Optimization 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

Simulated Annealing using Metropolis steps at decreasing temperatures is widely used to solve complex combinatorial optimization problems (Kirkpatrick et al, 1983). To improve its efficiency, we can use the Rejection-Free version of the Metropolis algorithm which avoids the inefficiency of rejections by considering all the neighbors at each step (Rosenthal et al, 2021). To avoid the algorithm getting stuck in a local extreme area, we propose an enhanced version of Rejection-Free called Partial Neighbor Search, which only considers random part of the neighbors while applying the Rejection-Free technique. In this paper, we apply these methods to several examples such as quadratic unconstrained binary optimization (QUBO) problems to demonstrate superior performance of the Rejection-Free Partial Neighbor Search algorithm.

Keywords: Simulated Annealing, Rejection-Free, Partail Neighbor Search, QUBO

1 Introduction

Optimization is a cornerstone of many areas, and it plays a key role in finding feasible solutions to real-life problems, from mathematical programming to operations research, economics, management science, business, medicine, life science, and artificial intelligence, etc. (Floudas and Pardalos, 2008). Optimization began with a number of independent topics such as optimum assignment, shortest spanning tree, transportation, and the traveling salesman problem, until the unifying tool of linear and integer programming invented in the 1950's put all of these topics into one framework (Schrijver, 2005). Combinatorial optimization plays an important role in today's research because most of its problems descend directly from practice, and instances of them were, and still are, attacked daily (Schrijver, 2005). Complex combination optimization may need a tremendous amount of time to find a feasible solution. For example, no algorithms has been found for NP-hard problems which can be guaranteed to find the optimal state within a time limit bound by a polynomial in the length of the input (Garey et al, 1974).

Metaheuristics are general algorithmic frameworks, often nature-inspired, designed to solve complex optimization problems (Bianchi et al, 2009) by getting a good enough feasible solution even if not the optimal one. The Simulated Annealing algorithm (Kirkpatrick et al, 1983), based on the Metropolis algorithm (Metropolis et al, 1953) at decreasing temperatures, is a typical method of this kind. However, the Simulated Annealing algorithm may face the inefficiency of rejections. Therefore, we adopt the Rejection-Free algorithm for sampling (Rosenthal et al, 2021) into an optimization version to improve the performance of Simulated Annealing. In addition, Rejection-Free may also face inefficiency when it enters a local extreme area. Thus, we propose another algorithm called Partial Neighbor Search based on the technique of the Rejection-Free algorithm to improve the efficiency further.

We next review the Simulated Annealing algorithm for optimization, as well as the Metropolis algorithm and the Rejection-Free algorithm for sampling. Then, in Section 2, we describe how to use the Rejection-Free algorithm to solve optimization problems. We then point out one problem of the Rejection-Free algorithm which may lead to another kind of inefficiency. After that, in Section 3, we introduce our Partial Neighbor Search (PNS) optimization algorithm, which considers just subsets of neighbor states for possible moves. In Section 4, we apply PNS to a quadratic unconstrained binary optimization (QUBO) problems, and show how it helps us to find better results. We then discuss why this improvement occurs (Section 5), and how its subsets should be chosen (Section 6), as well as its relation to Tabu Search algorithms (Section 7). Finally, we use several additional examples, including the Knapsack problem (Section 8) and the 3R3XOR problem (Section 9) to illustrate the advantages of the PNS algorithm in optimization problems.

1.1 Background on Simulated Annealing for optimization

Simulated annealing, introduced by Krkpatrick et al. in 1982 (Kirkpatrick et al, 1983) based on the Metropolis algorithm (Metropolis et al, 1953), is widely applied to solve combinatorial optimization problems such as approximating the optimal values of functions with many variables (Rutenbar, 1989). This algorithm does not guarantee an optimal solution, but it can give good feasible solutions quickly (Albright, 2007). Simulated Annealing contains the following basic elements (Bertsimas and Tsitsiklis, 1993):

1. A state space $\mathcal{S}$
2. A real-valued target distribution π on $\mathcal{S}$
3. $\forall X \in S$, $\exists$ a proposal distribution $\mathcal{Q}(X, \cdot)$ where $\int_{Y \in \mathcal{S} \backslash \{X\}} \mathcal{Q}(X, Y) = 1$. Usually, the proposal distribution is symmetric, i.e, $\mathcal{Q}(X, Y) = \mathcal{Q}(Y, X)$. We assume symmetricity in this paper as well.
4. $\forall X \in \mathcal{S}$, $\exists\, \mathcal{N}(X) = \{Y \in \mathcal{S} \mid \mathcal{Q}(X, Y) > 0\} \subset \mathcal{S} \backslash \{X\}$, called the neighbors of X
5. A non-increasing function $T : \mathbb{N} \to (0, \infty)$, called the Cooling Schedule. $T(k)$ is called the Temperature at step $k \in \mathbb{N}$
6. An initial State $X_0 \in \mathcal{S}$

With the above elements, the Simulated Annealing algorithm which consists of a discrete time-inhomogeneous Markov Chain $\{X_k\}_{k=0}^{K}$ can be generated by Algorithm 1.

Algorithm 1 Simulated Annealing 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 $U_k < [\frac{\pi(Y)}{\pi(X_{k-1})}]^{1/T(k)}$ **then**
    ▷ accept with probability $\min\left\{1, [\frac{\pi(Y_k)}{\pi(X_{k-1})}]^{1/T(k)}\right\}$
    $X_k = Y$ ▷ accept and move to state Y
  else
    $X_k = X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
end for

Algorithm 1 is designed to converge to states X_k with nearly-maximal values of $\pi(X_k)$, though that is not guaranteed.

1.2 Background on Metropolis algorithm for sampling

The above Simulated Annealing algorithm is based on the Metropolis algorithm (Metropolis et al, 1953). 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 which converges to a given target distribution π on a state space $\mathcal{S}$ (Rosenthal et al, 2021). With these two given elements π and $\mathcal{S}$, we can generate a Markov chain by Algorithm 2.

Algorithm 2 Metropolis 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 $U_k < \frac{\pi(Y)}{\pi(X_{k-1})}$ **then**
    ▷ accept with probability $\min\left\{1, \left[\frac{\pi(Y)}{\pi(X_{k-1})}\right]^{1/T(k)}\right\}$
    $X_k = Y$ ▷ accept and move to state Y
  else
    $X_k = X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
end for

Algorithm 2 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 $h : \mathcal{S} \to \mathbb{R}$ with respect to π can be estimated by $\frac{1}{M}\sum_{i=1}^{M} h(X_i)$ for sufficiently large run length M. The lack of the Cooling Schedule is the only difference between the Simulated Annealing algorithm and the Metropolis algorithm. Thus, like the Simulated Annealing algorithm, the Metropolis algorithm may also face the inefficiencies from the rejections (Rosenthal et al, 2021).

1.3 Background on Rejection-Free algorithm for sampling

Rejections in the Metropolis and Simulated Annealing algorithms could be a problem. In Algorithm 1, if $U_k \geq [\frac{\pi(Y)}{\pi(X_k)}]^{1/T(k)}$, 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 they are normally considered to be a necessary evil of the Metropolis and Simulated Annealing algorithms. However, we can compute all potential acceptance probabilities at once to allow for the possibility of skipping these rejection steps (Rosenthal et al, 2021). By taking out the inefficiencies of rejections in the Metropolis algorithm, such Rejection-Free algorithm can lead to significant speedup.

