Sampling via Rejection-Free Partial Neighbor Search

Sigeng Chen[1], Jeffrey S. Rosenthal[1], Aki Dote[2], Hirotaka Tamura[3] and Ali Sheikholeslami[4]

[1]Department of Statistical Sciences, University of Toronto, 700 University Avenue, Toronto, M5G 1Z5, Ontario, Canada.
[2]Fujitsu Ltd., 4-1-1 Kamikodanaka Nakahara-ku, Kawasaki, 211-8588, Kanagawa, Japan.
[3]DXR Laboratory Inc., 4-38-10 Takata-Nishi, Kohoku-ku, Yokohama, 223-0066, Kanagawa, Japan.
[4]Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, M5S 3G4, Ontario, Canada.

Contributing authors: sigeng.chen@mail.utoronto.ca; jeff@math.toronto.edu; dote.aki@fujitsu.com; tamura.hirotaka@dxrlab.com; ali@ece.utoronto.ca;

Abstract

The Metropolis algorithm (Metropolis et al, 1953; Hastings, 1970) involves producing a Markov chain to converge to a specified target density $\boldsymbol{\pi}$. In order to improve its efficiency, we can use the Rejection-Free (Rosenthal et al, 2021) version of the Metropolis algorithm, which avoids the inefficiency of rejections by evaluating all neighbors. Rejection-Free can be made more efficient through the use of parallelism hardware. However, for some specialized hardware, such as Digital Annealing Unit (Matsubara et al, 2020), the number of units will limit the number of neighbors being considered at each step. Hence, we propose an enhanced version of Rejection-Free known as Partial Neighbor Search, which only considers a portion of the neighbors while using the Rejection-Free technique. This method will be tested on several examples to demonstrate its effectiveness and advantages under different circumstances.

Keywords: Metropolis Algorithm, Rejection-Free, Partial Neighbor Search, Unbiased PNS, QUBO

1 Introduction

The Monte Carlo method involves the deliberate use of random numbers in a calculation with the structure of a stochastic process (Kalos and Whitlock, 2009). Monte Carlo techniques are based on repeating experiments sufficiently many times to obtain many quantities of interest using the Law of Large Numbers and other statistical inference methods (Kroese et al, 2014). The three main applications of Monte Carlo methods are optimization, numerical integration, and sampling (Kroese et al, 2014). This paper focuses on the Markov chain Monte Carlo method for sampling.

The Markov chain Monte Carlo method (MCMC) simulates observations from a target distribution to obtain a chain of states that eventually converges to the target distribution itself. The Metropolis algorithm (Metropolis et al, 1953; Hastings, 1970), an MCMC method, is one of the most popular techniques among its kind (Hitchcock, 2003). The Metropolis algorithm produces a Markov chain $\{X_0, X_1, X_2, \dots\}$ on the state space $\mathcal{S}$ and target density function π, as follows: given the current state x_k, the Metropolis algorithm first proposes a new state y from a symmetric proposal distribution $\mathcal{Q}(x_k, \cdot)$; it then accepts the new state (i.e., sets $x_k = y$) with probability $\min(1, \frac{\pi(y)}{\pi(x_k)})$; otherwise, it rejects the proposal (i.e., sets $x_{x+1} = x_k$). This simple algorithm ensures that the Markov chain has π as a stationary distribution.

However, the Metropolis algorithm may suffer from the inefficiency of rejections. We have a probability of $\left[1 - \min(1, \frac{\pi(y)}{\pi(x_k)})\right]$ to remain at the current state, even though we have spent time proposing a state, computing a ratio of target probabilities, generating a random variable, and deciding not to accept the proposal. Therefore, we proposed the Rejection-Free algorithm in Rosenthal et al (2021) to improve the Metropolis algorithm's performance. Furthermore, the parallelism of computer hardware can significantly increase the efficiency of Rejection-Free. The use of parallelism in Rejection-Free combined with simple techniques such as parallel tempering can yield 100x to 10,000x speedups (Sheikholeslami, 2021). However, there is a limit to the number of parallel tasks that can be executed simultaneously on most specialized parallelism hardware. Accordingly, Rejection-Free has a ceiling on the number of neighbors that can be evaluated at each step. Consequently, we present an enhanced version of Rejection-Free called Partial Neighbor Search (PNS), which only considers part of the neighbors when applying the Rejection-Free technique, whereas the Rejection-Free technique means considering all selected neighbors and calculating the next state when ignoring any immediately repeated states. PNS has also been developed to solve optimization problems, and it outperforms the Simulated Annealing and Rejection-Free algorithms in many optimization problems, including the QUBO, Knapsack, and 3R3XOR problems; see Chen et al (2022) for further information.

We next review the Metropolis-Hastings algorithm and Rejection-Free algorithm in more detail. Then, in Section 2, we introduce our Basic Partial Neighbor Search (Basic PNS) sampling algorithm, which considers subsets of

neighbor states for possible moves and calculates the multiplicity list directly from the subsets. Unfortunately, this version of the Markov chain does not converge to the target density. In Section 3, we introduce our unbiased version of Partial Neighbor Search (Unbiased PNS), where the sampling distribution will converge to the target density correctly; see Appendix A for the proof. Unlike Rejection-Free, Unbiased PNS can always use the advantage of the parallelism hardware to improve the sampling efficiency, no matter the dimension of the problem. We apply the Unbiased PNS to the QUBO question to illustrate its performance in Section 4. In addition, we discuss the choice of subsets of the Unbiased PNS for the QUBO question in Section 5. We further illustrate that we can apply the Unbiased PNS to continuous models in Section 6. We compare the Metropolis algorithm and Unbiased PNS in a continuous example called the Donuts example to demonstrate the performance of Unbiased PNS in Section 7. Furthermore, in our optimization paper (Chen et al, 2022), the performance of PNS in optimization questions is much better than Rejection-Free and Simulated Annealing. Thus we adapt the Optimization PNS and use it as the burn-in part for sampling in Section 8. Geyer (2011) stated that burn-in until converging to stationarity is not necessary for MCMC. If we take Geyer's (2011) argument, then we can use Optimization PNS to replace the burn-in. On the other hand, we can combine the Optimization PNS and the regular burn-in to get a better algorithm that will converge to stationarity faster. In Appendix A, we prove the convergence theorem of our Unbiased PNS algorithm. In addition, in Appendix B, we show how to sample proportionally and efficiently on parallelism hardware. Even when we apply the algorithm to a single core implementation, the technique also reduces the time of selecting the next state to some extent.

1.1 Background on the Metropolis-Hastings algorithm

Discrete sampling questions usually contain the following essential elements (adapted from the essential elements of Simulated Annealing in Bertsimas and Tsitsiklis (1993)):

1. a state space $\mathcal{S}$;
2. a real-valued target distribution $\pi : \mathcal{S} \to [0, 1]$ where $\sum_{x\in\mathcal{S}} \pi(x) = 1$;
3. $\forall x \in \mathcal{S}$, $\exists$ a proposal distribution $\mathcal{Q}(x, \cdot)$ where $\sum_{y\in\mathcal{S}\setminus\{x\}} \mathcal{Q}(x, y) = 1$, and $\mathcal{Q}(x, y) > 0 \iff \mathcal{Q}(y, x) > 0$, $\forall x, y \in \mathcal{S}$;
4. $\forall x \in \mathcal{S}$, $\exists$ a neighbor set $\mathcal{N}(x) = \{y \in \mathcal{S} \mid \mathcal{Q}(x, y) > 0\} \subset \mathcal{S}\setminus\{x\}$.

For simplicity, we focus on the discrete cases here. We will talk more about the general state space in Appendix A.

The Metropolis algorithm has been the most successful and influential of all the members of the Monte Carlo method (Beichl and Sullivan, 2000). It is designed to generate a Markov chain that converges to a given target distribution π on a state space $\mathcal{S}$. The Metropolis-Hastings(M-H) algorithm is a generalized version of the Metropolis algorithm, including the possibility of a

non-symmetric proposal distribution $\mathcal{Q}$ (Hitchcock, 2003). The M-H algorithm is stated in Algorithm 1.

Algorithm 1 the Metropolis-Hastings algorithm

initialize X_0
for k in 1 to K **do**
  random $Y \in \mathcal{N}(X_{k-1})$ based on $\mathcal{Q}(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \frac{\pi(Y)\mathcal{Q}(Y,X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1},Y)}$ **then**
    ▷ accept with probability $\min\left\{1, \frac{\pi(Y)\mathcal{Q}(Y,X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1},Y)}\right\}$
    $X_k \leftarrow Y$ ▷ accept and move to state Y
  else
    $X_k \leftarrow X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
end for

Algorithm 1 ensures the Markov chain $\{X_0, X_1, X_2, \ldots, X_K\}$ has π as stationary distribution. It follows (assuming irreducibility) that the expected value $E_\pi(h)$ of a functional $h : S \to \mathbb{R}$ with respect to π can be estimated by $\frac{1}{K}\sum_{i=1}^{K} h(X_i)$ for sufficiently large run length K.

In Algorithm 1, if $U_k \geq \frac{\pi(Y)\mathcal{Q}(Y,X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1},Y)}$, then we will remain at the current state, even though we have spent time in proposing a state, computing a ratio of target probabilities, generating a random variable U_k, and deciding not to accept the proposal. Such inefficiencies could happen frequently and are considered a necessary evil of the M-H algorithm. Thus, we proposed the Rejection-Free algorithm (Rosenthal et al, 2021) to improve the inefficiency caused by these rejections.

1.2 Background on Rejection-Free algorithm for sampling

Before introducing the Rejection-Free algorithm, we must first introduce the jump chain. Given a run $\{X_k\}$ of a Markov chain, we define the jump chain to be $\{J_k, M_k\}$, where $\{J_k\}$ represents the same chain as $\{X_k\}$ except omitting any immediately repeated states, and we use the multiplicity list $\{M_k\}$ to count the number of times the original chain remains at the same state.

For example, if the original chain is

$$\{X_k\} = \{a, b, b, b, a, a, c, c, c, c, d, d, a, \ldots\},$$

then the jump chain and the corresponding multiplicity list would be

$$\{J_k\} = \{a, b, a, c, d, a, \ldots\},\ \{M_k\} = \{1, 3, 2, 4, 2, 1, \ldots\}.$$

The jump chain itself is also a Markov chain. If we assume the transition probability of the original Markov chain $\{X_k\}$ generated by Algorithm 1 is

$$P[X_k = y \mid X_{k-1} = x] = \mathcal{Q}(x, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, x)}{\pi(x)\mathcal{Q}(x, y)}\right\}, \tag{1}$$

Then the transition probabilities $\hat{P}(y \mid x)$ for the jump chain $\{J_k, M_k\}$ is specified by

$$\hat{P}(J_k = x \mid J_{k-1} = x) = 0, \forall x \in \mathcal{S};$$

$$\begin{aligned} \hat{P}(J_k = y \mid J_{k-1} = x) &= P(X_k = y \mid X_{k-1} = x, X_{k-1} \neq x) \\ &= \frac{P(X_k = y \mid X_{k-1} = x)}{\sum_{z \neq x} P(X_k = z \mid X_{k-1} = x)}, \forall y \neq x. \end{aligned} \tag{2}$$

Moreover, the conditional distribution of $\{M_k\}$ given $\{J_k\}$ is equal to the distribution of $1 + G$ where G is a geometric random variable with success probability $1 - P(x \mid x) = \sum_{z \neq x} P(z \mid x)$; see Rosenthal et al (2021) for more details.

In addition, for the jump chain $\{J_k, M_k\}_{k=1}^K$, we call the total number of different states K to be the jump sample size, and we call $\sum_{k=1}^K M_k$ to be the original sample size, which is the corresponding length of the original Markov chain.

Given the above properties of the jump chain, the Rejection-Free algorithm is a sampling method that produces the jump chain as described by Algorithm 2. Note that the Rejection-Free algorithm described here can only deal with the discrete cases with at most a finite number of neighbors for all states. We'll review the Rejection-Free for general state space in Section 6.

