma} \end{aligned} \quad (13)$$

Finally, we observe that  $1 - A(\tilde{\mathbf{x}}, \mathbf{x}) Q(\tilde{\mathbf{x}} | \mathbf{x}) \geq 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})$  for all  $\tilde{\mathbf{x}} \in N(\mathbf{x})$  and for all  $\mathbf{x} \in \mathcal{X}$ . Therefore,

$$\begin{aligned} \mathcal{I}_\theta &\leq \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in N(\mathbf{x})} Q_1(\mathbf{x}) Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{Q_1(\mathbf{x}) Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{[1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x})]}{Q_1(\mathbf{x})} \left\{ \eta \left[ 1 - \sum_{\mathbf{x}'' \in \mathcal{X}} A(\mathbf{x}'', \mathbf{x}) Q(\mathbf{x}'' | \mathbf{x}) \right] - \tilde{p}(\mathbf{x}) \log \eta - \tilde{p}(\mathbf{x}) \right\} + \frac{\log \Gamma}{\Gamma} \\ &\leq \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} \sum_{\mathbf{x}' \in N(\mathbf{x})} Q_1(\mathbf{x}) Q_2(\mathbf{x}') \frac{\tilde{p}(\mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{Q_1(\mathbf{x}) Q_2(\mathbf{x}')} A(\mathbf{x}', \mathbf{x}) \log \frac{A(\mathbf{x}', \mathbf{x}) Q(\mathbf{x}' | \mathbf{x})}{\tilde{p}(\mathbf{x}')} + \\ &\quad + \frac{1}{\Gamma} \sum_{\mathbf{x} \in \mathcal{X}} Q_1(\mathbf{x}) \frac{[1 - A(\mathbf{x}^*, \mathbf{x}) Q(\mathbf{x}^* | \mathbf{x})]}{Q_1(\mathbf{x})} \left\{ \eta \left[ 1 - A(\mathbf{x}^*, \mathbf{x}) Q(\mathbf{x}^* | \mathbf{x}) \right] - \tilde{p}(\mathbf{x}) \log \eta - \tilde{p}(\mathbf{x}) \right\} + \frac{\log \Gamma}{\Gamma} \end{aligned} \quad (14)$$

and by defining  $\mathcal{M}(\mathbf{x}) = 1 - A(\mathbf{x}^*, \mathbf{x}) Q(\mathbf{x}^* | \mathbf{x})$ , we obtain case (b).  $\square$

Table 4. Summary of the properties of different approaches.

Method	Target likelihood evaluations per sampling step	Number of variables modified per sampling step
Gibbs 2	4	2
Gibbs 4	16	4
Gibbs 10	1024	10
HB-10-1	20	1
HB-10-2	180	2
HB-10-3	960	3
FLSB 2	<b>1</b>	<b>1</b>

## D. Hyperparameters Used in the Experiments

- Learning rate  $\eta = 1e - 2$  for SGD optimizer with momentum.
- Burn-in iterations  $K = 2000$  (for Ising),  $K = 24000$  (for RBM),  $K = 500$  (for UAI).
- Iterations for sampling 30000 (for Ising), 120000 (for RBM), 10000 (for UAI).
- Batch size  $N = 30$  (for Ising),  $N = 16$  (for RBM),  $N = 5$  (for UAI).
- MLP network with one hidden layer of 10 neurons (for ISING and RBM) and monotonic network with 20 blocks of 20 neurons (for UAI). Refer to Appendix G for additional experiments on UAI with MLP.

For experiments on RBM, we reused the code provided by (Grathwohl et al., 2021) .

## E. Further Results for Ising

See Figure 6.

## F. Additional Experiments for RBM

We provide additional comparisons of FLSB 2 against sampling strategies modifying more than 1 variable per step. In particular, we compare FLSB 2 against Gibbs 2, Gibbs 4 and Gibbs 10 and also compare FLSB 2 against HB-10-1, HB-10-2 and HB-10-3. Results are shown in Figures 8 and 9, respectively. It is important to mention also that FLSB 2 uses a much smaller number of target likelihood evaluations compared to these previous approaches, as summarized in Table 4.

