2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

Monte Carlo Methods for Randomized Likelihood Decoding

Alankrita Bhatt, Jiun-Ting Huang, Young-Han Kim, J. Jon Ryu, and Pinar Sen
Department of Electrical and Computer Engineering, University of California, San Diego

***Abstract*—A randomized decoder that generates the message estimate according to the posterior distribution is known to achieve the reliability comparable to that of the maximum a posteriori probability decoder. With a goal of practical implementations of such a randomized decoder, several Monte Carlo techniques, such as rejection sampling, Gibbs sampling, and the Metropolis algorithm, are adapted to the problem of efficient sampling from the posterior distribution. Analytical and experimental results compare the complexity and performance of these Monte Carlo decoders for simple linear codes and the binary symmetric channel.**

I. Introduction

Consider a communication channel $p(\mathbf{y}|\mathbf{x})$ with input $\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{X}^n$ and output $\mathbf{y} = (y_1, \dots, y_n) \in \mathcal{Y}^n$. For example, $p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^n p_{Y|X}(y_i|x_i)$ corresponds to n transmissions over a discrete memoryless channel $p_{Y|X}(y|x)$ with input $x \in \mathcal{X}$ and output $y \in \mathcal{Y}$. Suppose that we wish to communicate a k-bit message $\mathbf{m} \in \{0,1\}^k$ over this channel by encoding the message $\mathbf{m}$ into $\mathbf{x}(\mathbf{m})$ and decoding the received output $\mathbf{y}$ into $\hat{\mathbf{m}}(\mathbf{y}) \in \{0,1\}^k$. The encoder mapping $\mathbf{x} : \{0,1\}^k \to \mathcal{X}^n$ is referred to as an (n,k) code for the communication channel $p(\mathbf{y}|\mathbf{x})$. Together with a given decoder mapping $\hat{\mathbf{m}} : \mathcal{Y}^n \to \{0,1\}^k$, the performance of the (n,k) code is measured by the average probability of decoding error

$$p_e = \mathsf{P}(\mathbf{M} \neq \hat{\mathbf{m}}(\mathbf{Y})),$$

where $\mathbf{M}$ is drawn randomly according to the uniform distribution over $\{0,1\}^k$.

A. Optimal Decoding

Let

$$p_{\mathbf{y}}(\mathbf{m}) = p(\mathbf{m}|\mathbf{y}) = \frac{1}{2^k}\frac{p(\mathbf{y}|\mathbf{x}(\mathbf{m}))}{p(\mathbf{y})} \tag{1}$$

denote the posterior distribution of $\mathbf{M}$ given $\mathbf{Y} = \mathbf{y}$. For any fixed encoder of an (n,k) code, the probability of decoding error is minimized by the *maximum a posteriori probability (MAP) decoding* rule

$$\hat{\mathbf{m}}_{\text{MAP}}(\mathbf{y}) = \arg\max_{\mathbf{m}} p_{\mathbf{y}}(\mathbf{m}). \tag{2}$$

By (1), the MAP decoding rule can be expressed equivalently as the *maximum likelihood (ML) decoding* rule

$$\hat{\mathbf{m}}_{\text{ML}}(\mathbf{y}) = \arg\max_{\mathbf{m}} p(\mathbf{y}|\mathbf{x}(\mathbf{m})). \tag{3}$$

In a straightforward implementation, the MAP (ML) decoding involves 2^k evaluations of the posterior (likelihood).