Note that, in Algorithm 2, when we need to pick our next state according to the given probabilities, we can use the technique shown in Appendix B, which is specially designed for parallelism hardware. In addition, even when the Rejection-Free is applied to a single core implementation, such a technique is still faster than other methods to sample proportionally.

Algorithm 2 ensures (assuming irreducibility) that the expected value $E_\pi(h)$ of a functional $h : \mathcal{S} \to \mathbb{R}$ with respect to π can be estimated by $\frac{\sum_{k=1}^K M_k\, h(J_k)}{\sum_{k=1}^K M_k}$ for sufficiently large run length K, while avoiding any rejections. Rejection-Free can lead to great speedup in examples where rejections frequently happen for the M-H algorithm (Rosenthal et al, 2021).

2 Basic Partial Neighbor Search algorithm

In Algorithm 2, we can do this algorithm with parallelism in computer hardware to produce more efficient samples. However, the number of tasks that

Algorithm 2 Rejection-Free algorithm for discrete case

initialize J_0

for k in 1 to K **do**

choose the next jump chain State $J_k \in \mathcal{N}(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, y)}\right\}$$

calculate multiplicity list $M_{k-1} \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}(J_{k-1})} \mathcal{Q}(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, z)}\right\}$$

end for

can be computed simultaneously by the parallelism hardware is not unlimited, while the number of neighbors $|\mathcal{N}(x)|$ can be super large. How can we take full advantage of the Rejection-Free with limited parallel hardware?

Assume the number of neighbors in Rejection-Free is at most N. That is, for $\forall x \in S$, $|\mathcal{N}(x)| \leq N$. In addition, assume the number of tasks that can be computed simultaneously by the parallelism hardware is M. If $M > N$, then we can compute the transition probability of the original chain simultaneously by the parallelism hardware, where the transition probability is

$$P(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, y)}\right\}. \tag{3}$$

Then the transition probability $\hat{P}$ (defined at Equation 2) for the Rejection-Free algorithm as stated in Algorithm 2 is propositional to P, $\forall y \neq J_{k-1}$. On the other hand, if $M \leq N$, the simplest way to take advantage of parallelism hardware is to evenly distribute the calculation tasks of the transition probabilities to each unit. In this case, each unit of parallelism hardware needs to calculate the probabilities for either $\lfloor \frac{N}{M} \rfloor$ (the floor function) or $\lceil \frac{N}{M} \rceil$ (the ceiling function) times, and then we can put the information from all these parts together for the next step of the algorithm. This method works for processors designed for general purposes, such as Intel and AMD cores. However, these chips are not specially designed for parallel computing, and off-chip communication significantly slows down the transfer rate of data to and from the cores (Sodan et al, 2010). Therefore, using Intel and AMD cores as parallelism hardware is applicable but not ideal.

Moreover, several parallelization hardware specialized for parallel MCMC trials has been proposed. For example, the second generation of Fujitsu Digital Annealer uses a dedicated processor called a Digital Annealing Unit (DAU) (Matsubara et al, 2020) to achieve high speed. This dedicated processor is

designed to minimize communication overhead in arithmetic circuitry and with memory. In addition, the dedicated processor provides a virtually Rejection-Free process, resulting in a throughput that is orders of magnitude faster than that of a general-purpose processor. The problem with this Fujitsu chip is that it is rigidly constrained by on-chip memory capacity relative to the problem size M that can be processed in parallel. For problem sizes $N > M$, it is impossible to compute transition probabilities for all neighborhoods to achieve Rejection-Free or similar parallel trials. The number of neighbors considered in each step must be limited to be within the on-chip memory capacity.

Initially, we want to adapt our Optimization Partial Neighbor Search (Optimization PNS) algorithm from Chen et al (2022) to the sampling question here. Intuitively, we can use the Optimization PNS and add a step for calculating the multiplicity list. The Basic Partial Neighbor Search algorithm (Basic PNS) is shown in Algorithm 3. Again, we focus on discrete cases with at most a finite number of neighbors here. We will talk about PNS for general state space in Section 6 and Appendix A.

Algorithm 3 Basic Partial Neighbor Search algorithm

initialize J_0

for k in 1 to K **do**

pick the Partial Neighbor Set $\mathcal{N}_k(J_{k-1}) \subset \mathcal{N}(J_{k-1})$

choose the next jump chain State $J_k \in \mathcal{N}_k(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, y)}\right\}$$

calculate multiplicity list $M_{k-1} \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}_k(J_{k-1})} \mathcal{Q}(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, z)}\right\}$$

end for

The only difference between the Basic PNS (Algorithm 3) and Rejection-Free (Algorithm 2) is that we only calculate the transition probability and all the corresponding values for a subset $\mathcal{N}_k$ of all the neighbors for each step within the loop. Here, $\mathcal{N}_k(J_{k-1})$ is a subset of $\mathcal{N}(J_{k-1})$ at our choice, and the subscript k in $\mathcal{N}_k$ represents the subset of neighbors for step k. For example, we can simply say that $\mathcal{N}_k(J_{k-1})$ is a random subset of $\mathcal{N}(J_{k-1})$ with half of the elements. In addition, $\mathcal{Q}_k(X, Y)$ is the corresponding proposal distribution satisfying $\mathcal{Q}_k(x, y) \propto \mathcal{Q}(x, y)$ for $Y \in \mathcal{N}_k(x)$ and $\mathcal{Q}_k(x, y) = 0$ otherwise. However, the Markov chain produced by Algorithm 3 is different from the true MCMC, and it might not converge to the true density π, as we now show.

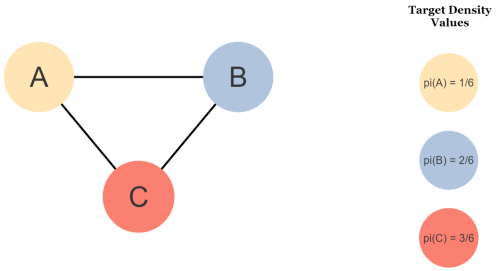

Fig. 1 Diagram of Example 1 showing non-convergence property of the Basic PNS.

2.1 Example 1 of the Nonconvergence problem by Basic PNS

The first example is shown in Figure 1, from which we have $\pi(A) \propto 1$, $\pi(B) \propto 2$, and $\pi(C) \propto 3$. We consider the Basic PNS algorithm with a uniform proposal distribution $\mathcal{Q}$. In addition, only half of the neighbors are chosen for $\mathcal{N}_k$ at each step. That is, we only need to consider one neighbor each time.

Then, if the MCMC is located at state A, then $\mathcal{N}(A) = \{B, C\}$. $\mathcal{N}_k(A) = \{B\}$ or $\{C\}$ each with 50% probability, and thus, the algorithm will force the chain to move to either B or C with 50% probability. Similarly, when the Markov chain is located at state B, the next state will be A or C with 50% probability, and when the Markov chain is located at state C, the next state will be A or B with 50% probability.

On the other hand, we can calculate the corresponding multiplicity lists M_A, M_B, and M_B at state A as follows:

1. $\hat{P}[B \mid A] \propto P[B \mid A] = \mathcal{Q}(A, B) \min\{1, \frac{\pi(B)\mathcal{Q}(B,A)}{\pi(A)\mathcal{Q}(A,B)}\} = 0.5$;
2. $\hat{P}[C \mid A] \propto P[C \mid A] = \mathcal{Q}(A, C) \min\{1, \frac{\pi(C)\mathcal{Q}(C,A)}{\pi(A)\mathcal{Q}(A,C)}\} = 0.5$;
3. the transition probabilities $\hat{P}$ from A to either B or C in Rejection-Free are both 50%;
4. $M_A = 1 + G$ where $G \sim \text{Geom}(P[B \mid A] + P[C \mid A]) = \text{Geom}(1)$
5. $\mathbb{E}(M_A) = 1$
6. Similarly, we have $\mathbb{E}(M_B) = \frac{5}{4}$, $\mathbb{E}(M_A) = \frac{9}{4}$

Thus, for the Basic PNS Chain $\{J_k, M_k\}_{k=1}^K$ with large K, the proportions $\mathcal{P}$ of state A, B, and C in the Markov chain are

$$\begin{aligned}
&\mathcal{P}_{\text{Basic PNS}}(A) = \frac{\sum_{J_k=A} M_k}{\sum_{k=1}^K M_k} = \frac{1}{1 + \frac{5}{4} + \frac{9}{4}} = \frac{2}{9} \neq \pi(A) = \frac{1}{6};\\
&\mathcal{P}_{\text{Basic PNS}}(B) = \frac{5}{18} \neq \pi(B) = \frac{1}{3};\\
&\mathcal{P}_{\text{Basic PNS}}(C) = \frac{1}{2} = \pi(C). \text{ For state C, it is just a coincidence}
\end{aligned} \tag{4}$$

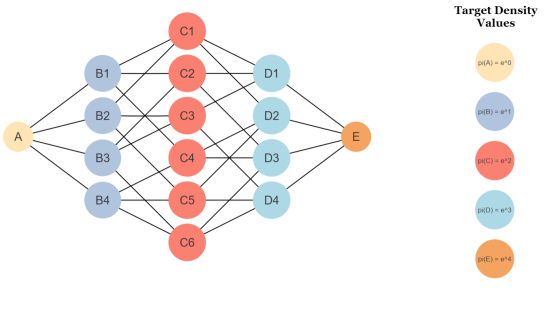

Fig. 2 Diagram of Example 2 showing non-convergence property of the Basic PNS.

This example shows that the samples from Basic PNS are not converging to the target density π.

2.2 Example 2 of the Nonconvergence problem by Basic PNS

The second example is shown in Figure 2, which is a much larger problem compared to the first example. We have 16 states in example 2. All States are connected to exactly four states. The target density is described as $\pi(A) \propto 1$, $\pi(B_1) = \pi(B_2) = \pi(B_3) = \pi(B_4) \propto e$, $\pi(C_1) = \pi(C_2) = \cdots = \pi(C_6) \propto e^2$, $\pi(D_1) = \pi(D_2) = \pi(D_3) = \pi(D_4) \propto e^3$, and $\pi(E) \propto e^4$. This example is too large to be calculated by hand, so we use simulations to calculate the limiting distribution of the samples. The convergence of the sampling distribution is measured by the Total Variation Distance (TVD).

Given the Markov chain $\{X_k\}_{k=1}^K$ generated by Metropolis algorithm, the sampling distribution is defined as $\mathcal{P}_{\text{Sampled}}(x) = \frac{\sum_{k=1}^K \mathbb{1}(X_k=x)}{K}$, $\forall x \in \mathcal{S}$, where $\mathbb{1}$ represents the indicator function. In addition, for the jump chain $\{J_k, M_k\}_{k=1}^K$ generated by either Rejection-Free or PNS, the sampling distribution is defined as $\mathcal{P}_{\text{Sampled}}(x) = \frac{\sum_{k=1}^K M_k \times \mathbb{1}(J_k=x)}{\sum_{k=1}^K M_k}$, $\forall x \in \mathcal{S}$. The corresponding TVD values in both cases are defined as

$$\text{TVD}(\mathcal{P}_{\text{Sampled}}, \pi) = \frac{1}{2} \sum_{x \in \mathcal{S}} \left| \mathcal{P}_{\text{Sampled}}(x) - \pi(x) \right|. \tag{5}$$

According to the definition, TVD is strictly between $[0, 1]$. When the sampling distribution $\mathcal{P}_{\text{Sampled}}$ gets closer to the target distribution π, TVD will decrease to 0. In other words, convergence to stationarity is described by how quickly TVD decreases to 0.