Before introducing Rejection-Free, we need to introduce the Jump Chain first. 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 the Multiplicity List $\{M_k\}$ is used to count the number of times the original chain remains at the same state.

For example, if the original chain is

$$\{X_k\} = \{a, b, b, b, a, a, c, c, c, c, d, d, a, \dots\}, \tag{1}$$

then the jump chain would be

$$\{J_k\} = \{a, b, a, c, d, a, \dots\}, \tag{2}$$

with the corresponding multiplicity list being

$$\{M_k\} = \{1, 3, 2, 4, 2, 1, \dots\}. \tag{3}$$

The jump chain $\{J_k, M_k\}$ itself is a Markov chain, with transition probabilities $\hat{P}(y \mid x)$ specified by $\hat{P}(x \mid x) = 0$, and for $y \neq x$, $\hat{P}(y \mid x) := P[J_{k+1} = y \mid J_k = x] = \frac{P(y|x)}{\sum_{z \neq x} P(z|x)}$. 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).

Given the above properties for the Jump chain, the Rejection-Free algorithm is a sampling method described by Algorithm 3.

Algorithm 3 Rejection-Free algorithm for Sampling

initialize J_0
for k in 1 to K **do**
  $p \leftarrow 0$ ▷ p is used to record $\sum_{z \neq x} P(z \mid x)$
  for Y in $\mathcal{N}(J_{k-1})$ **do**
    calculate $q(Y) = \mathcal{Q}(Y, J_{k-1}) \min\{1, \frac{\pi(y)}{\pi(J_{k-1})}\}$
    ▷ the transition prob. from J_{k-1} to Y
    $p \leftarrow p + q(Y)$ ▷ sum up $P(z \mid x)$ to get $\sum_{z \neq x} P(z \mid x)$
  end for
  choose $J_k \in N(J_{k-1})$ such that $\hat{P}(J_k = Y \mid J_{k-1}) \propto q(Y)$
  ▷ choose the next Jump Chain State
  calculate $M_{k-1} = 1 + G$ where $G \sim \text{Geom}(p)$
  ▷ multiplicity list for current state
end for

Algorithm 3 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. This can lead to great speedup in examples where rejections happen frequently for the Metropolis algorithm (Rosenthal et al, 2021).

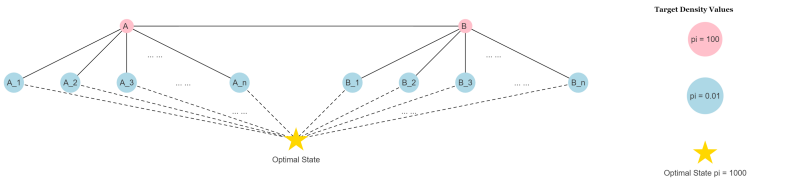

Fig. 1 Illustration of the local maximum area in an optimization problem where both Simulated Annealing and Rejection-Free may get stuck.

2 Rejection-Free algorithm for optimization

In addition to sampling, the above Rejection-Free algorithm can also be applied in optimization problems, as follows. Given a set $\mathcal{S}$ and a real-valued target distribution π on the set $\mathcal{S}$, we can use the Rejection-Free algorithm to find $X \in \mathcal{S}$ to maximize $\pi(X)$ by Algorithm 4. This algorithm is only different from the Algorithm 4 by getting rid of the multiplicity list $\{M_k\}$.

Algorithm 4 Rejection-Free algorithm for Optimization

initialize J_0
for k in 1 to K **do**
  for $Y \in N(J_{k-1})$ **do**
    calculate $q(Y) = \mathcal{Q}(Y, J_{k-1}) \min\{1, [\frac{\pi(Y)}{\pi(J_k)}]^{\frac{1}{T(k)}}\}$
    ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in N(J_{k-1})$ such that $\hat{P}(J_k = Y \mid J_{k-1}) \propto q(Y)$
    ▷ choose the next Jump Chain State
end for

Algorithm 4 is again designed to converge to states X_k with nearly-maximal values of $\pi(X_k)$, with greater efficiency by avoiding rejections, though that is again not guaranteed.

Although the Rejection-Free algorithm for optimization can help reduce the inefficiency of rejections, local maximum areas of π can still be a problem. For example, we want to find $X \in S$ which maximizes $\pi(X)$ from a state space starting at state A in Figure 1. If we use simple Simulated Annealing algorithm for $T = 1$ constantly, then we will have many rejections. Note that, $\pi(A_1), \pi(A_2), \ldots, \pi(A_n) = 0.01$ while $\pi(A) = \pi(B) = 100$. The probability of escaping from A is $\frac{1}{n+1} + \frac{n}{n+1} \times \frac{1}{10000}$, where $\frac{1}{n+1}$ is the probability of moving to state B and $\frac{n}{n+1} \times \frac{1}{10000}$ is the probability of moving to other states $A_1, A_2, \ldots, A_n$. Cooling Schedules can help to some extent at the beginning of the Simulated Annealing algorithm since we are supposed to have a large temperature T at the beginning. As we move on in the Simulated Annealing algorithm, we will be more and more likely to be trapped by local maximum areas like this. Rejection-Free algorithm for optimization can produce some

speedup in this case, but the Rejection-Free chain will still be stuck by the local maximum area $\{\pi(A), \pi(B)\}$. This chain will be switching between state A and B for a really long time as $\hat{P}(J_1 = B \mid J_0 = A) = \frac{\min\{1, \frac{\pi(B)}{\pi(A)}\}}{\sum_{z \neq A} \min\{1, \frac{\pi(z)}{\pi(A)}\}} = \frac{1}{1+0.0001 \times n} \approx 1$ and $\hat{P}(J_2 = A \mid J_1 = B) \approx 1$ for small n. To help our Markov chain escape from those local maximums in optimization, we propose another method called Partial Neighbor Search based on the Rejection-Free algorithm.

3 Proposed Search Algorithm: Partial Neighbor Search

Partial Neighbor Search (PNS) is an algorithm based on the Rejection-Free, also designed as a Markov chain used for optimization as described in Algorithm 5.

Algorithm 5 Partial Neighbor Search

initialize J_0
for k in 1 to K **do**
  pick $\mathcal{N}_k(J_{k-1}) \subset \mathcal{N}(J_{k-1})$ $(\star)$
  for $Y \in \mathcal{N}_k(J_{k-1})$ **do** ▷ Only neighbors in $\mathcal{N}_k$ will be considered
    calculate $q(Y) = \mathcal{Q}(Y, J_{k-1}) \min\{1, [\frac{\pi(Y)}{\pi(J_k)}]^{\frac{1}{T(k)}}\}$
    ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in \mathcal{N}_k(J_{k-1})$ such that $\hat{P}(J_k = Y \mid J_{k-1}) \propto q(Y)$
  ▷ choose the next Jump Chain State
end for

Algorithm 5 is again designed to converge to states X_k with nearly-maximal values of $\pi(X_k)$, with greater efficiency by avoiding both rejections and traps in local maximum areas.

The $(\star)$ step in Algorithm 5 could randomly pick 50% of the elements in $N(J_{k-1})$. We will explore many other choices for the $(\star)$ step to find the best method for this step in later sections

The motivation for this algorithm is simple: if we can force the algorithm to avoid some of the neighbors randomly, then we have a better chance of escaping from the local maximum area. For example, if we only consider half of the neighbors at state A, then we may ignore state B with probability 50%, then we have at least 50% probability to choose a state from $\{A_1, A_2, \ldots, A_n\}$ as our next state in the PNS chain. Then we are more likely to escape from the local maximum area in this case.