Suppose now that the channel is the binary symmetric channel

$$p(y|x) = \begin{cases} 1-p, & y = x, \\ p, & \text{otherwise} \end{cases}$$

with $\mathcal{X} = \mathcal{Y} = \mathbb{F}_2$ for some $p \in [0, 1/2]$. Equivalently, the channel output is $y = x \oplus Z$, where the binary additive noise $Z \sim \text{Bern}(p)$ is independent of the input x. Then the likelihood can be written as

$$p(\mathbf{y}|\mathbf{x}(\mathbf{m})) = \prod_{i: y_i = x_i(\mathbf{m})} (1-p) \cdot \prod_{i: y_i \neq x_i(\mathbf{m})} p,$$

and the ML decoding rule simplifies as the *minimum distance decoding* rule

$$\hat{\mathbf{m}}_{\text{MD}}(\mathbf{y}) = \arg\min_{\mathbf{m}} d_H(\mathbf{x}(\mathbf{m}), \mathbf{y}), \tag{4}$$

where $d_H(\mathbf{x}, \mathbf{y}) = |\{i \colon y_i \neq x_i\}|$ denotes the Hamming distance between $\mathbf{x}$ and $\mathbf{y}$. Instead of evaluating the distances between $\mathbf{y}$ and the 2^k codewords, one can perform minimum distance decoding by searching over the neighborhood of $\mathbf{y}$ step by step, first checking whether $\mathbf{y}$ is a codeword, then checking whether each of the $\binom{n}{1}$ 1-bit neighbors of $\mathbf{y}$ is a codeword, then checking whether each of the $\binom{n}{2}$ 2-bit neighbors of $\mathbf{y}$ is a codeword, and so on. Enumeration of all l-bit neighbors of $\mathbf{y}$ can be implemented efficiently using a technique of Cover [1]. On average, this approach requires roughly $2^{nH(p)}$ neighbors of $\mathbf{y}$ to be checked.

When the encoder is linear as

$$\mathbf{x}(\mathbf{m}) = \mathbf{m}\mathsf{G}$$

for some generator matrix $\mathsf{G} \in \mathbb{F}_2^{k \times n}$ and a corresponding parity check matrix $\mathsf{H} \in \mathbb{F}_2^{(n-k) \times n}$ such that $\mathsf{G}\mathsf{H}^T = 0$, then $\mathbf{x} \in \mathbb{F}_2^n$ is a codeword if and only if (iff) $\mathsf{H}\mathbf{x}^T = 0$. The MAP decoding rule in this case can be simplified as the *syndrome decoding* rule that computes the $(n-k)$-bit syndrome

$$\mathbf{s} = \mathsf{H}\mathbf{y}^T = \mathsf{H}(\mathbf{y} \oplus \mathbf{x}(\mathbf{m}))^T \tag{5a}$$

and maps it to the most likely error vector

$$\mathbf{z}(\mathbf{s}) = \arg\max_{\mathbf{z}: \mathsf{H}\mathbf{z}^T = \mathbf{s}} \prod_{i: z_i = 0} (1-p) \cdot \prod_{i: z_i = 1} p. \tag{5b}$$

The MAP estimate (or more precisely, the corresponding codeword) can then be found as

$$\mathbf{x}(\mathbf{m}) = \mathbf{y} \oplus \mathbf{z}(\mathbf{s}).$$

Syndrome decoding can be performed by precomputing and storing the mapping $\mathbf{z}(\mathbf{s})$ for 2^{n-k} syndromes.

All these approaches to optimal decoding discussed so far require time or space complexity that is exponential in k (assuming that k and n are linearly related). This is a rather fundamental limitation and the problem of optimal decoding of a general linear code is NP complete [2].

B. Randomized Likelihood Decoding

In [3], Yassaee, Aref, and Gohari proposed a suboptimal, randomized decoding rule as a theoretical alternative to MAP decoding so as to simplify the average error probability analysis of random code ensembles in the finite-blocklength regime and establish the desired second-order behavior (cf. [4]). In this *randomized likelihood* (RL) decoding rule, the message estimate is randomly chosen from the posterior distribution as

$$\hat{\mathbf{M}}_{\mathrm{RL}}(\mathbf{y}) \sim p(\mathbf{m}|\mathbf{y}). \tag{6}$$

In a sense, it plays the role of *joint typicality decoding* (see, for example, [5]) that is widely used to analyze the asymptotic first-order behavior of random codes.