The simulation results are shown in Figure 3. For a given amount of samples ($K = 50, 100, 150, 200, \ldots, 500, 1000, 1500, 2000, \ldots, 7500$), we did 1000 simulations for each of them. The TVD values and the times here are the average values from these 1000 simulations. We compared four methods:

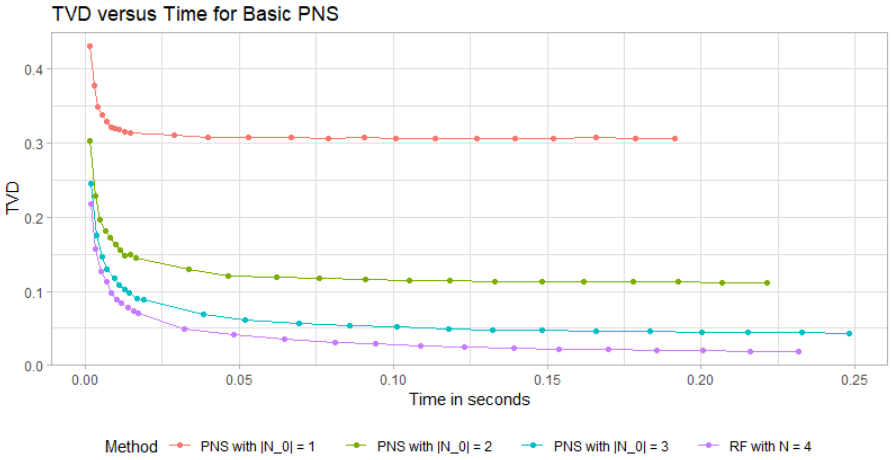

Fig. 3 Average values of TVD between samples and the target density π for Example 2 as a function of average CPU time in seconds for four scenarios: Rejection-Free and Basic PNS with three different Partial Neighbor Set sizes. Each dot within the plot represents the result of the average TVD value and average CPU time in seconds from 1000 simulation runs given a certain original sample size, where the sizes are $\{50, 100, 150, 200, \ldots, 500, 1000, 1500, 2000, \ldots, 7500\}$.

Rejection Free and Basic PNS with three different subset sizes. The Markov chains, produced by Rejection-Free, will converge to the target density, so the TVD value gets close to 0 at last. For PNS with $|\mathcal{N}_0| = 1$, we select one random neighbor among all four neighbors at a time, forcing the chain to move to that state. This method is the worst, and it converges at around 0.3. PNS with $|\mathcal{N}_0| = 2$ means that we randomly select two neighbors at every step and apply the Rejection-Free technique (select from these two states by probability proportional to the transition probability, and calculate the multiplicity list by the average of the transition probabilities). All three PNS algorithms are not converging to the target density π.

Both Examples show that the samples from Basic PNS will not converge to the target distribution π. Thus, we turn attention to a more promising avenue, the unbiased version of the Partial Neighbor Search algorithm, where convergence to stationarity is guaranteed.

3 Unbiased Partial Neighbor Search algorithm

We first need to review the alternating chains technique for the Rejection-Free algorithm (Rosenthal et al, 2021) to introduce our upgraded algorithm version.

3.1 Alternating Chains for Rejection-Free

We may wish to switch between two or more different proposal distributions for the M-H algorithm. An example of the M-H algorithm with alternating

chains for every L_0 steps among $\mathcal{I}$ proposal distributions $\mathcal{Q}^0, \mathcal{Q}^1, \ldots, \mathcal{Q}^{\mathcal{I}-1}$ is shown as Algorithm 4.

Algorithm 4 Metropolis-Hasting algorithm with Alternating Chains

initialize $i \leftarrow 0$ ▷ start with proposal distribution $\mathcal{Q}^0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining samples
initialize X_0
for k in 1 to K **do**
  random Y based on $\mathcal{Q}^i(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \frac{\pi(Y)\mathcal{Q}^i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}^i(X_{k-1}, Y)}$ **then**
    ▷ accept with probability $\min\left\{1, \frac{\pi(Y)\mathcal{Q}^i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}^i(X_{k-1}, Y)}\right\}$
    $X_k \leftarrow Y_k$ ▷ accept and move to state Y
  else
    $X_k \leftarrow X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
  $L \leftarrow L - 1$ ▷ one less remaining sample from the proposal distribution
  if $L = 0$ **then** ▷ if we don't have enough remaining samples
    $i \leftarrow i + 1 \mod \mathcal{I}$ ▷ switch to the next proposal distribution
    $L \leftarrow L_0$ ▷ L_0 remaining states for the next proposal distribution
  end if
end for

However, if we proceed with alternating chains naively for Rejection-Free, it can lead to bias. For each proposal distribution $\mathcal{Q}_i$, we need to get the same amount of samples by the original sample size ($\sum_{k=1}^{K} M_k$) instead of the jump sample size (K) to fix the bias problem. For $\mathcal{I}$ proposal distributions $\mathcal{Q}_0, \mathcal{Q}^1, \ldots, \mathcal{Q}^{\mathcal{I}-1}$, the corresponding neighbor sets are $\mathcal{N}^0, \mathcal{N}^1, \ldots, \mathcal{N}^{\mathcal{I}-1}$ where $\mathcal{N}^i(x) = \{y : y \in \mathcal{S}, \mathcal{Q}^i(x, y) > 0\}$ for $i = 0, 1, \ldots, \mathcal{I} - 1$. Then, if we choose to switch between proposal distributions for L_0 original samples, we can do alternating chains in a Rejection-Free manner as Algorithm 5.

Algorithm 5 is equivalent to Algorithm 4 except that algorithm 5 computes immediate repeated state for each proposal distribution all at once. As such, it has no bias, is consistent, and will converge to the target distribution correctly.

3.2 Alternating Chains for Partial Neighbor Search

Alternating Chains can also be applied to PNS. We first define the meaning of Partial Neighbor Sets here. For simplicity, we focus on discrete cases here and will define the Partial Neighbor Sets for general state space in Appendix A.

Before we start our Markov chain, we have a proposal distribution $\mathcal{Q}$ with a corresponding neighbor set $\mathcal{N}$ where $\mathcal{N}(x) \coloneqq \{y \in \mathcal{S} \mid \mathcal{Q}(x, y) > 0\}$. A Partial Neighbor Set means any function $\mathcal{N}_i$ satisfies the following conditions:

Algorithm 5 Rejection Free algorithm with Alternating Chains

initialize $i \leftarrow 0$ ▷ start with proposal distribution $\mathcal{Q}^0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining original samples
initialize J_0
for k in 1 to K **do**
calculate multiplicity list $m \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}^i(J_{k-1})} \mathcal{Q}^i(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}^i(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}^i(J_{k-1}, z)}\right\}$$

if $m \leq L$ **then** ▷ if we have enough remaining original samples
$M_{k-1} \leftarrow m$, $L \leftarrow L - m$
choose the next jump chain State $J_k \in \mathcal{N}^i(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}^i(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}^i(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}^i(J_{k-1}, y)}\right\}$$

else ▷ if we don't have enough remaining original samples
$M_{k-1} \leftarrow L$, $L \leftarrow L_0$, $J_k \leftarrow J_{k-1}$, $i \leftarrow (i + 1 \mod \mathcal{I})$
▷ stay at J_{k-1} for L times and switch to the next $\mathcal{N}^i$
end if
end for

1. $\mathcal{N}_i : \mathcal{S} \to \mathbf{P}(\mathcal{S})$, where $\mathcal{S}$ is the state space, and $\mathbf{P}(\mathcal{S})$ is the power set of $\mathcal{S}$;
2. $\mathcal{N}_i(x) \subset \mathcal{N}(x)$, $\forall x \in \mathcal{S}$;
3. $y \in \mathcal{N}_i(x) \iff x \in \mathcal{N}_i(y)$, $\forall x, y \in \mathcal{S}$;

Usually, we want to pick $\mathcal{N}_i$ such that $|\mathcal{N}_i(x)| < |\mathcal{N}(x)|$ to perform proper PNS. In addition, to insure irreducibility, we need to make sure $\cup_{i=0}^{\mathcal{I}-1}\mathcal{N}_i(x) = \mathcal{N}(x)$ for all $x \in \mathcal{S}$. The corresponding proposal distribution is defined to be $\mathcal{Q}_i(x, y) : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, where $\mathcal{Q}_i(x, y) \propto \mathcal{Q}(x, y)$ for $y \in \mathcal{N}_i(x)$ and $\mathcal{Q}_i(x, y) = 0$ otherwise;

Therefore, we propose the Unbiased Partial Neighbor Search (Unbiased PNS) with Alternating Chains for every L_0 original samples as shown in Algorithm 6. The proof that the Markov chain produced by Unbiased PNS will converge to the target distribution π is shown in Appendix A.

Again, in Algorithm 6, when we need to pick our next state according to the given probabilities, we can use the technique shown in Appendix B, which is faster than other methods to sample proportionally.

The Markov chains produced by Algorithm 6 will converge to the target distribution, but how is its efficiency compared to the Metropolis-Hasting algorithm and Rejection-Free? We will compare these three algorithms with some simulations in Section 4.

Algorithm 6 Unbiased Partial Neighbor Search

select $\mathcal{N}_i$ for $i = 0, 1, \ldots, \mathcal{I}-1$ where $\cup_{i=0}^{\mathcal{I}-1}\mathcal{N}_i(X) = \mathcal{N}(X)$
initialize $i \leftarrow 0$ ▷ start with proposal distribution $\mathcal{Q}_0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining original samples
initialize J_0
for k in 1 to K **do**
calculate multiplicity list $m \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}_i(J_{k-1})} \mathcal{Q}_i(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}_i(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_i(J_{k-1}, z)}\right\}$$

if $m \leq L$ **then** ▷ if we have enough remaining original samples
$M_{k-1} \leftarrow m$, $L \leftarrow L - m$
choose the next jump chain State $J_k \in \mathcal{N}_i(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}_i(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}_i(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_i(y, J_{k-1})}\right\}$$

else ▷ if we don't have enough remaining original samples
$M_{k-1} \leftarrow L$, $L \leftarrow L_0$, $J_k \leftarrow J_{k-1}$, $i \leftarrow (i+1 \mod \mathcal{I})$
▷ stay at J_{k-1} for L times and switch to the next $\mathcal{N}_i$
end if
end for

4 Application to QUBO model

Quadratic unconstrained binary optimization (QUBO) has been rising in importance in combinatorial optimization because of its wide range of applications in finance and economics to machine learning (Kochenberger et al, 2014). It can also be used as a sampling question, which aims to sample from the distribution

$$\pi(x) = \exp\{x^T Q x\}, \text{ where } x \in \{0,1\}^N \tag{6}$$

for a given N by N matrix Q (usually symmetric or upper triangular).

To run our algorithm, we used uniform proposal distributions among all neighbors where the neighbors are defined as binary vectors with Hamming distance 1. That is, $\mathcal{Q}(x, y) = \frac{1}{N}$ for $\forall y$ such that $|x - y| = \sum_{i=1}^{N} |x_i - y_i| = 1$, $\forall x, y \in \{0,1\}^N$. Thus, the neighbors are all binary vectors different by one flip. For the first simulation here, the PNS neighbor sets $\mathcal{N}_0$, $\mathcal{N}_1$ are chosen systematically, where $\mathcal{N}_0$ represents flip entries from 1 to $\lfloor \frac{N}{2} \rfloor$, and $\mathcal{N}_1$ represents flip entries from $\lfloor \frac{N}{2} \rfloor + 1$ to N. We will discuss many other choices for the PNS neighbor sets in Section 5

Figure 4 shows the results for comparing the Metropolis algorithm, Rejection-Free, and Unbiased PNS by sampling from a 16×16 QUBO question