4 Application to QUBO question

The Quadratic unconstrained binary optimization (QUBO) has been rising in importance in the field of combinatorial optimization because of its wide range of applications in finance and economics to machine learning (Kochenberger et al, 2014). QUBO is also a well known NP-hard problem (Glover et al, 2018). Thus, using the Simulated Annealing to find the optimal or workable solutions is common. We now consider applying our PNS algorithm to this problem. (Additional applications are in Sections 8 and 9 below.)

For a given N by N matrix Q (usually upper triangular), the QUBO question aims to find

$$\arg\max x^T Qx, \text{ where } x \in \{0,1\}^N \tag{4}$$

(Sometimes arg min is used in place of arg max, which is equivalent to taking the negative of Q, so for simplicity we focus on the arg max version here.)

To run our algorithm, we used a uniform proposal distributions among all neighbors where the neighbors are defined to be binary vectors with Hamming distance 1. That is, $\mathcal{Q}(X,Y) = \frac{1}{N}$ for $\forall Y \in \mathcal{N}(X)$, where $Y \in \mathcal{N}(X) \iff |X-Y| = \sum_{i=1}^{N} |X_i - Y_i| = 1$, $\forall X, Y \in \{0,1\}^N$. We randomly choose half of the neighbors at each step of PNS, which means we only consider a random subset $\mathcal{N}_K(x) \subset \mathcal{N}(x)$ whose cardinality is $|\mathcal{N}_k(X)| = \frac{1}{2}|\mathcal{N}(X)| = \frac{1}{2}N$ for $\forall X \in \{0,1\}^N$. In addition, the target distribution $\pi(x) = \exp\{x^T Qx\}$, since maximizing $x^T Qx$ is the same as maximizing $\exp\{x^T Qx\}$. Furthermore, $T(k)$ represents the temperature at step k for the cooling schedule here.

We compare the following three algorithms in 1000 simulation runs here: Simulated Annealing, Rejection-Free for Optimization, and PNS. We randomly generate a 200 by 200 upper triangular as the QUBO matrix Q. The non-zero elements from the 200 by 200 upper triangular matrix Q were generated randomly by $Q_{i,j} \sim \text{Normal}(0, 100^2)$, $\forall i \leq j$.

The result for the simulation is shown in Figure 2. Here, we used a violin plot to summarize the results. The violin plot uses the information available from local density estimates and the basic summary statistics to provide a useful tool for data analysis and exploration (Hintze and Nelson, 1998). The violin plot here combines two density traces on both sides and three quantile lines (25%, 50%, and 75%) to reveal the data structure. In addition, we added a long segment of the bottom layer as the mean for the values. We also added a short segment on the y-axis to help compare the mean values.

From Figure 2, we can see that the PNS is always the best in all four different cooling schedules. Note that, the number of iterations used for Simulated Annealing is 200,000 for Simulated Annealing while they are 1000 for both Rejection-Free and PNS. This is because we need to consider 200 neighbors at each iteration in Rejection-Free for optimization, while we only need to consider 1 neighbor for each iteration in Simulated Annealing. In fact, if we proceed all three algorithms on a single-core machine, the run time of a single simulation run for simulated annealing is about 20 seconds; the run time

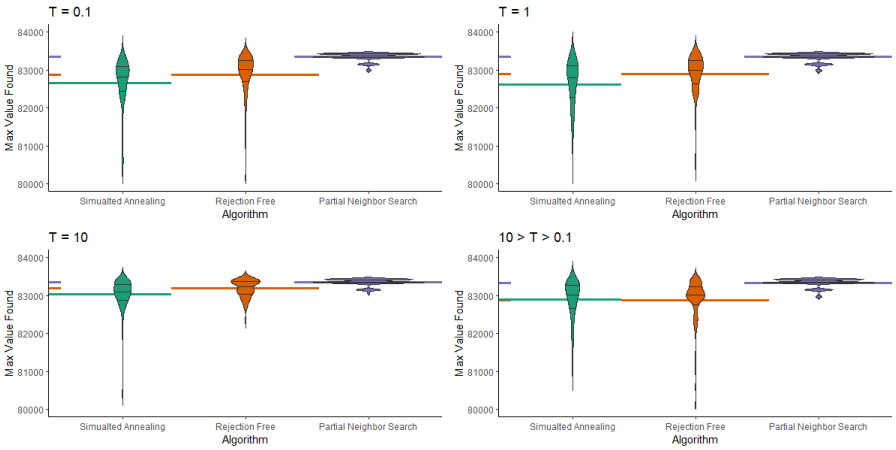

Fig. 2 Comparison of Simulated Annealing, Rejection-Free, and PNS in terms of the highest (log) target distribution value $\log \pi(x) = x^T Qx$ being found, for a random upper triangular QUBO matrix Q where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules: $T(k) = 0.1$, 1 and 10 constantly, and $T(k)$ being geometric from 10 to 0.1. The number of iterations for Simulated Annealing is 200,000, and the numbers of iterations for for Rejection-Free and PNS are 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

for Rejection-Free is about 10 seconds; the run time for PNS is only 5 seconds. In addition, parallelism in computer hardware can increase the speed of both Rejection-Free and PNS by distributing 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 PNS (Sheikholeslami, 2021).

In the above example, the improvement in efficiency of Rejection-Free is not hard to understand. The performance of PNS is somehow counter-intuitive. Comparing to Rejection-Free, why would we get a better result by considering less neighbors at each step? To illustrate how PNS works in more details, we can have a closer look at the Markov chains generated in the above example.

5 Understanding the improvement of Partial Neighbor Search

In this section, we found a local maximum area for the target distribution π purposefully in the previous QUBO example in section 4 by looking at the final results from the simulation runs from the previous section. Many Rejection-Free chains stopped at this local maximum area after 1000 iterations. For this local maximum area, the energy is around 82600, and this local maximum area contains three states whose cost function values are much larger than

all of their other neighbors. Thus, this local maximum can trap the regular Rejection-Free chain for a long time just like the cases we mentioned in Figure 1. We can plot the Markov chains by PNS with the cost function values for all the neighbors by Rejection-Free or random subset of neighbors by PNS in the form of boxplots. The boxplot of the first 30 steps from the first simulation in PNS is shown at the first plot in Figure 3

From the first plot in Figure 3, most of the cost function values within the boxplot are not useful at all, since they are too small to be picked by the algorithm. Therefore, we only need to consider the important neighbors which are likely to be chosen. Firstly, for each state J_k in the Markov Chain, we find the max value among all the transition probabilities, and we define the important neighbors to be those neighbors whose transition probability is larger than $\exp\{-10\}$ times the highest transition probability among all neighbors. That is, for each J_k from the chain, we find $q(Y_0) = \max\{q(Y) \mid Y \in \mathcal{N}(J_k)\}$, and then we define $\{Y \mid Y \in \mathcal{N}(J_k), q(Y) > \exp\{-10\} \times q(Y_0)\}$ to be important neighbors for J_k. This time, we only have several important neighbors at each step. Thus, we used points instead of boxplots to show the important neighbors. The result from both Rejection-Free and PNS are also shown in Figure 3.