## G. Additional Experiments on UAI

We repeated the experiments on UAI using a MLP network with one hidden layer of 10 neurons, shown in Figure 10. We found that in this domain characterized by large close-to-zero mass regions, the initial random input  $\mathbf{x}$  is likely to fall outside the distribution support. In such case, there is no need to have balancing functions with  $g(0) > 0$ . Notably, this doesn't occur when using a monotonic network, as inspected by the experiments in Figure 11, where we train the monotonic network to match a function like  $\max\{1, t\}$ .

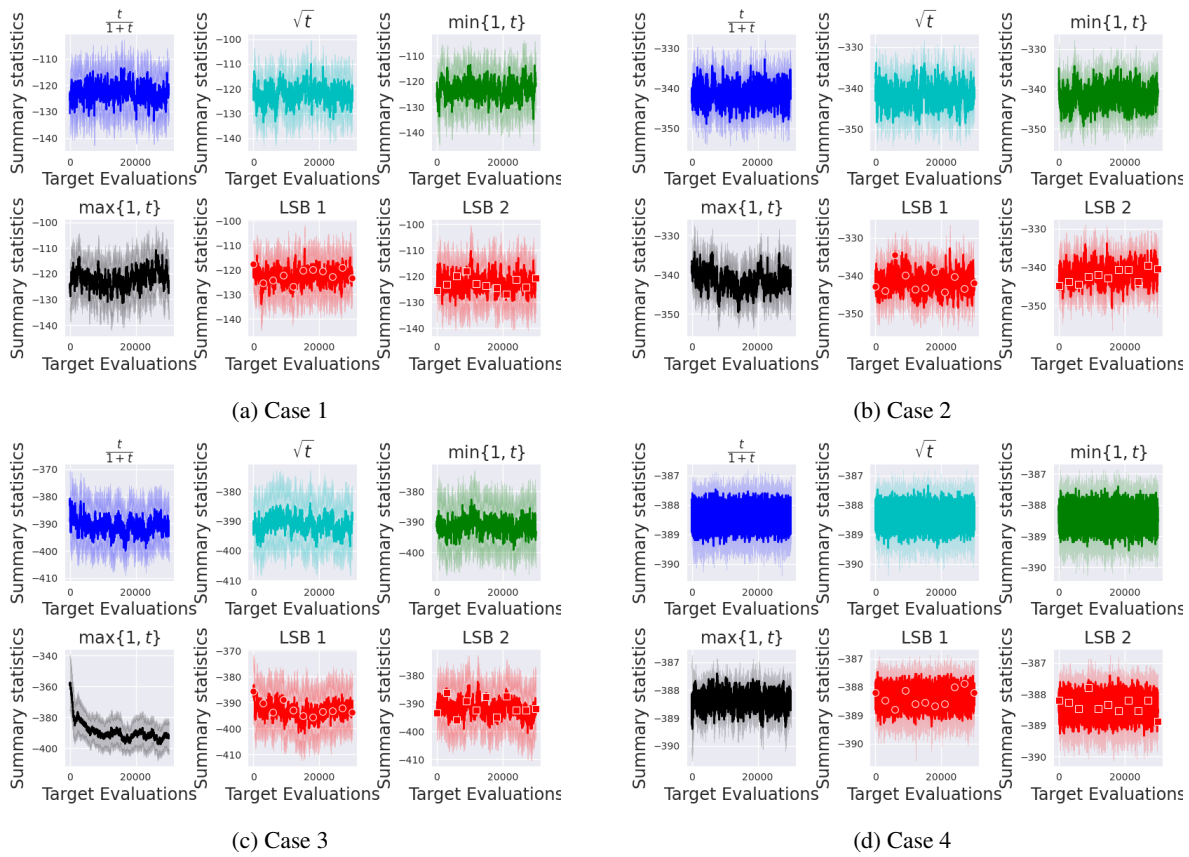


Figure 6. Traceplots on four cases of the Ising model ( $30 \times 30$ ) for the mixing phase. (a) Case 1: Independent-noisy, (b) case 2: Independent-clean, (c) case 3: Dependent-noisy, (d) case 4: Dependent-clean