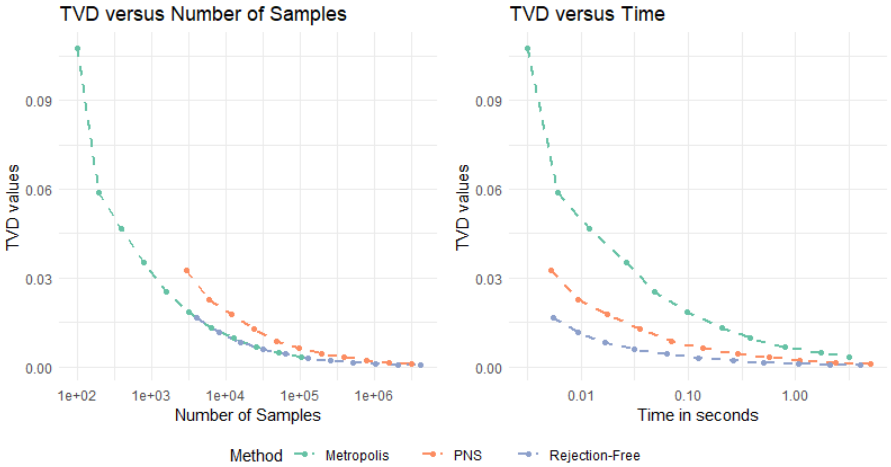

Fig. 4 Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for three methods: Metropolis algorithm, Rejection-Free, and Unbiased PNS. We used upper triangular 16×16 QUBO matrix, generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the result of the average TVD value and time used for 1000 simulation runs given certain original sample sizes. The original sample sizes for the Metropolis algorithm are $\{300, 600, 1200, 2400, \ldots, 3072000\}$. The original samples from Rejection-Free are 40x more than those from Metropolis, and the original samples from Unbiased PNS are 30x more than those from Metropolis. We choose these sizes to get a close average CPU time for all three methods. For Unbiased PNS, we used $|\mathcal{N}_k| = 8$ and $L_0 = 100$.

to a single-core implementation. The QUBO matrix Q is an upper triangular matrix, where the non-zero elements were generated randomly by $Q_{i,j} \sim \text{Normal}(0, 10^2)$, $\forall i \leq j$. We compare the TVD values from the Metropolis Algorithm, Rejection-Free, and Unbiased PNS with different original sample sizes. For the Metropolis algorithm, the numbers of original samples are $\{100, 200, 400, 800, \ldots, 102400\}$. The number of original samples for Rejection-Free is 40 times more than the number for the Metropolis algorithm, and the number for Unbiased PNS is 30 times more. We used these many numbers of original samples to make the run-time for all three algorithms to be about the same. For each given number of original sample sizes, we simulated 1000 runs, recorded the corresponding TVD values and times used for the sampling part, and calculated the average values given the number of original samples. Note that the average time represents the CPU time for where the algorithm is calculated by running the algorithm on a single-core implementation. In addition, before we generate the samples, we apply the algorithm for the same number of steps for burn-in.

From Figure 4, we can see that the quality of the samples by the Metropolis algorithm and Rejection-Free are the same given the original sample sizes. This result is consistent with our conclusion that Rejection-Free is identical to the Metropolis algorithm, except Rejection-Free generates the same states simultaneously with all immediately repeated states. Thus, these two algorithms

are different only by the CPU time. In addition, the quality of the samples by Unbiased PNS is worse than both the Metropolis algorithm and Rejection-Free given a certain number of original samples because each Partial Neighbor Set is biased within its L_0 original samples, while the combination of them is unbiased. Thus, the average TVD value for Unbiased PNS is more significant for the same amount of original samples. However, for a given amount of CPU time, the performance of Unbiased PNS is much better than the Metropolis algorithm and worse than Rejection-Free.

In this case, Unbiased PNS can provide significant speedups compared to the Metropolis algorithm. On the other hand, we did not expect the Unbiased PNS can beat Rejection-Free under this circumstance. Unbiased PNS is worse than Rejection-Free in two aspects. First, the Unbiased PNS is biased within each L_0 original samples. In addition, at the end of each L_0 original samples, the algorithm is very likely to reject once and stay in the same state. Thus, Unbiased PNS is not entirely rejection-free anymore and usually rejects once for every L_0 original samples.

However, we need the Unbiased PNS because we may not have as many circuit blocks in the parallelism hardware as we want. Thus, we can, at most, consider a limited number of neighbors for some specialized hardware, such as DA. Thus, Rejection-Free is not applicable in this case, and we would need the help of Unbiased PNS, which is better than applying the Metropolis algorithm.

Again, parallelism in computer hardware can increase the speed for both Rejection-Free and Unbiased PNS by mapping the calculation of the transition probabilities for different neighbors onto different cores (Rosenthal et al, 2021). Besides that, we can also use multiple replicas at different temperatures, such as in parallel tempering, or deploy a population of replicas at the same temperature (Sheikholeslami, 2021). Combining these methods by parallelism can yield 100x to 10,000x speedups for both Rejection-Free and Unbiased PNS (Sheikholeslami, 2021).

5 Optimal Choice for the Partial Neighbors

In the section 4, we used two systematically pre-selected neighbor sets $\mathcal{N}_0$, $\mathcal{N}_1$. However, for the optimization version of the QUBO question, we concluded that random Partial Neighbor Sets are better than systematic Partial Neighbor Sets; see Chen et al (2022). Thus, we compare two ways of choosing partial neighbor sets here: systematic and random. For simplicity, assume that we have N neighbors for all states, and we use Unbiased PNS neighbor sets of size n. Therefore, we have $\binom{N}{n}$ different partial neighbor sets. For systematic method, we choose $\mathcal{I}$ PNS neighbor sets $\{\mathcal{N}_i\}_{i=1}^{\mathcal{I}}$, where $\cup_{i=1}^{\mathcal{I}}\mathcal{N}_i(x) = \mathcal{N}(x)$. We proceed with each Partial Neighbor Sets within the loop for L_0 original samples. We use the notation $\mathcal{N}_i(x)$ for systematic Partial Neighbor Sets because $\mathcal{N}_i(x)$ is pre-determined for $i = 1, 2, \ldots, \mathcal{I}$. On the other hand, for random Partial Neighbor Sets, we choose a new set $\mathcal{N}_k$ from all $\binom{N}{n}$ potential Partial Neighbor Sets after each L_0 original samples. We use the notation

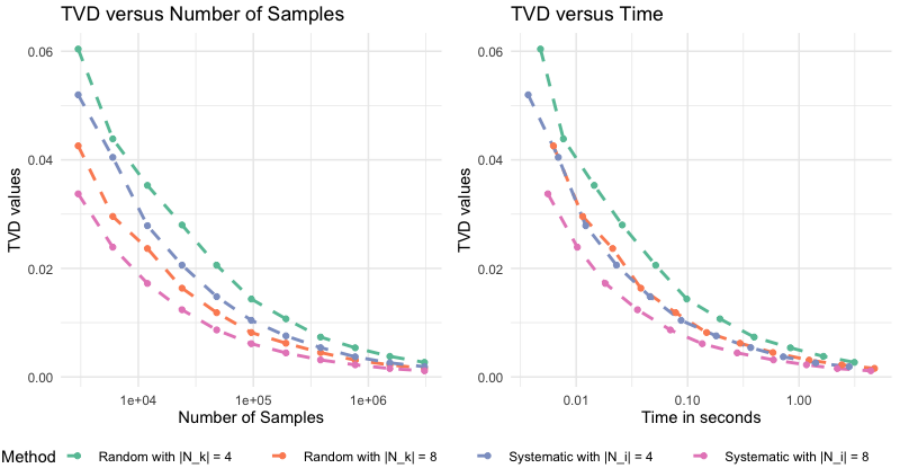

Fig. 5 Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for four scenarios: Systematic PNS and Random PNS, each with Partial Neighbor Set sizes of 4 and 8. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. For all PNS, we used $L_0 = 100$.

$\mathcal{N}_k(x)$ for random partial neighbor sets because $\mathcal{N}_k(x)$ can be different for every PNS step, and the subscript k represents the special partial neighbor set for step k. For both methods, $\mathcal{Q}_i(x, y), \mathcal{Q}_k(x, y) \propto \mathcal{Q}(x, y)$ for $y \in \mathcal{N}_i(x)$, and $\mathcal{Q}_i(x, y) = \mathcal{Q}_k(x, y) = 0$ otherwise.

To compare the above two methods of selecting Partial Neighbor Sets, we apply them to the previous 16×16 QUBO question, and we test the following four scenarios:

- two systematic partial neighbor sets where the first set considers flipping the first half of the bits, and the second set considers flipping the second half of the bits;
- four systematic partial neighbor sets where each set considers flipping a quarter of the bits;
- random partial neighbor sets with $\frac{N}{2}$ partial neighbors; that is, each set considers flipping a random set of bits with size $\frac{N}{2}$;
- random partial neighbor sets with $\frac{N}{4}$ partial neighbors; that is, each set considers flipping a random set of bits with size $\frac{N}{4}$;

The result is shown in Figure 5; for this case, systematic Partial Neighbor Sets are better than random Partial Neighbor Sets. However, random Partial Neighbor Sets can be better when we run the same code with a different random seed. After running this simulation for 100 different random seeds, the systematic neighbor sets are better 56 times. Thus, we conclude that the

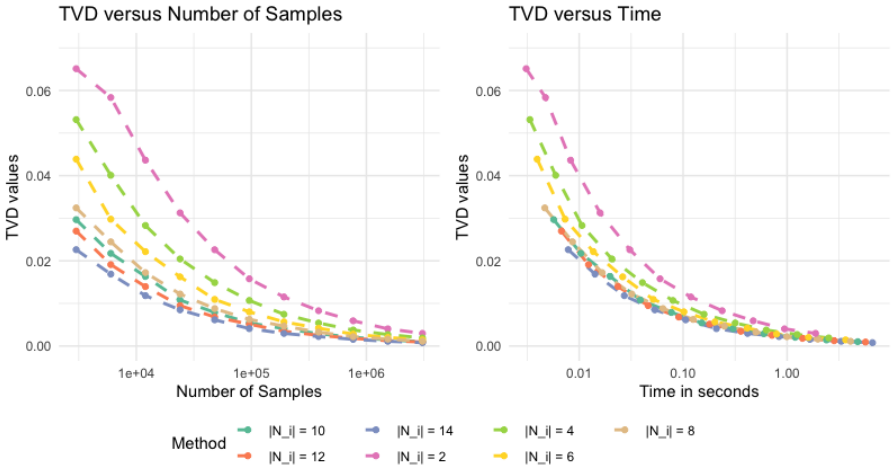

Fig. 6 Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for four scenarios: Unbiased PNS with different partial neighbor set sizes $\{2, 4, 6, \ldots, 14\}$. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. For all PNS, we used $L_0 = 100$.

performance of these two Partial Neighbor Sets is close to each other. We will continue using the systematic Partial Neighbor Sets in our later simulation.

In previous simulations, we naively use we used $|\mathcal{N}_i(x)| = 4$ or 8 and $L_0 = 100$ in previous examples. What is the optimal choice for $|\mathcal{N}_i(x)|$? We want to compare $|\mathcal{N}_i(x)| = 2, 4, 6, 8, \ldots, 14$ by the QUBO question. In previous cases, we only used the systematic Partial Neighbor Set size n that can be divided evenly by N. For other n such as 14, we used the following Partial Neighbor Sets: first we consider flipping bits 1 to 14, then we consider bits 15, 16, and 1 to 12, then 13, 14, 15, 16, and 1 to 10, etc. We used eight systematic Partial Neighbor Sets of size 14.

Figure 6 shows the results for comparing the Unbiased PNS with $|\mathcal{N}_i| \equiv 2$, 4, 6, 8, 10, 12, and 14 for $\forall X \in \{0, 1\}^{16}$. Every other simulation setting is the same as the previous simulations for the QUBO question. The choice of L_0 is still 100. According to the left plot, we can say that given the same amount of original samples, the Markov chain from $|\mathcal{N}_i| = 14$ is the least biased. On the other hand, from the right plot, we can conclude that, given the same amount of CPU time, the sample quality from $|\mathcal{N}_i| = 14$ is the best. In addition, the performances are close to each other for all cases where $|\mathcal{N}_i| \geq 8$. Note that a single-core implementation makes all these comparisons by the CPU time, and parallelism hardware can provide speedups. Intuitively, the more tasks that can be calculated simultaneously, the greater the speedup. Thus, if we apply our Unbiased PNS on parallelism hardware with a limited number of parallel tasks

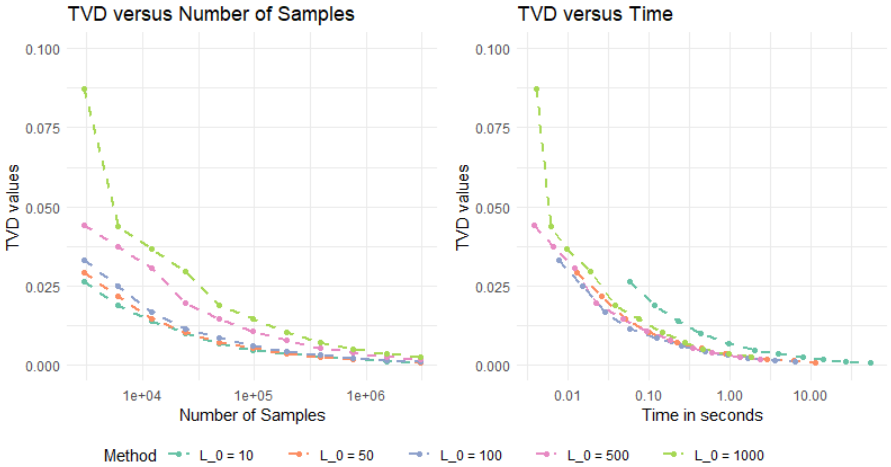

Fig. 7 Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased Partial Neighbor Search with different sizes of L_0. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{300, 600, 1200, 2400, \ldots, 3072000\}$. For all PNS, we used $|\mathcal{N}_i| = 8$.

that can be computed simultaneously, we should choose the largest possible partial neighbor set size $|\mathcal{N}_i|$.