Although quite simple, randomized likelihood decoding can achieve the decoding error probability comparable to that of optimal decoding. By the LCV lemma [6]–[8], for any channel $p(\mathbf{y}|\mathbf{x})$ and any code $\mathbf{x}(\mathbf{m})$,

$$\mathsf{P}(\mathbf{M} \neq \hat{\mathbf{M}}_{\mathrm{RL}}(\mathbf{Y})) \leq 2\,\mathsf{P}(\mathbf{M} \neq \hat{\mathbf{m}}_{\mathrm{MAP}}(\mathbf{Y})) := 2p_e^*. \tag{7}$$

If an even closer proxy is needed, it can be shown that the empirical mode from three random samples from $p(\mathbf{m}|\mathbf{y})$ achieves the multiplicative factor of $\leq$ 1.58 from MAP decoding, and that this factor converges to 1 exponentially as the number of samples increases [8]. Since most good codes operate at $p_e^* \ll 1$, the factor of two here is essentially negligible. In particular, RL decoding achieves the same asymptotic metrics such as error exponent and finite-length scaling as MAP decoding.

C. Monte Carlo Decoding

With the performance guarantee of RL decoding in (7), we now ask how one can generate a single sample from the posterior distribution $p_{\mathbf{y}}(\mathbf{m})$ efficiently. Our fundamental premise is that although $p_{\mathbf{y}}(\mathbf{m})$ is rather difficult to compute (due to the normalization factor $p(\mathbf{y})$ in (1)), it may be far easier to sample from. In fact, most Monte Carlo sampling algorithms rely only on the proportional quantity of likelihood $p(\mathbf{y}|\mathbf{x}(\mathbf{m}))$, which is straightforward to compute for most channels.

Thus motivated, we investigate in this paper several approaches to Monte Carlo sampling from the posterior distribution $p_{\mathbf{y}}(\mathbf{m})$. We first study *rejection sampling* (Section II) for generating an exact sample from $p_{\mathbf{y}}(\mathbf{m})$. We develop a decoding technique for the binary symmetric channel based on rejection sampling, which replaces the (exponential) space complexity of syndrome decoding in (5) by time complexity. With a goal of achieving polynomial decoding complexity, we then study two classical Markov chain Monte Carlo (MCMC) methods—*Gibbs sampling* (Section III) and *Metropolis sampling* (Section IV). We develop corresponding decoding techniques, tweaking them with heuristic and disciplined variations such as *simulated annealing* and parallelization. The performance of these *Monte Carlo decoding* techniques are illustrated through experimental results for simple toy examples. Although there is no conclusive evidence that these techniques would become feasible in practice, our simulation results demonstrate some potential of Monte Carlo decoding for augmenting conventional decoding techniques for certain communication applications.

On a historical note, the use of Monte Carlo methods for decoding traces to a 2001 talk by Radford Neal [9], who developed a decoding algorithm for LDPC codes based on Gibbs sampling (cf. Section III), but concluded that Gibbs decoding is too slow to be used in practice. A similar decoding method based on Gibbs sampling was also presented in the textbook [10], in which the asymptotic decoding performance of the Gibbs decoder was compared to that of a *belief propagation decoder*. Other than these two earlier investigations, Monte Carlo methods for decoding have hardly been studied in the literature, and the asymptotic performance guarantee has not been noted. The current paper seems to be the first to study various Monte Carlo methods for decoding.

Beyond decoding of error correction codes, Monte Carlo methods, especially those based on Markov chains, have found numerous applications. For example, MCMC-based algorithms have been developed for difficult combinatorial optimization problems (see [11] and the references therein), which reduce the computational complexity of (approximate) counting from exponential to polynomial, a feat not replicated by any other approach. As yet another example, Monte Carlo (especially MCMC) methods have been applied to several signal processing problems [12]. The most relevant to our problem is the MCMC-based detection method for multiple-input multiple-output (MIMO) systems [13], [14], whereby multiple samples are generated by Gibbs sampling. We remark that our Monte Carlo approach to decoding can be applied to MIMO (or any) detection problems with performance guarantee in (7), which appears to be previously unknown.