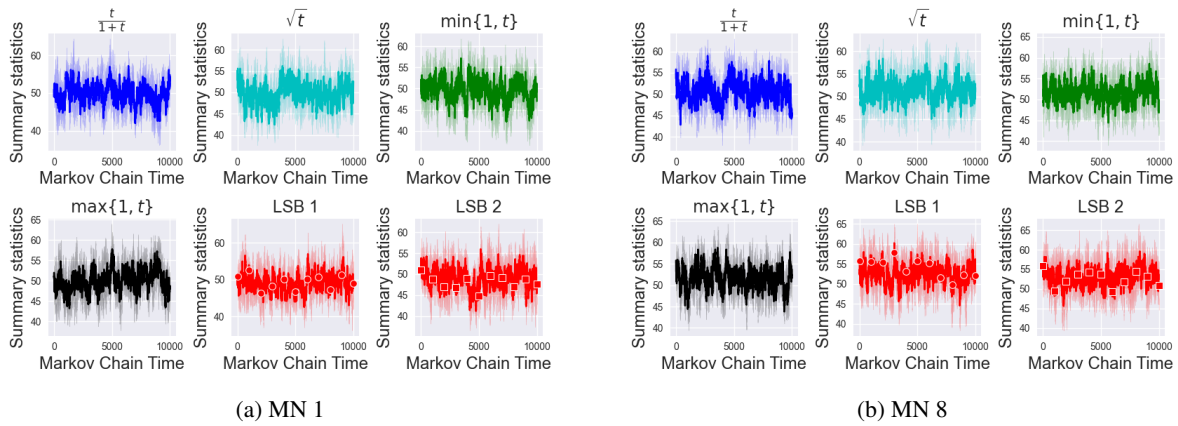


Figure 7. Traceplots on UAI benchmarks model (100 vars near-deterministic dependencies) for the mixing phase.

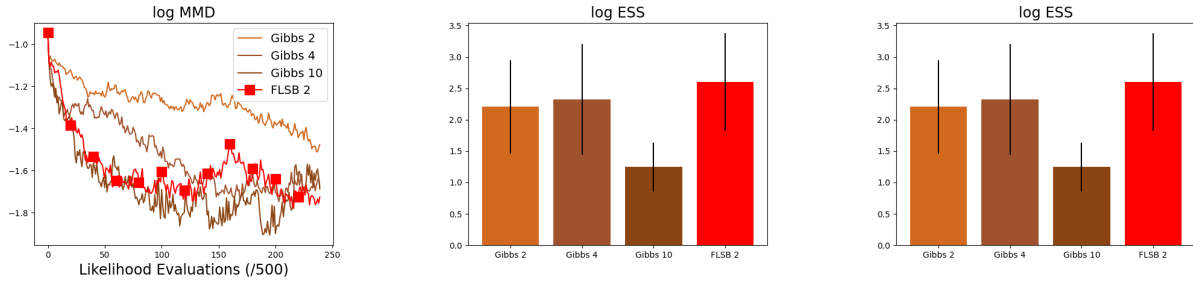


Figure 8. Comparison of FLSB 2 against block Gibbs sampling with block size of 2, 4 and 10 variables on RBMs.