Furthermore, Figure 7 shows the results for comparing the Unbiased PNS with $L_0 = 10$, 50, 100, 500, and 1000. Again, every other settings of the simulation is the same, and $|\mathcal{N}_i|$ is still 8, $\forall x \in \{0,1\}^{16}$. The left plot shows that given the same samples, the Markov chain from $L_0 = 10$ is the least biased. However, the right plot shows that, given the same amount of CPU time, the TVD values are about the same except $L_0 = 10$. The case with $L_0 = 100$ is slightly better than the other cases, but the difference is not too large. $L_0 = 10$ becomes the worst since such L_0 has too many rejections (about one rejection for every ten samples). Thus, in practice, the choice of L_0 is not that important as long as it is not extreme.

6 Continuous Models

We talked about the application of Unbiased PNS to discrete cases in the previous sections. Can we apply Unbiased PNS on continuous models? We first review how to apply Rejection-Free on general (continuous) state space as in Theorem 13 from Rosenthal et al (2021).

Let $\mathcal{S}$ be a general state case, and μ a σ-finite reference measure on $\mathcal{S}$. Suppose a Markov chain on $\mathcal{S}$ has transition probabilities $P(x, dy) \propto q(x, y)\mu(dy)$ for $q : \mathcal{S} \times \mathcal{S} \to [0, 1]$. Again let $\hat{P}$ be the transitions for the corresponding jump chain J_k with multiplicities M_k. Then:

1. $\hat{P}(x, \{x\}) = 0$, and for $x \neq y$, $\hat{P}(x, dy) = \frac{q(x,y)}{\int q(x,z)\mu(dz)}\mu(dy)$
2. The conditional distribution of M_k given J_k is equal to the distribution of $1 + G$ where G is a geometric random variable with success probability $p = \alpha(J_k)$ where $\alpha(x) = P[X_{k+1} \neq x \mid X_k = x] = \int q(x, z)\mu(dz) = 1 - r(x) = 1 - P(x \mid x)$
3. If the original chain is ϕ-irreducible (see, e.g., Meyn and Tweedie (2012)) for some positive σ-finite measure ϕ on $\mathcal{X}$, then the jump chain is also ϕ-irreducible for the same ϕ.
4. If the original chain has stationary distribution $\pi(x)\mu(dx)$, then the jump chain has stationary distribution given by $\hat{\pi}(x) = c\alpha(x)\pi(x)\mu(dx)$ where $c^{-1} = \int \alpha(y)\pi(y)\mu(dy)$
5. If $h : \mathcal{S} \to \mathbb{R}$ has finite expectation, then with probability 1,

$$\lim_{K\to\infty} \frac{\sum_{k=1}^{K} M_k h(J_k)}{\sum_{k=1}^{K} M_k} = \lim_{K\to\infty} \frac{\sum_{k=1}^{K} [\frac{h(J_k)}{\alpha(J_k)}]}{\sum_{k=1}^{K} [\frac{1}{\alpha(J_k)}]} = \pi(h) := \int h(x)\pi(x)\mu(dx)$$

Although we have a solid theory base for Rejection-Free on general state space, applying Rejection-Free to the continuous sampling questions efficiently on most computer hardware is pretty hard. The biggest challenge is the calculation of integration $\int q(x, z)\mu(dz)$. We need many calculations for the numerical integration. In addition, such tasks can hardly be split efficiently into specialized hardware with a reasonable amount of parallel calculating units. At the same time, Unbiased PNS can be surprisingly helpful in this case. As long as the Metropolis algorithm can be applied, PNS can be applied straightforwardly without any calculation of integration. We need to choose the Partial Neighbors Sets $\mathcal{N}_i(x)$ to be a finite subset of all the neighbors $\mathcal{N}(x)$ in Algorithm 6. We check the performance of our Unbiased PNS on a simple continuous sampling question: the Donuts Example.

7 Application to the Donuts Example

Inspired by Feng (2021), we use a donuts example to show Unbiased PNS's performance on continuous state space. Suppose we have two independent random variables μ and θ where

$$\mu \sim \text{Normal}^{+}(\mu_0, \sigma^2),\ \theta \sim \text{Uniform}[0, \pi). \tag{7}$$

Here, Normal^{+} means the Truncated Normal distribution without the negative tail, and π in the Uniform distribution means the circular constant instead of the target density. Then we define two random variables X_1 and X_2 to be

$$X_1 = \sqrt{\mu}\sin\theta,\ X_2 = \sqrt{\mu}\cos\theta. \tag{8}$$

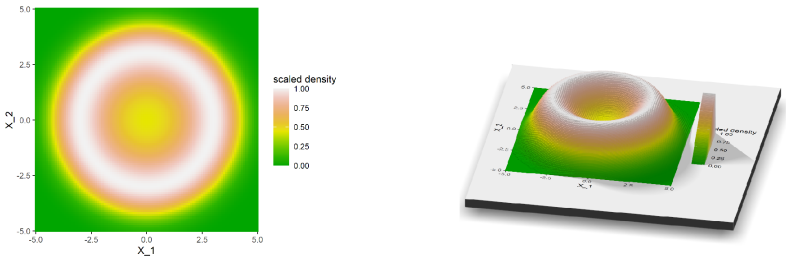

Fig. 8 The scaled probability density plot for the Donuts Example with $\mu_0 = 9$ and $\sigma = 10$. The density is scaled to $[0, 1]$. We used large σ to show the shape of our distribution. With small σ, it is hard to see the shape of a sharply peaked distribution.

The determinant of the Jacobian matrix is $\frac{1}{2}$. Thus we have

$$f_{X_1,X_2}(x_1, x_2) \propto \frac{1}{\sigma} \exp\left[- \frac{(x_1^2 + x_2^2 - \mu_0)^2}{2\sigma^2} \right], \tag{9}$$

For example, a 3-D map for the density for X_1 and X_2 with $\mu_0 = 9$, and $\sigma = 10$ is shown in Figure 8. The density is scaled to $[0, 1]$. In our later simulation, we use $\mu_0 = 9$ and $\sigma = 0.1$ instead. We use large σ to show the shape of our distribution because it is hard to see its shape when it sharply peaks with a small σ. However, for the simulation, PNS can outperform the Metropolis algorithm when there are many rejections, so we use a small σ to get a sharply peaked distribution to increase the rejection rate in the Metropolis algorithm. Note that Unbiased PNS and Rejection-Free are not always better than the Metropolis algorithm. For an extreme example, when we have a distribution where all the states have the same target density values, there will be no rejection for the Metropolis algorithm. At each step, the Metropolis algorithm will uniformly pick a random neighbor from the current state and move to that neighbor, while (Rejection-Free / PNS) will calculate the transition probabilities for (all / part) of the neighbors and uniformly pick a random one. The Metropolis algorithm will be far better than Rejection-Free and PNS in this case. In practice, the higher the dimension of the problem and the more sharply peaked the distribution is, the better the Rejection-Free and Unbiased PNS will be. Thus, we use $\sigma = 0.1$ for the simulation to create a sharply peaked distribution for later simulation.

In addition, the proposal distribution is defined to be the standard normal distribution for both dimensions. That is, for $x = (x_1, x_2), y = (y_1, y_2) \in \mathbb{R}^2$, $\mathcal{Q}(x, y) = \phi(y_1 - x_1)\phi(y_2 - x_2)$ where ϕ is the density function of the standard normal distribution. Then for any $x \in \mathbb{R}^2$, we have $\mathcal{N}(x) = \mathbb{R}^2$.

Since the Partial Neighbor Sets are always the whole space of $\mathbb{R}^2$, it is tough for us to apply Rejection-Free here since the integration of the whole space needs too many computational resources. Even if we limit the neighbors

to a small area around the current state, integration is needed as long as the problem is continuous, and the Rejection-Free will be consequentially slow. At the same time, Unbiased PNS can be applied to continuous cases without calculating integration by making minor changes to Algorithm 6. The Unbiased PNS algorithm for continuous is stated as Algorithm 7. In Algorithm 7, we did not define the systematic Partial Neighbor Sets as we had for the discrete cases. We want to use Unbiased PNS with finite many partial neighbors being considered at each step, but we have uncountable neighbors. It is impossible to divide these uncountable neighbors into finite partial neighbor sets with finite sizes. Thus, we can only use the random Partial Neighbor Set, which randomizes a new finite partial neighbor set for every L_0 original samples. In later simulation, we use partial neighbor sets with $|\mathcal{N}_k| = 50$. That is, we consider 50 partial neighbors at each step. Note that, in Section 3.2, we defined the Partial Neighbor Sets, and according to the third condition, we must have reversibility for all $\mathcal{N}_i(x)$, which means $y \in \mathcal{N}_i(x) \iff x \in \mathcal{N}_i(y)$, $\forall x, y \in \mathcal{S}$. Therefore, we choose the Partial neighbor Set $\mathcal{N}_i(x)$ as follows:

1. generate $\delta_1, \delta_2 \sim \text{Normal}(0, 1)$;
2. for state $x = (x_1, x_2)$, put $y = (x_1 + \delta_1, x_2 + \delta_2)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;
3. to ensure the reversibility, also put $y' = (x_1 - \delta_1, x_2 - \delta_2)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;
4. repeats the above steps 25 times to generate a total of 50 neighbors for the Partial Neighbor Set $\mathcal{N}_i(x)$.

In addition, L_0 is selected to be 1000. Using $L_0 = 100$ to 1000 will not affect the sampling speed too much, similar to the conclusion in Section 5.

Moreover, we measure the sampling results by bias instead of the TVD. The calculation of TVD in the continuous case also needs much integration, which is hard to calculate. On the other hand, given samples $\{X_1, X_2 \ldots, X_K\}$, we usually use the MCMC to approximate the expected value $\mathbb{E}_\pi(h)$ of a function $h : S \to \mathbb{R}$ by the usual estimator, $\hat{e}_K(h) = \frac{1}{K}\sum_{k=1}^{K} h(X_{1,k}, X_{2,k})$. The Strong Law of Large Numbers for Markov chains says that assuming that $\mathbb{E}_\pi(h)$ is finite and that the Markov chain is irreducible with stationary distribution π, we must have $\lim_{K\to\infty} \hat{e}_K = \mathbb{E}_\pi(h)$. Therefore, $\text{Bias}(h) = |\hat{e}_K(h) - \mathbb{E}_\pi(h)| = |\frac{1}{K}\sum_{k=1}^{K} h(X_k) - \mathbb{E}_\pi(h)|$ can also be a good measurement for the quality of the samples. According to the definition, bias is greater or equal to 0. When the samples $\{X_1, X_2 \ldots, X_K\}$ gets closer to the target distribution π, the bias will decrease to 0. Thus, convergence to stationarity is described by how quickly the bias decreases to 0 for all function h. This property is similar to TVD from Section 4. In fact, for any probability distribution $\mathcal{P}_1$ and $\mathcal{P}_2$, $\text{TVD}(\mathcal{P}_1, \mathcal{P}_2) = \sup_{\mathcal{S}} \big(\mathcal{P}_1(\mathcal{S}), \mathcal{P}_2(\mathcal{S})\big)$ (Chen et al, 2016).