From the second plot in Figure 3, the red dots represent the important neighbors, and the pink line means the Rejection-Free chain. We can see that this local maximum area of three states can easily trap the Rejection-Free chains because their cost function values are much higher than all other neighbors. Thus, the important neighbors for any of these three states are only the rest two of them, and the Rejection-Free chain will be switching between these three states for a really long time. At the same time, the blue dots in the second plot represent the important neighbors if we start to do PNS from that state. Although we did not apply PNS in the second plot, we still put the random subset for PNS there as a comparison. From the blue dots in the second plot, we can say that, if we perform PNS, then the Markov chain is more likely to escape from this local maximum area faster since some groups of the blue dots don't contain any of the three states with high cost function values.

On the other hand, the third plot in Figure 3 shows that the PNS chain (blue line) escapes from this local maximum area within 5 steps. Again, the blue dots represent the important neighbor from PNS and the red dots represent the important neighbor if we start to perform Rejection-Free from that step. For each step of PNS within the local maximum area of three states, the Markov Chain has the probability of 25% probability to include neither of the remaining two neighbors from the three states. Thus, PNS helped the Markov chain to escape from this local maximum area. In addition, at the middle part of the PNS chain, when the cost function value of the PNS chain is increasing, we usually have more than 1 important neighbors. Let's say if we have 3 important neighbors, then we only have 12.5% for considering none of them by PNS.

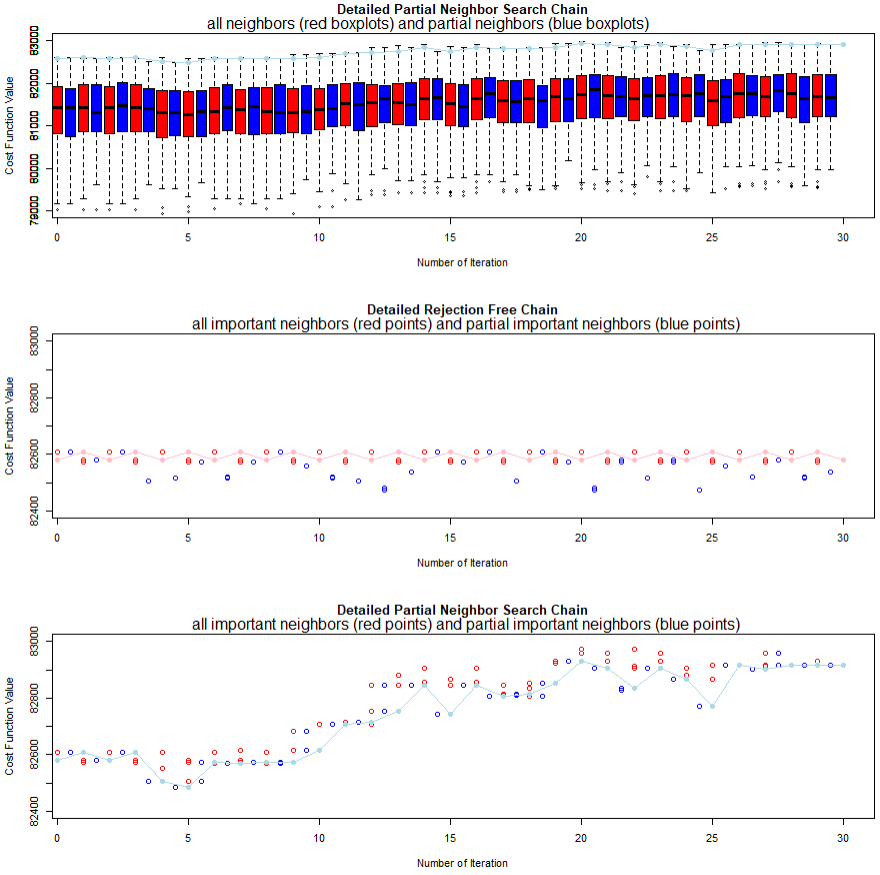

Fig. 3 The detailed Markov Chains from Rejection-Free (the pink chain in the second plot) and PNS (the light blue chain in the first and the third plot). The red box-plots in the first plot represents the cost function values for all neighbors, and the blue box-plots represent the partial neighbors. Most of these values are useless because they are too small to be picked by the Markov chain. The second and the third plot only plot the important neighbors, which is defined as whose transition probability is larger than $\exp\{-10\}$ times the highest transition probability among all neighbors. Here, red points represents all important neighbors and blue points means important neighbors of a random subset of all neighbors used for PNS. The Rejection-Free Chain is switching between three local maximum states all the time while the PNS chain escape from the local maximum area after 5 iterations.

Thus, the PNS is better than Rejection-Free because the PNS performs much better than the Rejection-Free algorithm when the Markov chain is being trapped by the local maximum areas. On the other hand, PNS is not so much worse than the Rejection-Free when the Markov chain is increasing with respect to the cost function value.

In this section, we use 50% random partial neighbors for each step. In fact, we have many other choices. We will consider and compare these choices in the next section

6 Optimal subset choice for Partial Neighbor Search

We need to formally define the way of choosing partial neighbors first. Before we start our Markov Chain, we need to define a pair $\{\mathcal{Q},\mathcal{N}\}$ where $\mathcal{Q}(X,\cdot)$ means the proposal distribution, and $\mathcal{N}(X) := \{Y \in S \mid \mathcal{Q}(X,Y) > 0\}$ is the corresponding neighbor sets. A PNS kernel for $\{\mathcal{Q},\mathcal{N}\}$ means any pair $\{\mathcal{Q}_k,\mathcal{N}_k\}$ satisfies the following conditions:

1. $\mathcal{Q}_k : S \times S \to \mathbb{R}$ is a proposal distribution
2. $\mathcal{N}_k(X) = \{Y \in S \mid \mathcal{Q}_k(X,Y) > 0\}$ is the corresponding neighbor set
3. $\mathcal{N}_k(X) \subset \mathcal{N}(X)$
4. $\mathcal{Q}_k(X,Y) \propto \mathcal{Q}(X,Y)$ for $Y \in \mathcal{N}_k(X)$, and $\mathcal{Q}_k(X,Y) = 0$ otherwise

Note that the footnote k on $\mathcal{Q}$ and $\mathcal{N}$ means the proposal distribution for step k from algorithm 5. Usually, we want to pick $\mathcal{Q}_k$ such that $|\mathcal{N}_k(X)| < |\mathcal{N}(X)|$ to perform proper PNS.

Here, we compare the four different ways to choose the proposal distribution $\{\mathcal{Q}_k,\mathcal{N}_k\}$ for PNS in the $(\star)$ step in Algorithm 5:

- Method A (random subset every step): The proposal kernel $\mathcal{Q}_k$ and neighbor set $\mathcal{N}_k$ are randomized for every step, where $|\mathcal{N}_k(X)| = \frac{1}{2} \times |\mathcal{N}(X)|$, and $\mathcal{Q}_k(X,Y) \propto \mathcal{Q}(X,Y)$ for $Y \in \mathcal{N}_k(X)$, and $\mathcal{Q}_k(X,Y) = 0$ otherwise.
- Method B (random subset every 10 steps): The proposal kernel $\mathcal{Q}_k$ and neighbor set $\mathcal{N}_k$ are randomized for once 10 steps, where $|\mathcal{N}_k(X)| = \frac{1}{2} \times |\mathcal{N}(X)|$, and $\mathcal{Q}_k(X,Y) \propto \mathcal{Q}(X,Y)$ for $Y \in \mathcal{N}_k(X)$, and $\mathcal{Q}_k(X,Y) = 0$ otherwise. That is, $\{\mathcal{Q}_{10\times k+1},\mathcal{N}_{10\times k+1}\} = \{\mathcal{Q}_{10\times k+2},\mathcal{N}_{10\times k+2}\} = \ldots = \{\mathcal{Q}_{10\times k+10},\mathcal{N}_{10\times k+10}\}$ for $\forall k \in \mathbb{N}$.
- Method C (systematic subset every step): Before we start our Markov Chain, based on the pair $\{\mathcal{Q},\mathcal{N}\}$, we define two pairs of PNS kernel and neighbor sets $\{\mathcal{Q}_1,\mathcal{N}_1\}$ and $\{\mathcal{Q}_2,\mathcal{N}_2\}$, where $|\mathcal{N}_1(X)| = |\mathcal{N}_2(X)| = \frac{1}{2} \times |\mathcal{N}(X)|$, $\mathcal{N}_1(X) \cap \mathcal{N}_2(X) = \emptyset$, and $\mathcal{Q}_k(X,Y) \propto \mathcal{Q}(X,Y)$ for $Y \in \mathcal{N}_k(X)$, and $\mathcal{Q}_k(X,Y) = 0$ otherwise.
  For step k of the Markov chain, we only randomly generate $r_k \in \{1,2\}$, and apply $\{\mathcal{Q}_r,\mathcal{N}_r\}$ for step k.
- Method D (systematic subset every 10 steps): Before we start our Markov Chain, we define two sets of the proposal function and neighbor function $\{\mathcal{Q}_1,\mathcal{N}_1\}$ and $\{\mathcal{Q}_2,\mathcal{N}_2\}$, where $|\mathcal{N}_1(X)| = |\mathcal{N}_2(X)| = \frac{1}{2} \times |\mathcal{N}(X)|$, $\mathcal{N}_1(X) \cap \mathcal{N}_2(X) = \emptyset$, and $\mathcal{Q}_k(X,Y) \propto \mathcal{Q}(X,Y)$ for $Y \in \mathcal{N}_k(X)$, and $\mathcal{Q}_k(X,Y) = 0$ otherwise.
  For once 10 steps of the Markov chain, we only randomly generate $r_k \in \{1,2\}$, and apply $\{\mathcal{Q}_r,\mathcal{N}_r\}$. That is $r_{10\times k+1} = r_{10\times k+2} = \ldots = r_{10\times k+10}$ for $\forall k \in \mathbb{N}$.

Again, we still use the 200×200 QUBO example here. The settings for the simulation are the same as the previous section. For method C and D, the two pairs $\{\mathcal{Q}_1,\mathcal{N}_1\}$ and $\{\mathcal{Q}_2,\mathcal{N}_2\}$ are uniform proposal distribution for

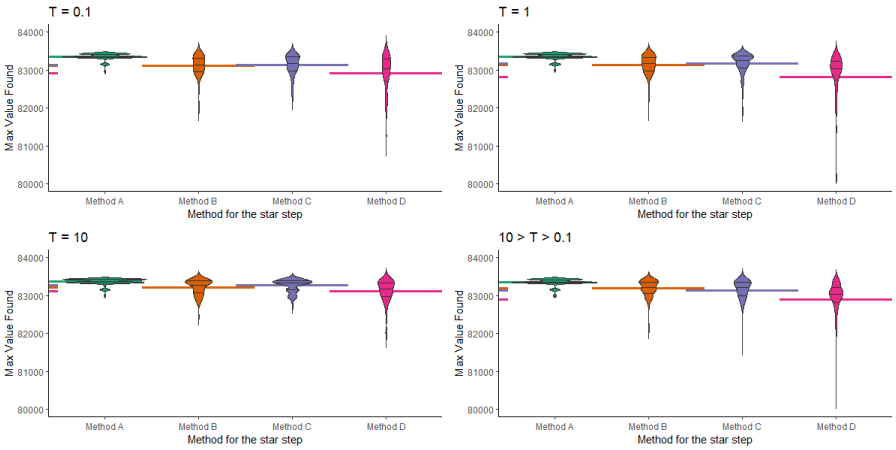

Fig. 4 Comparison of different methods to choose the subsets for PNS, in terms of the highest (log) target density value $\log \pi(x) = x^T Q x$ found. Method A: random subset every step; method B: random subset every 10 steps; method C: systematic subset every step; method D: systematic subset every 10 steps. Random upper triangular QUBO matrix where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules where T = 0.1, 1 and 10 for all n, and T being geometric from 10 to 0.1. The number of iterations for all method are 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

only flipping the first 100 entries in x, and uniform proposal distribution for only flipping the last 100 entries in x. The result for the simulation is shown in Figure 4. From this figure, we can see that random subset at every step (Method A) performs the best in all four cooling schedules. Therefore, we will keep using Method A in all later parts.

In addition, we simply used $|\mathcal{N}_k(X)|\frac{1}{2} \times |\mathcal{N}(X)|$ in previous simulations. Now we compare the following cardinality for the random subset in the $(\star)$ step in Algorithm 5: $|\mathcal{N}(X)| \times \{1, \frac{3}{4}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}\}$ by the same simulation settings as before. From Figure 5, we can see that $\frac{1}{3}, \frac{1}{4}, \frac{1}{5}$ are overall the best among all the choices. Thus, we can conclude that PNS with random subset about 25% of the neighbors for each step performs the best in this simulation for the QUBO question here.

Therefore, we conclude that our best method to do optimization for the 200×200 QUBO question as Algorithm 6.

7 Comparison with Tabu Rejection-Free algorithm

Tabu search (Glover, 1989) (Glover, 1990) is also a methodology in optimization that guides a local heuristic search procedure to explore the solution space beyond local optimality. The idea of Tabu search is to prohibit access to certain previously-visited solutions. Tabu search is the most intuitive method to help the Markov Chain escape from local maximum areas as what we had in

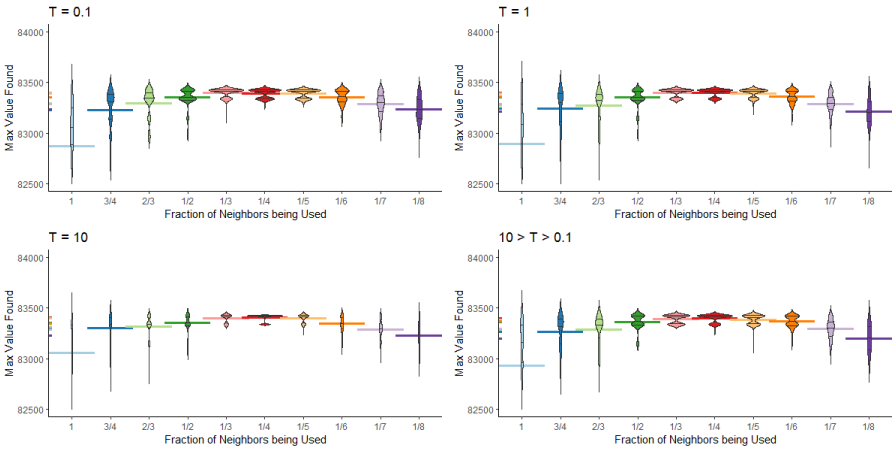

Fig. 5 Comparison of different sizes of the random subsets for PNS, in terms of the highest (log) target density value $\log \pi(x) = x^T Q x$ being found. Subset sizes are $N \times \{1, \frac{3}{4}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}\}$. Random upper triangular QUBO matrix where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules: where T = 0.1, 1 and 10 for all n, and T being geometric from 10 to 0.1. The number of iterations for all method are 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