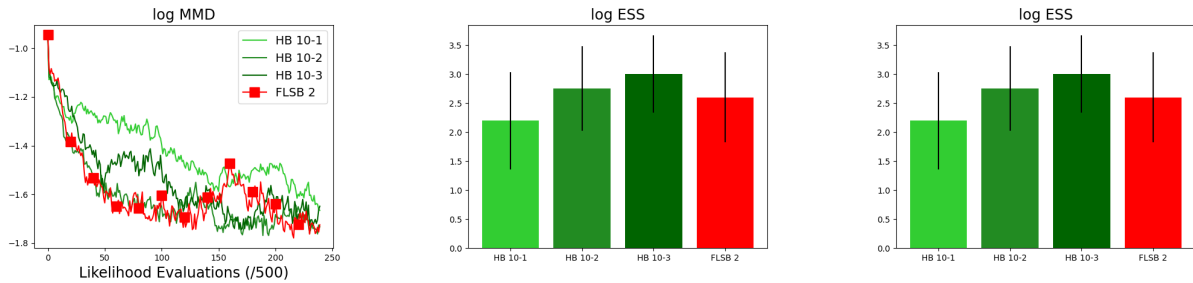


Figure 9. Comparison of FLSB 2 against the Hamming Ball sampler using 10 variables per block and updating 1, 2 and 3 variables per step.

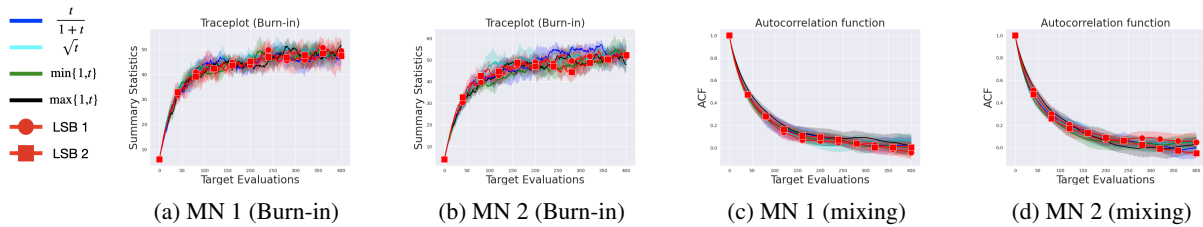


Figure 10. Samplers' performance on Markov networks from UAI competition (100 variables) with MLP network. (a)-(b) are the traceplots for the burn-in phase, while (c)-(d) are the autocorrelation function for the mixing one

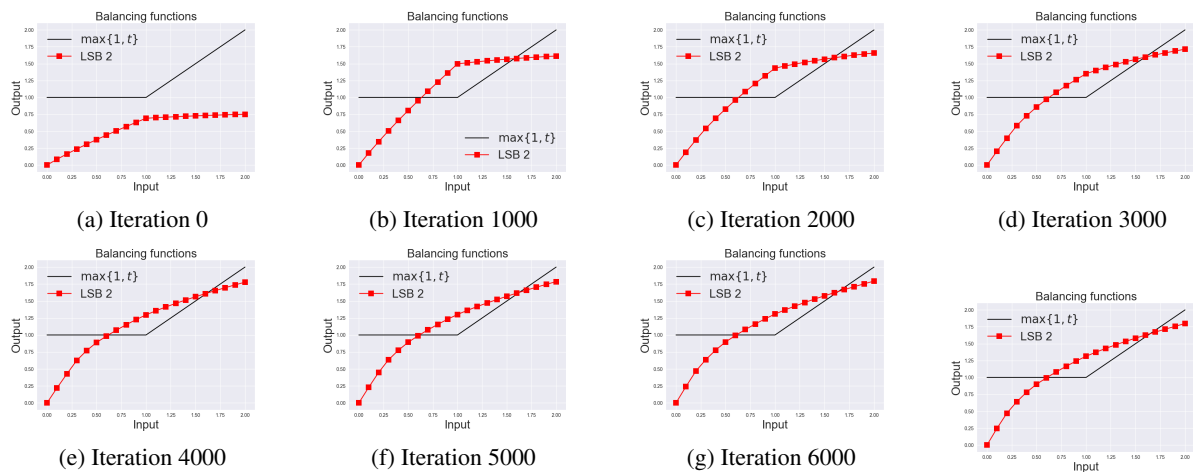


Figure 11. Training the monotonic network to match  $\max\{1, t\}$  balancing function.