For example, we check the sum of the bias from the first-degree terms X_1 and X_2. Since the Donuts example is centered at 0, thus $\mathbb{E}_\pi(X_1) = \mathbb{E}_\pi(X_2) = 0$.

Algorithm 7 Unbiased PNS for Continuous Case

select one Partial Neighbor Set $\mathcal{N}_0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining original samples
initialize J_0
for k in 1 to K **do**
calculate multiplicity list $m \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}_0(J_{k-1})} \mathcal{Q}_0(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}_0(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_0(J_{k-1}, z)}\right\}$$

if $m \leq L$ **then** ▷ if we have enough remaining original samples
$M_{k-1} \leftarrow m$, $L \leftarrow L - m$
choose the next jump chain State $J_k \in \mathcal{N}_0(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}_0(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}_0(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_0(y, J_{k-1})}\right\}$$

else ▷ if we don't have enough remaining original samples
$M_{k-1} \leftarrow L$, $L \leftarrow L_0$, $J_k \leftarrow J_{k-1}$,
▷ stay at J_{k-1} for the remaining L times
select a new Partial Neighbor Set $\mathcal{N}_0$
end if
end for

Thus, we have

$$\begin{aligned} \text{Bias}(X_1) + \text{Bias}(X_2) &= \hat{e}_K(X_1) - \mathbb{E}_\pi(X_1) + \hat{e}_K(X_2) - \mathbb{E}_\pi(X_2) \\ &= \left|\frac{1}{K}\sum_{k=1}^{K} X_{1,k}\right| + \left|\frac{1}{K}\sum_{k=1}^{K} X_{2,k}\right|, \end{aligned} \tag{10}$$

Note that both biases will decrease to 0 if our Markov chain converges to the target density π. In addition, for the Rejection-Free Chain $\{J_{1,k}, J_{2,k}, M_k\}_{k=1}^{K}$ generated by the Unbiased PNS algorithm, the bias is defined to be

$$\text{Bias}(J_1) + \text{Bias}(J_2) = \frac{\left|\sum_{k=1}^{K} M_k \times J_{1,k}\right|}{\sum_{k=1}^{K} M_k} + \frac{\left|\sum_{k=1}^{K} M_k \times J_{2,k}\right|}{\sum_{k=1}^{K} M_k} \tag{11}$$

The result for comparing the Metropolis algorithm and Unbiased PNS by the bias of first-degree term is shown in Figure 9. Each dot within the plot represents the average value of 100 simulation runs. For each run, we generate a Markov chain for a given number of samples for both algorithms. The average

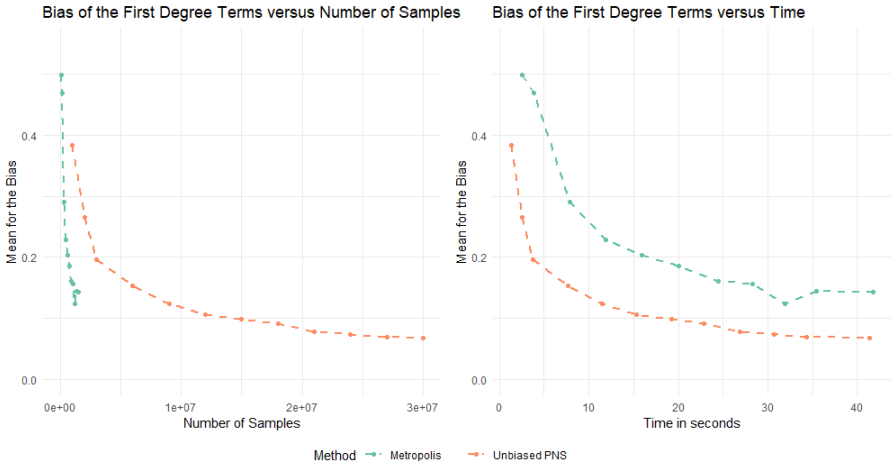

Fig. 9 Sum of the Average Bias of X_1 and X_2 between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for two methods: Metropolis algorithm and Unbiased PNS. We used Donuts example with $r_0 = 10$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 1000 simulation runs given certain original sample sizes. The original sample sizes for the Metropolis algorithm are $\{50000, 100000, 150000, 300000, 450000, \ldots, 1500000\}$. The sizes for Unbiased PNS are 20x more than the sizes for the Metropolis. We choose these sizes to get a close average CPU time for both methods. For Unbiased PNS, we used $|\mathcal{N}_k| = 50$ and $L_0 = 1000$.

time represents the CPU time we apply the algorithm by a single-core implementation. Again, parallelism hardware such as DA can yield 100x to 10,000x speedups for Unbiased PNS (Sheikholeslami, 2021).

From Figure 9, we can see that the quality of the samples by Unbiased PNS is again worse than the Metropolis algorithm because each Partial Neighbors Set is biased within L_0 original samples, while the combination of them is unbiased. Thus, the average bias values for Unbiased PNS are more significant for the same amount of samples. However, for a given amount of CPU time, the performance of Unbiased PNS is much better than the Metropolis algorithm. For this example, the Unbiased PNS can get 30x more samples than the Metropolis algorithm within the same time by a single-core implementation. Rejections slow down the Metropolis algorithm while Unbiased PNS is not influenced, and thus, Unbiased PNS works much better in this simulation.

In addition, we can also check the sum of the bias from the second degree terms $\text{Bias}(X_1^2) + \text{Bias}(X_2^2)$, the sum of the bias from the fourth degree terms $\text{Bias}(X_1^4) + \text{Bias}(X_2^4)$, and the sum of the bias from the positive rate $\text{Bias}(\mathbb{1}(X_1 > 0)) + \text{Bias}(\mathbb{I}(X_1 > 0))$, where $\mathbb{1}$ means the indicator function. To calculate the bias of the second degree terms , we have $X_1^2 + X_2^2 = \mu \sim \text{Normal}^+(\mu_0, \sigma^2)$. Note that, for the Truncated normal distribution with mean 9 and standard deviation 0.1, the probability for a negative tail is too small,

so we can treat it as a normal distribution. Thus,

$$\mathbb{E}_\pi(X_1^2) = \frac{1}{2}\mathbb{E}_\pi(X_1^2 + X_2^2) = \frac{1}{2}\mathbb{E}_\pi(\mu^2) \approx \frac{1}{2}\mu_0^2. \tag{12}$$

$$\begin{aligned}\text{Bias}(X_1^2) + \text{Bias}(X_2^2) &= \hat{e}_K(X_1^2) - \mathbb{E}_\pi(X_1^2) + \hat{e}_K(X_2^2) - \mathbb{E}_\pi(X_2^2) \\ &\approx |\frac{1}{K}\sum_{k=1}^{K} X_{1,k}^2 - \frac{1}{2}\mu_0^2| + |\frac{1}{K}\sum_{k=1}^{K} X_{2,k}^2 - \frac{1}{2}\mu_0^2|.\end{aligned} \tag{13}$$

Similarly,

$$\begin{aligned}\text{Bias}(X_1^4) + \text{Bias}(X_2^4) =& \hat{e}_K(X_1^4) - \mathbb{E}_\pi(X_1^4) + \hat{e}_K(X_2^4) - \mathbb{E}_\pi(X_2^4) \\ \approx& |\frac{1}{K}\sum_{k=1}^{K} X_{1,k}^4 - \frac{3}{8}(\mu_0^4 + \sigma^2)| + \\ & |\frac{1}{K}\sum_{k=1}^{K} X_{2,k}^4 - \frac{3}{8}(\mu_0^4 + \sigma^2)|;\end{aligned} \tag{14}$$

$$\begin{aligned}\text{Bias}(\mathbb{1}(X_1 > 0)) + \text{Bias}(\mathbb{1}(X_2 > 0)) =& \hat{e}_K(\mathbb{1}(X_1 > 0)) - \mathbb{E}_\pi(\mathbb{1}(X_1 > 0)) + \\ & \hat{e}_K(\mathbb{1}(X_2 > 0)) - \mathbb{E}_\pi(\mathbb{1}(X_2 > 0)) \\ =& |\frac{1}{K}\sum_{k=1}^{K} \mathbb{1}(X_{1,k} > 0) - \frac{1}{2}| + \\ & |\frac{1}{K}\sum_{k=1}^{K} \mathbb{1}(X_{2,k} > 0) - \frac{1}{2}|.\end{aligned} \tag{15}$$

The results for the comparison of the Metropolis algorithm and Unbiased PNS by the sum of the average bias from the second degree terms $\text{Bias}(X_1^2) + \text{Bias}(X_2^2)$, the fourth degree terms $\text{Bias}(X_1^4) + \text{Bias}(X_2^4)$, and the positive rate $\text{Bias}(\mathbb{I}(X_1 > 0)) + \text{Bias}(\mathbb{I}(X_1 > 0))$ are shown in Figure 10. From the result for different choices of the terms, we can conclude that Unbiased PNS performs better than the Metropolis algorithm in this continuous Donuts example.

8 Burn In by Partial Neighbor Search

8.1 Optimization instead of Burn-In

In previous sections, when we need K original samples, we have to generate $2K$ original samples. We use the first K original samples as the burn-in part. This way, we started our sampling process from stationarity. However, Geyer (2011) argued that burn-in is not necessary for MCMC. As an alternative to burn-in, any point the researcher does not mind having in a sample is a good starting point. His argument indicates that we can usually start at a point

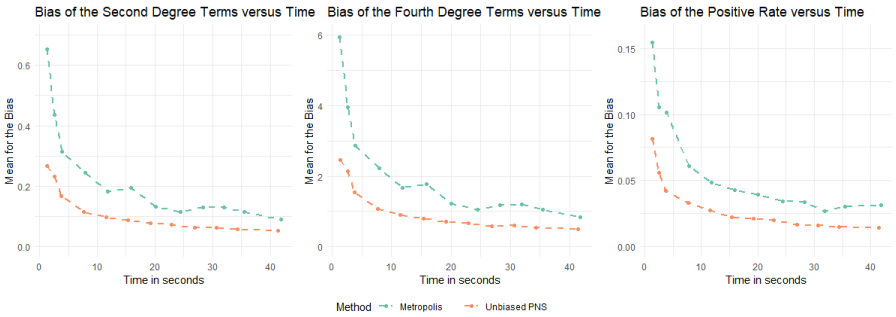

Fig. 10 Sum of the average bias from the second degree terms $\text{Bias}(X_1^2) + \text{Bias}(X_2^2)$ (left), the fourth degree terms $\text{Bias}(X_1^4) + \text{Bias}(X_2^4)$ (middle), and the positive rate $\text{Bias}(\mathbb{1}(X_1 > 0)) + \text{Bias}(\mathbb{1}(X_1 > 0))$ (right) between sampling and target density π as a function of average time in seconds for two methods: Metropolis algorithm and Unbiased PNS. $\mathbb{I}$ means the indicator function. We used Donuts example with $r_0 = 10$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 1000 simulation runs given certain original sample sizes. The original sample sizes for the Metropolis algorithm are $\{50000, 100000, 150000, 300000, 450000, \ldots, 1500000\}$. The sizes for Unbiased PNS are 20x more than the sizes for the Metropolis. We choose these sizes to get a close average CPU time for both methods. For Unbiased PNS, we used $|\mathcal{N}_k| = 50$ and $L_0 = 1000$.

whose target density value is large. Geyer (2011) claimed that this alternative method is usually better than regular burn-in.

We can apply optimization algorithms in Chen et al (2022) before sampling if we accept the above statement. For example, we can cancel the burn-in part of the QUBO question in Section 4. Instead, before we start sampling from the target density, we consider optimization algorithms which tries to maximize $\pi(x) = \exp\{x^T Q x\}$ for $x \in \{0,1\}^N$. Then we can start sampling from a state with a large $\pi(x)$ value, although it may not be optimal. Simulated Annealing is one such algorithm. In addition, we can also use Optimization Rejection-Free and Optimization PNS from Chen et al (2022).