Algorithm 6 Partial Neighbor Search for the 200 by 200 QUBO question

initialize J_0
for k in 1 to K **do**
  randomly pick $N_k(J_{k-1}) \subset N(J_{k-1})$ where $|N_k(J_{k-1})| = 50$
    ▷ Only 50 out of the 200 neighbors will be considered
  for $Y \in N_k(J_{k-1})$ **do**
    calculate $q(Y) = \min\{1, [\frac{\exp(Y^T Q Y)}{\exp(J_{k-1}^T Q J_{k-1})}]^{\frac{1}{T(k)}}\}$
      ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in N_0$ such that $\hat{P}(J_k = Y \mid J_{k-1}) \propto q(Y)$
    ▷ choose the next Jump Chain State
end for

Figure 1. After we move from state A to state B, then we must choose our next state among $\{B_1, B_2, \ldots, B_N\}$. We can combine our Rejection-Free algorithm for optimization with Tabu search and then compare this new method to the PNS by the QUBO question. Here, we can modify the Tabu search a little bit to make it more friendly to the context here. First of all, we do not need to record all visited states, since we are extremely unlikely to re-visit a state after a certain number of steps. Thus, we only need to record the last several steps and prohibit our Markov chain from visiting them again. The new algorithm is formulated as Algorithm 7.

Algorithm 7 L steps Simplified Tabu Rejection-Free for optimization

initialize J_0
for k in 1 to K **do**
  for $Y \in N(J_{k-1}) \backslash \{J_{k-2}, \ldots, J_{k-L-1}\}$ **do**
    ▷ Remove states from the last L steps
    $q(Y) = \min\{1, [\frac{\exp(Y^T QY)}{\exp(J_k^T QJ_k)}]^{\frac{1}{T(k)}}\}$
    ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in N(J_{k-1})$ such that $\hat{P}(J_k = Y \mid J_{k-1}) \propto q(Y)$
    ▷ choose the next Jump Chain State
end for

Here, we compare PNS with L-step Simplified Tabu Rejection-Free for $L = 1, 2, 3, \ldots, 9$. Again, we randomly generate a 200 by 200 upper triangular QUBO matrix. The non-zero elements from the 200 by 200 upper triangular matrix Q were generated randomly with $Q_{i,j} \sim N(0, 100^2)$ for $i < j$. Note that we need to consider about 200 neighbors at each step for both Rejection-Free and Simplified Tabu Rejection-Free, while we only need to consider 50 neighbors at each iteration for PNS. If we proceed all the algorithms on a single core, then the run times for both Rejection-Free and Tabu Rejection-Free are about 4 times larger than the run time for PNS with the same number of iterations. Therefore, we can compare the PNS with $4 \times 100 = 400$ iterations with the other methods to get a fair comparison for the program on a single core. Note that, we are using this many number of steps here because 400 steps is enough for PNS to find a good enough answer. The result for the simulation is shown in Figure 6. From this plot, we can see that PNS performs much better than either Rejection-Free or Simplified Tabu Rejection-Free.

8 Application to Knapsack problem

The Knapsack problem is another well known NP-hard problem in optimization (Salkin and De Kluyver, 1975). We consider the simplest 0-1 Knapsack problem here. Given a knapsack of max capacity W and N items with corresponding values $\{v_i\}_{i=1}^N$ and weights $\{w_i\}_{i=1}^N$, we want to find finite number of items among all N items which can maximize the total value while not exceeding the max capacity of the knapsack. That is, for given $W > 0$, $\{v_i\}_{i=1}^N > 0$ and $\{w_i\}_{i=1}^N > 0$, find a sequence of N binary variable $\{X_i\}_{i=1}^N \in \{0, 1\}$ to maximize

$$\sum_{i=1}^{N} v_i X_i \quad \text{with respect to} \quad \sum_{i=1}^{N} w_i X_i \leq W \tag{5}$$

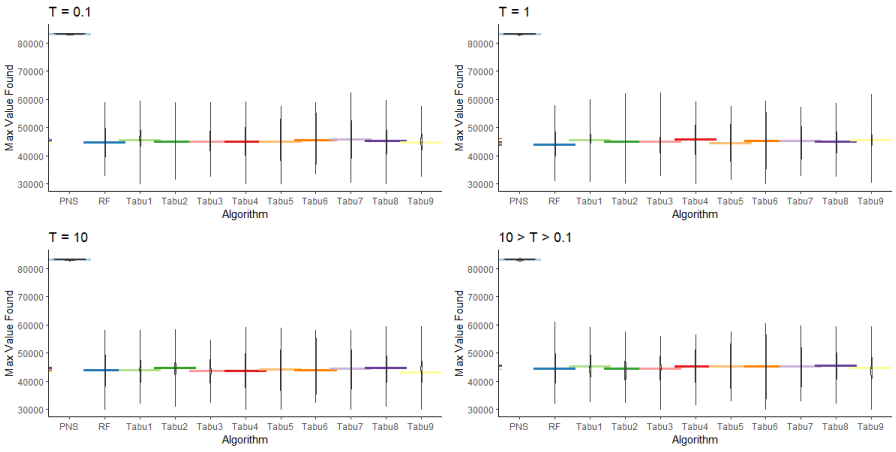

Fig. 6 Comparison of PNS, Rejection-Free and 1-Step to 9-steps Simplified Tabu Rejection-Free, in terms of the highest (log) target density value $\log \pi(x) = x^T Q x$ found. Random upper triangular QUBO matrix where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules where $T = 0.1$, 1 and 10 constantly, and T being geometric from 10 to 0.1. The number of iterations for PNS is 400, and the number of iterations for all other methods is 100. The colored segments represent the mean values.

Since the Knapsack problem is NP-hard, we can use Simulated Annealing algorithm to find a solution to this problem. For this simulation, we set $W = 100,000$. We randomly generate $N = 1000$ items where the values and weights are random by $w_i, v_i \sim \text{Poisson}(1000)$. The mean and the variance for Poisson(1000) are both 1000. Suppose we want to find a binary vector $X = (X_1, X_2, \ldots, X_N)^T$ of dimension N to maximize $v^T X$ with respect to $w^T X \leq W$.

Again, we used a uniform proposal distributions among all neighbors where the neighbors are defined to be binary vectors with Hamming distance 1. That is, $\mathcal{Q}(X, Y) = \frac{1}{N}$ for $\forall Y \in \mathcal{N}(X)$, where $Y \in \mathcal{N}(X) \iff |X - Y| = \sum_{i=1}^{N} |X_i - Y_i| = 1, \forall X, Y \in \{0,1\}^N$. We randomly choose half of the neighbors at each step for PNS. That is, $|\mathcal{N}_k(X)| = \frac{1}{2}|\mathcal{N}(X)| = 500$ for $\forall X \in \{0,1\}^N$. Moreover, the target density $\pi(X) = \mathbb{1}(w^T X \leq W) \times v^T X$. In addition, $T(k)$ represents the temperature at step k for the cooling schedule here.