II. Decoding Based on Rejection Sampling

We start with a general description of rejection sampling and then discuss the randomized likelihood decoding based on rejection sampling. Let $p(v) = p^*(v)/Z_p$ denote the target pmf we wish to sample from where Z_p is the normalization constant. Let $q(v) = q^*(v)/Z_q$ be another pmf we can easily sample from. Suppose that for every v with $q^*(v) > 0$, the ratio satisfies

$$\frac{p^*(v)}{q^*(v)} \leq 1 \tag{8}$$

and can be computed efficiently. In rejection sampling [15], a sample v from $q(v)$ and a sample u from $\mathrm{Unif}([0,1])$ are generated randomly and independently from each other. If $u \leq \frac{p^*(v)}{q^*(v)}$, the sample v is accepted; otherwise, v is rejected and the steps are repeated from the beginning.

At every step in the rejection sampling, the points $(v, q^*(v) \cdot u)$ are generated independently from the previous steps and uniformly random over the region under the curve $q^*(v)$ as illustrated in Fig 1. Since all the points $(v, q^*(v) \cdot u)$ above $q^*(v)$ are rejected, the accepted point is generated uniformly random over the region under the curve $p^*(v)$. Therefore, the accepted sample v is generated from the distribution $p(v)$.

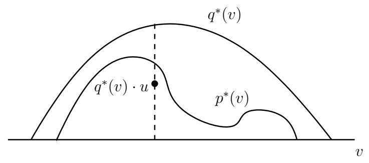

Fig. 1. Illustration for rejection sampling

We now consider the time complexity of rejection sampling. Let N denote the number of samples drawn from $q(v)$ until the first sample is accepted, i.e., the number of iterations. N is a geometric random variable with success probability δ, where

$$\begin{aligned}\delta &= \mathsf{P}\left(U \leq \frac{p^*(V)}{q^*(V)}\right) \\ &\overset{(a)}{=} \sum_v q(V=v)\,\mathsf{P}\left(U \leq \frac{p^*(v)}{q^*(v)}\right) \\ &\overset{(b)}{=} \sum_v q(V=v)\frac{p^*(v)}{q^*(v)} \\ &= \sum_v \frac{p^*(v)}{Z_q}.\end{aligned}$$

Here (a) follows since U and V are independent, and (b) follows since $U \sim \mathrm{Unif}([0,1])$. Therefore, the expected number of iterations for rejection sampling is $1/\delta$, or equivalently

$$\mathsf{E}[N] = Z_q/Z_p. \tag{9}$$

We are now ready to discuss how one can use rejection sampling for RL decoding of an error correction code. A rejection sampling based RL decoder draws a sample from $p_{\mathbf{y}}(\mathbf{m})$ by setting

$$p^*(\mathbf{m}) = p(\mathbf{y}|\mathbf{x}(\mathbf{m})) \tag{10}$$

with a proper choice of $q_{\mathbf{y}}(\mathbf{m}) = q^*(\mathbf{m})/Z_q(\mathbf{y})$ such that $q^*(\mathbf{m})$ satisfies the condition in (8). The following lemma describes the expected number of iterations required by this class of decoders.

Lemma 1. *The expected number of iterations for a rejection sampling based RL decoder is* $2^{-k}\sum_{\mathbf{y}\in\mathcal{Y}^n} Z_q(\mathbf{y})$*, where* $\mathcal{Y}_n$ *denotes the set of* $\mathbf{y}$ *such that* $p(\mathbf{y}|\mathbf{m}) > 0$ *for some* $\mathbf{m}$.