To find x from the state space $\mathcal{S}$ which maximizes $\pi(x)$, given the proposal distribution $\mathcal{Q}$, and the corresponding neighbors $\mathcal{N}$, and a non-increasing cooling schedule $T : \mathbb{N} \to (0, \infty)$, the corresponding algorithms for Simulated Annealing, Optimization Rejection-Free, and Optimization PNS are described in Algorithm 8, 9, and 10.

In Chen et al (2022), we illustrated the superior performance of Optimization PNS with many examples, such as the QUBO question, the Knapsack problem, and the 3R3XOR problem. In all these problems, Optimization PNS is the best algorithm compared to the Simulated Annealing algorithm and Optimization Rejection-Free. See Chen et al (2022) for more details. Therefore, we can use Optimization PNS as in Algorithm 10 to replace the burn-in part before sampling.

However, the starting states obtained by the proposed three optimization algorithms will not converge to the target density. Therefore, people can use these optimization methods to replace the burn-in part only if they believe that the sampling can start without stationarity, just like Geyer (2011).

Algorithm 8 Simulated Annealing

initialize X_0, and $X_{\max} = X_0$
for k in 1 to K **do**
  random $Y \in \mathcal{N}(X_{k-1})$ based on $\mathcal{Q}(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < [\frac{\pi(Y)}{\pi(X_{k-1})}]^{1/T(k)}$ **then**
    ▷ accept with probability $\min\left\{1, \left[\frac{\pi(Y_k)}{\pi(X_{k-1})}\right]^{1/T(k)}\right\}$
    $X_k = Y$ ▷ accept and move to state Y
    if $\pi(Y) > \pi(X_{\max})$ **then**
      $X_{\max} = Y$
    end if
  else
    $X_k = X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
end for

Algorithm 9 Optimization Rejection-Free

initialize J_0, and set $X_{\max} = J_0$
for k in 1 to K **do**
  choose the next jump chain State $J_k \in \mathcal{N}(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, y)}\right\}$$

  if $\pi(J_k) > \pi(X_{\max})$ **then**
    $X_{\max} = J_k$
  end if
end for

Algorithm 10 Optimization Partial Neighbor Search

initialize J_0, and set $X_{\max} = J_0$
for k in 1 to K **do**
  pick the Partial Neighbor Set $\mathcal{N}_k(J_{k-1}) \subset \mathcal{N}(J_{k-1})$
  choose the next jump chain State $J_k \in \mathcal{N}_k(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(y, J_{k-1})}\right\}$$

  if $\pi(J_k) > \pi(X_{\max})$ **then**
    $X_{\max} = J_k$
  end if
end for

8.2 Burn-In until Convergence

Section 8.1, we mentioned that some people believe that MCMC does not necessarily need to start from stationarity. However, some people may insist on starting from stationarity. Then we can combine the algorithm for optimization and sampling and try to take advantage of both versions to get a burn-in algorithm. We can apply the optimization algorithm for a certain number of steps K_0, and then we apply the sampling algorithms such as Rejection-Free (Algorithm 2) or Unbiased PNS (Algorithm 6) for K_1 samples.

To check the distribution of the states after a certain number of steps of the hybrid algorithm, We generate a certain number of Markov chains by the algorithms and record each chain's last state. As a result, we can get the distribution after burn-in, and we call this distribution the starting distribution for sampling. For example, just like the previous example in Section 4, we still consider a 16×16 QUBO question. Every setting is exactly the same as what we have in Section 4 except we used $Q_{i,j} \sim \text{Normal}(0, 1^2)$, $\forall i \leq j$. We didn't use the standard deviation of 10 like Section 4, since we only use the last states from one Markov chain, we generate one such starting distribution with $100,000$ Markov chains and check the TVD value between the starting distribution and the target density. Thus, if we use a standard deviation of 10, we need much a much longer time to get a small TVD value. The number of steps for Optimization PNS K_0 is chosen to be $\lfloor \frac{1}{20} K_1 \rfloor$ ($\lfloor\ \rfloor$ represents the floor function). The number of samples $K_1 = 20, 40, 60, \ldots, 200$. In addition, we also compare Unbiased PNS with Optimization PNS plus Unbiased PNS. Since we believe Unbiased PNS will converge slower than Rejection-Free, so we choose $K_1 = 40, 80, \ldots, 600$, and $K_0 = \lfloor \frac{1}{40} K_1 \rfloor$. The temperature function $T(k)$ in the optimization algorithms is set to be constantly 1. The result is shown in Figure 11.

From Figure 11, algorithms with the help of Optimization PNS converge faster with respect to the CPU time. We used a 16×16 QUBO question here. In addition, we concluded that the higher the dimension is and the more sharply peaked the distribution is, the better the optimization PNS will be (Chen et al, 2022). Optimization PNS performs extremely well in the optimization version of 200×200 QUBO question (Chen et al, 2022). Thus, we can also use the Optimization PNS to help burn-in in high dimension or sharply peaked distributions. Since the calculation of TVD for high dimension problems is infeasible, so we just did a simulation of 16×16 QUBO question here. See Chen et al (2022) for more simulation results from optimization problems with higher dimensions.

9 Conclusion

We introduced three versions of the Partial Neighbor Search algorithms of sampling. Basic PNS is straightforward but does not converge to the target density. The Unbiased PNS will converge to the target density, but it performs worse than Rejection-Free compared to a single-core implementation in the

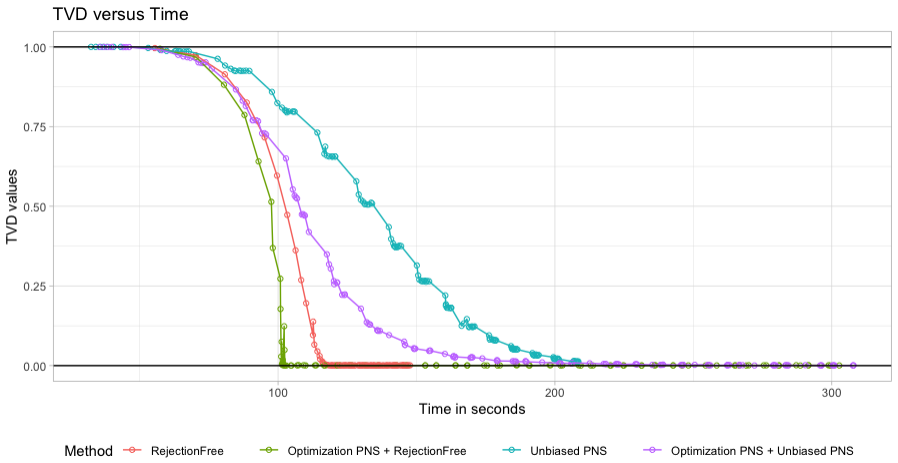

Fig. 11 Average values of TVD between the starting distribution from 100,000 chains and target density π as a function of the average time for the chains in seconds for four methods: Rejection-Free, Optimization PNS plus Rejection-Free, Unbiased PNS, and Optimization PNS plus Unbiased PNS. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 1^2)$ for upper triangular elements. The original sample sizes for Rejection Free are $K_1 = \{20, 30, 40, 50, \ldots, 1000\}$, and the number of steps for the corresponding Optimization PNS is $K_0 = \lfloor \frac{K_1}{20} \rfloor$. The original sample sizes for Unbiased PNS are $K_1 = \{40, 50, 60, \ldots, 1500\}$, and the number of steps the corresponding Optimization PNS is $K_0 = \lfloor \frac{K_1}{40} \rfloor$. Each dot within the plot represents the TVD value between the target distribution π and the distribution of the last state of 100,000 Markov chains.

QUBO question. However, the Unbiased PNS can use specialized parallelism hardware such as DA to improve the sampling efficiency significantly, while Rejection-Free cannot. In addition, Rejection-Free is infeasible in many continuous cases, but the Unbiased PNS can be applied to all continuous cases and works much better than the Metropolis algorithm. Finally, we illustrated that Optimization PNS from Chen et al (2022) can be used to improve the burn-in part before sampling.

Acknowledgments

The authors thank Fujitsu Ltd. and Fujitsu Consulting (Canada) Inc. for providing financial support.

References

Beichl I, Sullivan F (2000) The Metropolis algorithm. Computing in Science & Engineering 2(1):65–69

Bertsimas D, Tsitsiklis J (1993) Simulated annealing. Statistical science 8(1):10–15

Bortz AB, Kalos MH, Lebowitz JL (1975) A new algorithm for monte carlo simulation of ising spin systems. Journal of Computational Physics 17(1):10–18

Chen M, Gao C, Ren Z (2016) A general decision theory for huber's ϵ-contamination model. Electronic Journal of Statistics 10(2):3752–3774

Chen S, Rosenthal JS, Dote A, et al (2022) Optimization via rejection-free partial neighbor search. arXiv preprint arXiv:220502083

Efraimidis PS, Spirakis PG (2006) Weighted random sampling with a reservoir. Information processing letters 97(5):181–185

Feng C (2021) MCMC interactive gallery. https://chi-feng.github.io/mcmc-demo/app.html?algorithm=RandomWalkMH&target=donut, Last accessed on 2022-07-05

Geyer CJ (2011) Introduction to Markov chain Monte Carlo. Handbook of Markov chain Monte Carlo 20116022:45

Hastings WK (1970) Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57:97–109

Hitchcock DB (2003) A history of the Metropolis-Hastings algorithm. The American Statistician 57(4):254–257

Kalos MH, Whitlock PA (2009) Monte Carlo methods. John Wiley & Sons

Kochenberger G, Hao JK, Glover F, et al (2014) The unconstrained binary quadratic programming problem: a survey. Journal of combinatorial optimization 28(1):58–81

Kroese DP, Brereton T, Taimre T, et al (2014) Why the Monte Carlo method is so important today. Wiley Interdisciplinary Reviews: Computational Statistics 6(6):386–392

Matsubara S, Takatsu M, Miyazawa T, et al (2020) Digital annealer for high-speed solving of combinatorial optimization problems and its applications. 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC) pp 667–672. https://doi.org/10.1109/ASP-DAC47756.2020.9045100

Metropolis N, Rosenbluth AW, Rosenbluth MN, et al (1953) Equation of state calculations by fast computing machines. The journal of chemical physics 21(6):1087–1092

Meyn SP, Tweedie RL (2012) Markov chains and stochastic stability. Springer Science & Business Media

Rosenthal JS, Dote A, Dabiri K, et al (2021) Jump Markov chains and rejection-free Metropolis algorithms. Computational Statistics pp 1–23

Sheikholeslami A (2021) The power of parallelism in stochastic search for global optimum: Keynote paper. In: ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC), pp 36–42.

Sodan AC, Machina J, Deshmeh A, et al (2010) Parallelism via multithreaded and multicore CPUs. Computer 43(3):24–32

A Unbiased PNS Convergence Theorem

Definition 1 For sampling questions in general state space, we usually have the following elements:

1. a state space $\mathcal{S}$;
2. a σ-finite reference measure μ on $\mathcal{S}$, where μ could be a counting measure for discrete cases, and μ could be a Lebesgue measure for continuous cases;
3. a target density $\pi : \mathcal{S} \to [0, 1]$, where $\int_{x\in\mathcal{S}} \pi(x)\mu(dx) = 1$;
4. a target distribution $\Pi : \mathbf{P}(\mathcal{S}) \to [0, 1]$, where $\Pi(\mathcal{A}) \coloneqq \int_{\mathcal{A}} \pi(x)\mu(dx)$, $\forall \mathcal{A} \subset \mathcal{S}$, and $\mathbf{P}$ means the power set;
5. a proposal density $q(x, y) : \mathcal{S} \times \mathcal{S} \to [0, 1]$, where $\int_{\mathcal{S}} q(x, y)\mu(dy) = 1$, $\forall x, y \in \mathcal{S}$;
6. a proposal distribution $\mathcal{Q}(x, dy) \propto q(x, y)\mu(dy)$;
7. a corresponding neighbor set $\mathcal{N}(x) \coloneqq \{y \in \mathcal{S} \mid q(x, y) > 0\} \subset \mathcal{S}\backslash\{x\}$;
8. the transition probabilities $P(x, dy) = q(x, y) \min\left(1, \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}\right)\mu(dy)$, where $P(x, dy) = q(x, y)\mu(dy)$ if the denominator $\pi(x)q(x, y) = 0$.