Again, we compare Simulated Annealing, Rejection-Free with PNS here. The result is shown in Figure 7. From the plot, we can see that Rejection-Free for optimization and PNS algorithm are better than regular Simulated Annealing algorithm in all four different cooling schedules. Again,for the simulation shown in Figure 7, the number of iterations used for three methods are different to have a fair comparison between these three methods. We set the number of iterations for Simulated Annealing algorithm to be $1000,000$, and the number of iterations for Rejection-Free and PNS are set to be 1000. This is because we need to consider 1000 neighbors at each iteration for Rejection-Free for optimization, while we only need to consider 1 neighbor for each iteration in Simulated Annealing.

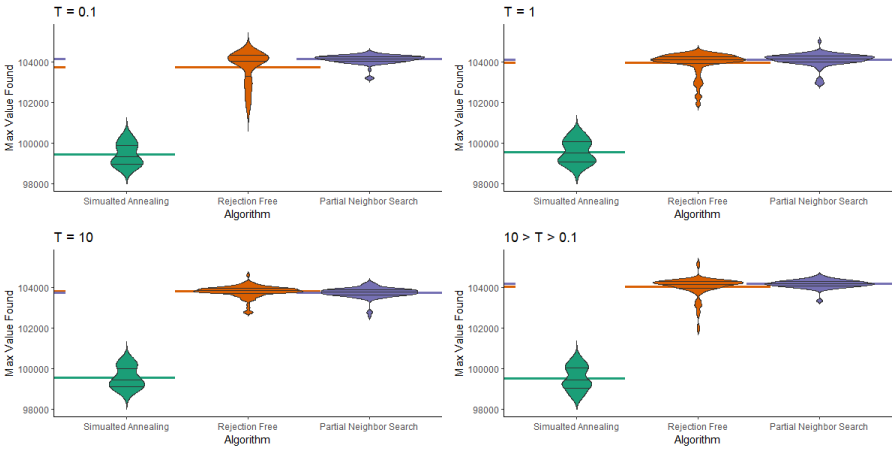

Fig. 7 Comparison of Simulated Annealing, Rejection-Free, and PNS in terms of the highest target density values found in Knapsack Problem with $W = 100,000$, $N = 1000$, $w_i, v_i \sim$ Poisson(1000). Four different cooling schedules: $T = 0.1$, 1 and 10 constantly, and T being geometric from 10 to 0.1. The number of iterations for Simulated Annealing is 1,000,000 while the number for Rejection-Free and PNS are 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

This results shows that PNSis not always that much better than Rejection-Free when the number of iterations are the same. In some cases, where there are not too many local extreme areas, Rejection-Free can have excellent performance as well. On the other hand, if we run the above simulation on a single core, PNS will only take about half of the time used by Rejection-Free.

In addition, Rejection-Free is not always better than simple Simulated Annealing. For example, if $\pi(X) \equiv 1$ for all $X \in S$, then the will be no rejections at all. The Simulated Annealing will move to a new state by computing a single probability, while the Rejection-Free will do the same but compute the probabilities for all neighbors. However, when the dimension of the problem is large, or the target density is sharply peaked, then the PNS will perform much better than Rejection-Free, and Rejection-Free will perform much better Simulated Annealing.

9 Application to 3R3XOR problem

The 3R3XOR problem is a methodology for generating benchmark problem sets for Ising machines devices which is designed to solve discrete optimization problems cast as Ising models introduced by Hen (2019). The Ising model, name after Ernst Ising, is concerned with the physics of magnetic-driven phase transitions (Cipra, 1987). The Ising model is defined on a lattice, where a spin s_i with a value of either -1 or 1 is located on each lattice site (Block and Preis, 2012). Optimization question for the Ising model has been widely applied to many scientific problems such as neuroscience (Hopfield, 1982) and

environmental science (Ma et al, 2014). Thus, algorithms, even special-purpose programmable devices, designed to solve discrete optimization problems cast as Ising models is popular (Hen, 2019), and our PNS algorithm is one of them.

However, the non-planar Ising model is proved to be NP-complete in 2000 (Cipra, 2000). That is, we are unable to find an optimal state from an Ising model in polynomial time. Then, it is hard for us to compare the performance of the algorithms, usually heuristic solvers', by the time used to find the optimal state from a random Ising model. On the other hand, Hen (2019) introduced a tool for benchmarking Ising machines in 2019. In his approach, linear systems of equations are cast as Ising cost functions. The linear systems can be solved easily, while the corresponding Ising model exhibits the features of NP-hardness (Hen, 2019). This way, we can construct some special Ising models whose has a unique optimal state with known optimal value. Then we can use these special Ising models to compare the heuristic solvers' runtimes of finding the optimal state.

In this section, we focus on the construction of a simplified version of 3-body Ising with N spins from a binary linear system of N equations, where the simplified version is defined as follows:

$$H(\{s_j\}) = \sum_{a<b<c} \mathbf{M}_{a,b,c} s_a s_b s_c, \tag{6}$$

where $s_i \in \{-1, 1\}$ for $\forall i = 1, 2, \ldots, N$. $\mathbf{M}_{a,b,c}$ is a $N \times N \times N$ matrix where $\mathbf{M}_{a,b,c} = 0$ $\forall a \geq b$, $b \geq c$, or $a \geq c$.

In Hen's (2019) approach, we start by choosing a binary matrix $\{\mathbf{A}_{i,j}\}$ and a binary vector $\{\mathbf{b}_j\}$ to form a modulo 2 linear system of N equations in N variables.

$$\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \equiv \mathbf{b}_i \mod 2, \text{ for } i = 1, 2, \ldots, N. \tag{7}$$

This modulo 2 linear system of equations can always be solved in polynomial time using Gaussian elimination. In addition, as long as the binary matrix $\{\mathbf{A}_{i,j}\}$ is invertible, the solution (if exists) is unique. Suppose $\{x_1, ..., x_n\}$ are n binary variables. Then for given $\{\mathbf{A}_{i,j}\}$ and $\{\mathbf{b}_j\}$, we define

$$F(\{x_j\}) = \sum_{i=1}^{N} \mathbb{1}\Big(\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \not\equiv \mathbf{b}_i \mod 2 \Big). \tag{8}$$

$\mathbb{1}$ means indicator function here. Since F is a sum of N indicator functions, then $0 \leq F \leq N$ and the minimum bound is reached when $\{x_j\}$ is the solution to the modulo 2 linear system.

Let $s_j = 1 - 2x_j \in \{-1, 1\}$ for $j = 1, 2, \ldots, N$ be N Ising spins. Then we must have $\forall i = 1, 2, \ldots, m$

$$\prod_{j:\mathbf{A}_{i,j}=1} s_j = (-1)^{\mathbf{b}_i} \text{ if and only if } \sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \equiv \mathbf{b}_i \mod 2. \tag{9}$$

Then

$$\begin{aligned} F &= \sum_{i=1}^{N} \mathbb{1}\Big(\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \not\equiv \mathbf{b}_i \mod 2 \Big) \\ &= \sum_{i=1}^{N} \mathbb{1}\Big(\prod_{j:\mathbf{A}_{i,j}=1} s_j \neq (-1)^{\mathbf{b}_i} \Big), \text{ since } \prod_{j:\mathbf{A}_{i,j}=1} s_j \text{ and } (-1)^{\mathbf{b}_i} \in \{-1, 1\} \\ &= \frac{1}{2}\Big[\sum_{i=1}^{N} \Big(1 - (-1)^{\mathbf{b}_i} \prod_{j:\mathbf{A}_{i,j}=1} s_j \Big) \Big]. \end{aligned} \tag{10}$$

After dropping immaterial constants, we define

$$F_0(\{s_j\}) = \sum_{i=1}^{N} \left[(-1)^{\mathbf{b}_i} \prod_{j:\mathbf{A}_{i,j}=1} s_j \right]. \tag{11}$$

Note that, $F \geq 0$ and the minimum bound is reached when $\{x_j\}$ is the solution to the modulo 2 linear system. Thus, $F_0 \leq N$, and the maximum bound will be reached when $\{x_j \mid x_j = \frac{1}{2}(1 - s_j)\}$ is the solution to the modulo 2 linear system. In addition, as long as the matrix $\{\mathbf{A}_{i,j}\}$ is invertible, the solution (if exists) to the equation system must be unique, and then there must exist a single configuration maximize F_0 whose maximum value is exactly N.

Again, the Hamiltonian for simplified 3-body Ising model including only the cubic term to be $H(\{s_j\}) = \sum_{a<b<c} \mathbf{M}_{a,b,c} s_a s_b s_c$ Here, we assume, on each row of binary matrix $\{\mathbf{A}_{i,j}\}$, $\sum_{j=1}^{N} \mathbf{A}_{i,j} = 3$. Then, let $\mathbf{M}_{a,b,c} = (-1)^{\mathbf{b}_i}$ if $\exists i, a < b < c$ such that $\mathbf{A}_{i,a} = \mathbf{A}_{i,b} = \mathbf{A}_{i,c} = 1$, and $\mathbf{M}_{a,b,c} = 0$ otherwise. Then, we have $H(\{s_j\}) = F_0(\{s_j\})$.

Thus, we can construct an Ising model with unique optimal bound with known optimal value N as follows:

1. find an invertible binary matrix $\{\mathbf{A}_{i,j}\}$ and a binary vector $\{\mathbf{b}_i\}$, where $\sum_{j=1}^{N} \mathbf{A}_{i,j} = 3$, $\forall i$
2. solve the modulo 2 linear equation system $\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \equiv \mathbf{b}_i \mod 2$, for $i = 1, 2, \ldots, N$ to make sure the unique solution exists
3. define $\mathbf{M}_{a,b,c}$ be a $N \times N \times N$ matrix where $\mathbf{M}_{a,b,c} = (-1)^{\mathbf{b}_i}$ if $\exists i, a < b < c$ such that $\mathbf{A}_{i,a} = \mathbf{A}_{i,b} = \mathbf{A}_{i,c} = 1$, and $\mathbf{M}_{a,b,c} = 0$ otherwise
4. then we must have a unique optimal solution $s_{\max}$ for $H(s_{\max}) = \max(H(s)) = N$

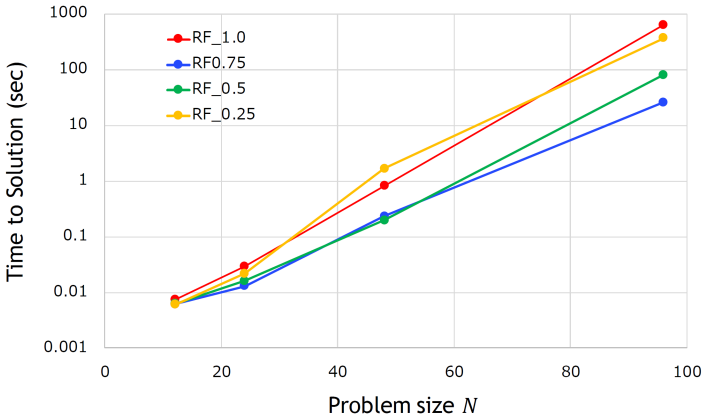

Fig. 8 Comparison of the minimum value for the time used to find the optimal state by Rejection-Free and PNS, for a random Ising model generated by 3R3XOR. Each dot represents 50 repeated simulation for a given problem size N. The number after RF represents the fraction of neighbors selected at each step. That is, RF_1.0 represents regular Rejection-Free, and RF_0.5 represents PNS where 50% of the neighbors are chosen randomly at each step, etc.

By constructing the special 3-body $N \times N \times N$ Ising model with a unique optimal solution of maximum bound N, we can examine the performance of Rejection-Free and PNS algorithm on these special Ising models. Again, uniform proposal distributions is used here and the neighbors are defined to be binary vectors with Hamming distance 1. We random generate the special Ising models with five different sizes $N = 12, 24, 48, 96$ and 192. For each of these five different sizes, we generate 50 different Ising models, and record the time used by the algorithms to reach the unique optimal state. The median of these 50 results for both Rejection-Free and PNS algorithm are shown in Figure 8. From this figure, 25% PNS performs comparably to Rejection-Free, while the 50% and 75% PNS perform significantly better.

10 Summary

This paper has illustrated Rejection-Free versions of the Simulated Annealing algorithm, which consider all neighbors at each step to avoid the inefficiency of rejections. In addition, to solve the issue of local maximum area, we have also proposed a Partial Neighbor Search (PNS) algorithm based on the Rejection-Free technique. Simulations of these methods on three different sets of examples have illustrated that PNS can produce significant speedups in optimization questions.

Acknowledgments

The author(s) would like to thank Fujitsu Ltd. and Fujitsu Consulting (Canada) Inc. for providing financial support.

References

Albright B (2007) An introduction to simulated annealing. The College Mathematics Journal 38(1):37–42.

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.

Bianchi L, Dorigo M, Gambardella LM, et al (2009) A survey on metaheuristics for stochastic combinatorial optimization. Natural Computing 8(2):239–287.

Block B, Preis T (2012) Computer simulations of the Ising model on graphics processing units. The European Physical Journal Special Topics 210(1):133–145.

Cipra BA (1987) An introduction to the Ising model. The American Mathematical Monthly 94(10):937–959.

Cipra BA (2000) The Ising model is NP-complete. SIAM News 33(6):1–3.

Floudas CA, Pardalos PM (2008) Encyclopedia of optimization, Springer Science & Business Media, pp 1538–1542.

Garey MR, Johnson DS, Stockmeyer L (1974) Some simplified NP-complete problems. In: Proceedings of the sixth annual ACM symposium on Theory of computing, pp 47–63.

Glover F (1989) Tabu search—part I. ORSA Journal on computing 1(3):190–206.

Glover F (1990) Tabu search—part II. ORSA Journal on computing 2(1):4–32.

Glover F, Kochenberger G, Du Y (2018) A tutorial on formulating and using QUBO models. arXiv:1811.11538

Hen I (2019) Equation planting: A tool for benchmarking Ising machines. Phys Rev Applied 12:011,003.

Hintze JL, Nelson RD (1998) Violin plots: a box plot-density trace synergism. The American Statistician 52(2):181–184.

Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences 79(8):2554–2558.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. science 220(4598):671–680.

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.

Ma YP, Sudakov I, Strong C, et al (2014) Ising model for melt ponds on Arctic sea ice. arXiv:1408.2487

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.

Rosenthal JS, Dote A, Dabiri K, et al (2021) Jump Markov chains and rejection-free Metropolis algorithms. Computational Statistics 36(4):2789–2811.

Rutenbar RA (1989) Simulated annealing algorithms: An overview. IEEE Circuits and Devices magazine 5(1):19–26.

Salkin HM, De Kluyver CA (1975) The knapsack problem: a survey. Naval Research Logistics Quarterly 22(1):127–144.

Schrijver A (2005) On the history of combinatorial optimization (till 1960). Handbooks in operations research and management science 12:1–68.

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.