Given the above elements, assume irreducibility and aperiodicity, we can generate a Markov chain $\{X_0, X_1, \ldots, X_K\}$ such that the limiting distribution of $\lim_{n\to\infty} X_n$ converges to the stationarity distribution $\pi(x)\mu(dx)$ by Algorithm 1.

Definition 2 Suppose we have a state space $\mathcal{S}$, a reference measure μ, and a target density π, the proposal distribution $\mathcal{Q}$ and the corresponding neighbor set $\mathcal{N}$. Then, a Partial Neighbor Set $\mathcal{N}_i$ means a function $\mathcal{N}_i$ satisfying the following conditions:

1. $\mathcal{N}_i : \mathcal{S} \to \mathbf{P}(\mathcal{S})$, where $\mathcal{S}$ is the state space, and $\mathbf{P}(\mathcal{S})$ is the power set of $\mathcal{S}$;
2. $\mathcal{N}_i(x) \subset \mathcal{N}(x)$, $\forall x \in \mathcal{S}$, and we must pick a finite subset $\mathcal{N}_i(x)$ to ensure a finite for loop in Algorithm 6;
3. $y \in \mathcal{N}_i(x) \iff x \in \mathcal{N}_i(y)$, $\forall x, y \in \mathcal{S}$;

Given a Partial Neighbor Set $\mathcal{N}_i$, the proposal distribution for $\mathcal{N}_i$ is defined to be $\mathcal{Q}_i : \mathcal{S} \times \mathbf{P}(\mathcal{S}) \to \mathbb{R}$, where $\mathcal{Q}_i(x, dy) = \frac{\sum_{r\in\mathcal{N}_i} q(x,r)\delta_r(dy)}{\sum_{z\in\mathcal{N}_i} q(x,z)}$, where δ_r means the point mass at r.

Here, before we prove the convergence theorem of the Unbiased PNS as stated in Algorithm 6, we first prove it for another version of the Unbiased PNS

as stated in Algorithm 11. It is easy to see that the only difference between Algorithm 11 and Algorithm 6 is that we are not using the Rejection-Free technique here, where we calculate all the transition probabilities at once, pick the next jump chain state, and calculate the multiplicity list according to the transition probabilities.

Algorithm 11 Unbiased Partial Neighbor Search without Rejection-Free technique

select $\mathcal{N}_i$ for $i = 0, 1, \ldots, \mathcal{I}-1$ where $\cup_{i=0}^{\mathcal{I}-1}\mathcal{N}_i(X) = \mathcal{N}(X)$
initialize $i = 0$ ▷ start with neighbor set $\mathcal{N}_0$
initialize $L = L_0$ ▷ start with L_0 remaining samples
initialize X_0 ▷ initial the starting state
for k in 1 to K **do**
  random $Y \in \mathcal{N}_i(J_{k-1})$ based on $\mathcal{Q}_i(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \frac{\pi(Y)\mathcal{Q}_i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}_i(X_{k-1}, Y)}$ **then**
    ▷ accept with probability $\min\left\{1, \frac{\pi(Y)\mathcal{Q}_i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}_i(X_{k-1}, Y)}\right\}$
    $X_k = Y$ ▷ accept and move to state Y
  else
    $X_k = X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
  L = L - 1 ▷ a new sample from $\mathcal{N}_i$
  if $L = 0$ **then** ▷ if we don't have enough remaining samples
    $L = L_0$, and $i = i + 1 \mod \mathcal{I}$ ▷ switch to the next $\mathcal{N}_i$
  end if
end for

Proposition 1 *Suppose we have a state space $\mathcal{S}$, a reference measure μ, and a target density π, the proposal distribution $\mathcal{Q}$ and the corresponding neighbor set $\mathcal{N}$. In addition, suppose the Partial Neighbor Set $\{\mathcal{N}_i\}_{i=0}^{\mathcal{I}-1}$ satisfies all the conditions in Definition 1. Then $\pi(x)\mu(dx)$ is the stationary distribution for Algorithm 11 with the partial neighbor set $\mathcal{N}_i$.*

Proof Let $P_i(x, dy)$ be the transition probability for Partial Neighbor Set $\mathcal{N}_i$. Then $\forall y \in \mathcal{N}_i(x)$ where $q(x, y) > 0$, we have

$$\begin{aligned}\pi(x)\mu(dx)P_i(x, dy) &= \pi(x)\mu(dx)q(x, dy)\min\left(1, \frac{\pi(y)q(y, x)\mu(dx)}{\pi(x)q(x, dy)\mu(dy)}\right)\\ &= \min\left(\pi(x)\mu(dx)q(x, dy), \pi(y)q(y, x)\mu(dx)\right)\\ &= \pi(y)\mu(dy)P_i(y, dx)\end{aligned} \tag{16}$$

Thus, by reversibility, $\mathcal{N}_i$ is stationary with $\pi(x)\mu(dx)$. □

Proposition 2 *Suppose we have a state space $\mathcal{S}$, a reference measure μ, a target density π, and a Markov chain $\{X_0, X_1, X_2, \dots\}$ produced by algorithm 11. In addition, suppose $\pi(x)\mu(dx)$ is the stationary distribution is the stationary distribution for Algorithm 11 with all $\{\mathcal{N}_i\}_{i=0}^{\mathcal{I}-1}$, and $\cup_{i=0}^{\mathcal{I}-1}\mathcal{N}_i$ makes the Markov chain irreducible. Moreover, suppose there are rejections for the Markov chain, and thus the Markov chain is aperiodic. Then the Markov chain converges in total variation distance; i.e.:*

$$\lim_{k\to\infty} \sup_{\mathcal{A}\subset\mathcal{S}} \left| P(X_k \in \mathcal{A}) - \int_{\mathcal{A}} \pi(y)\mu(dy) \right| = 0 \tag{17}$$

Proof This follows immediately from Theorem 13.0.1 in Meyn and Tweedie (2012). □

Theorem 3 *Suppose we have a state space $\mathcal{S}$, a reference measure μ, a target density π, a Markov chain $\{X_0, X_1, X_2, \dots\}$ produced by algorithm 11, and a jump chain $\{(J_0, M_0), (J_1, M_1), (J_2, M_2), \dots\}$ produced by algorithm 6. Meanwhile, suppose the proposal distribution $\mathcal{Q}$ and the corresponding neighbor set $\mathcal{N}$ ensure the Markov chain produced by the Metropolis-Hastings algorithm converges to the stationarity $\pi(x)\mu(dx)$. In addition, suppose $\pi(x)\mu(dx)$ is the stationary distribution for all $\{\mathcal{N}_i\}_{i=0}^{\mathcal{I}-1}$, and $\cup_{i=0}^{\mathcal{I}-1}\mathcal{N}_i$ makes both chains irreducible. Moreover, suppose both chains are aperiodic. Then the jump chain has the following properties:*

1. *the transition probability P_i from the Markov chain and the transition probability $\hat{P}_i$ from the jump chain satisfy $\hat{P}_i(x, dy) = \frac{1}{\alpha(x)} P(x, dy)\mathbb{1}(x \neq y)$, and $\hat{P}_i(x, \{x\}) = 0$;*
2. *The conditional distribution of M_k given J_k is equal to the distribution of $1+G$ where G is a geometric random variable with success probability $\alpha(J_k)$ where $\alpha(x) \coloneqq 1 - P_i(x, \{x\})$;*
3. *If the original chain is ϕ-irreducible (see, e.g., Meyn and Tweedie (2012)) for some positive σ-finite measure ϕ on $\mathcal{S}$, then the jump chain is also ϕ-irreducible for the same ϕ.*
4. *If the Markov chain has stationary distribution $\pi(x)\mu(dx)$, then the jump chain has stationary distribution given by $\hat{\pi}(x) = c\alpha(x)\pi(x)\mu(dx)$ where $c^{-1} = \int \alpha(y)\pi(y)\mu(dy)$*
5. *If $h : \mathcal{S} \to \mathbb{R}$ has finite expectation, then with probability 1,*

$$\lim_{K\to\infty} \frac{\sum_{k=1}^{K} M_k h(J_k)}{\sum_{k=1}^{K} M_k} = \lim_{K\to\infty} \frac{\sum_{k=1}^{K} [\frac{h(J_k)}{\alpha(J_k)}]}{\sum_{k=1}^{K} [\frac{1}{\alpha(J_k)}]} = \pi(h) := \int h(x)\pi(x)\mu(dx)$$

Proof The proof is trivial given the Proposition 2 and Theorem 13 from the Rejection-Free paper (Rosenthal et al, 2021). We reviewed Theorem 13 from the Rejection-Free paper in Section 6. □

B How to Sample Proportionally

Given $A_i > 0$, for $i = 1, 2, \ldots, N$, how can we sample Z so that $P(Z = i) = \frac{A_i}{\sum_j A_j}$? We could choose $U \sim \text{Uniform}[0, 1]$, and then set $Z = \min\{i, \sum_{j=1}^{i} A_j > U \times \sum_{j=1}^{N} A_j\}$. However, this involves summing all of the A_j , which is inefficient. If $\sum_{j=1}^{N} A_j = 1$, then we could choose $U \sim \text{Uniform}[0, 1]$ and just set $Z = \min\{i, \sum_{j=1}^{i} A_j > U\}$, which is slightly easier, and can be done by binary searching. However, it still requires summing lots of the A_j, which could still be inefficient. If $\sum_{j=1}^{N} A_j < 1$, then we could choose $U \sim \text{Uniform}[0, 1]$, and then still set$Z = \max\{i, \sum_{j=1}^{i} A_j > U\}$, except if no such i exists then we reject that choice of U and start again. In addition to the previous problems, this could involve lots of rejection if $\sum_{j=1}^{N} A_j$ is much smaller than 1, which is again inefficient. Another option is the following method, based on Efraimidis and Spirakis (2006); see also the n-fold way approach to kinetic Monte Carlo in Bortz et al (1975).

Proposition 4 *Let $A_1, A_2, \ldots, A_N$ be positive numbers, Let $\{R_j\}_{j=1}^{N}$ be i.i.d. $\sim$ Uniform$[0, 1]$, and let $d_j = -\frac{\log(R_j)}{A_j}$ for $j = 1, 2, \ldots, N$. Finally, set $Z = \arg\min_j d_j$. Then $P[Z = i] = \frac{A_i}{\sum_j A_j}$, i.e. Z selects i from $\{1, 2, \ldots, N\}$ with probability proportional to A_i.*

Proof

$$
\begin{aligned}
P[Z = i] &= P[d_j > d_i, \forall j \neq i] \\
&= P[-\frac{\log(R_j)}{A_j} > -\frac{\log(R_i)}{A_i}, \forall j \neq i] \\
&= P[R_j < R_i^{A_j/A_i}, \forall j \neq i] \\
&= \int_0^1 P[R_j < R_i^{A_j/A_i}, \forall j \neq i \mid R_i = x] dx \\
&= \int_0^1 P[R_j < x^{A_j/A_i}, \forall j \neq i] dx \\
&= \int_0^1 \prod_{j \neq i} x^{A_j/A_i} dx \\
&= \int_0^1 x^{\sum_{j \neq i} A_j/A_i} dx \\
&= \frac{x^{[\sum_{j \neq i} A_j/A_i + 1]}}{\sum_{j \neq i} A_j/A_i + 1} \Bigg|_{x=0}^{1} \\
&= \frac{A_i}{\sum_j A_j}
\end{aligned}
\tag{18}
$$

□

Proposition 4 is useful, especially when we apply Rejection-Free and PNS to parallelism hardware. In addition, even when we apply it to a single core implementation, computing arg min is faster than dividing by the sums.