Proof: The expected number of iterations is written as

$$\begin{aligned}\mathsf{E}[N] &\overset{(a)}{=} \mathsf{E}[\mathsf{E}[N|\mathbf{x}(\mathbf{M}), \mathbf{Y}]] \\ &= \sum_{\mathbf{m}}\sum_{\mathbf{y}} 2^{-k} p(\mathbf{y}|\mathbf{x}(\mathbf{m}))\,\mathsf{E}[N|\mathbf{x}(\mathbf{m}), \mathbf{y}] \\ &\overset{(b)}{=} \sum_{\mathbf{m}}\sum_{\mathbf{y}} 2^{-k} p(\mathbf{y}|\mathbf{x}(\mathbf{m}))\frac{Z_q(\mathbf{y})}{Z_p(\mathbf{y})} \\ &\overset{(c)}{=} \sum_{\mathbf{m}}\sum_{\mathbf{y}} 2^{-k} p(\mathbf{y}|\mathbf{x}(\mathbf{m}))\frac{Z_q(\mathbf{y})}{\sum_{\mathbf{m}'} p(\mathbf{y}|\mathbf{x}(\mathbf{m}'))} \\ &= 2^{-k}\sum_{\mathbf{y}} Z_q(\mathbf{y})\frac{\sum_{\mathbf{m}} p(\mathbf{y}|\mathbf{x}(\mathbf{m}))}{\sum_{\mathbf{m}'} p(\mathbf{y}|\mathbf{x}(\mathbf{m}'))} \\ &= 2^{-k}\sum_{\mathbf{y}} Z_q(\mathbf{y}),\end{aligned}$$

where (a) follows by the iterated expectation theorem, (b) follows by (9), and (c) follows by (10). ■

The following presents a concrete example of a rejection sampling based RL decoder.

Example 1. We consider decoding of a systematic linear code over the binary symmetric channel with cross over probability $p \in (0, 0.5]$. For this channel, (10) is simplified to

$$p^*(\mathbf{m}) = (p/\overline{p})^{d_H(\mathbf{x}(\mathbf{m}),\mathbf{y})}\overline{p}^n,$$

where $\overline{p} := 1 - p$. We use the rejection sampling based RL decoder specified by

$$q_{\mathbf{y}}(\mathbf{m}) = \left(\frac{p}{\overline{p}}\right)^{d_H(\mathbf{m},\mathbf{y}_1)}\overline{p}^k,$$

and

$$Z_q = \overline{p}^{(n-k)},$$

where $\mathbf{y}_1$ denote the first k bits of the received sequence $\mathbf{y}$. Note that the requirement in (8) is satisfied. Let $\mathbf{y}_2$ denote the last $(n-k)$ bits of $\mathbf{y}$. Upon receiving $\mathbf{y}$, this RL decoder executes the following steps.

1. Draw $\mathbf{m}$ from $q_{\mathbf{y}}(\mathbf{m})$ by passing $\mathbf{y}_1$ through a binary symmetric channel with cross over probability of p.
2. Compute parity bits $(\mathbf{z}(\mathbf{m}))$ for the sample $\mathbf{m}$.
3. Draw u from $\mathrm{Unif}([0,1])$.
4. If $u \leq (p/\overline{p})^{d(\mathbf{z}(\mathbf{m}),\mathbf{y}_2)}$, accept $\mathbf{m}$ and declare as an estimate; otherwise, reject $\mathbf{m}$ and return to step 1.

Corollary 1. *The expected number of iterations for the rejection sampling based RL decoder described in Example 1 is* $(2\overline{p})^{n-k}$.

Proof: It follows by incorporating $Z_q = \overline{p}^{(n-k)}$, and $|\mathcal{Y}^n| = 2^n$ in Lemma 1. ■

Within Example 1, we consider a $(23, 12)$ Golay code and the binary symmetric channel with cross over probability $p = 0.06$. By Corollary 1, the expected number of iterations for this case is computed as $(2 \cdot 0.94)^{11} \approx 1037$. One way to improve the block error rate (BLER) of the decoder in Example 1 at any iteration step is to output the most likely sample among the ones that were drawn from $q_{\mathbf{y}}(\mathbf{m})$ until the current step. The performance of this decoder is shown in Fig. 2.

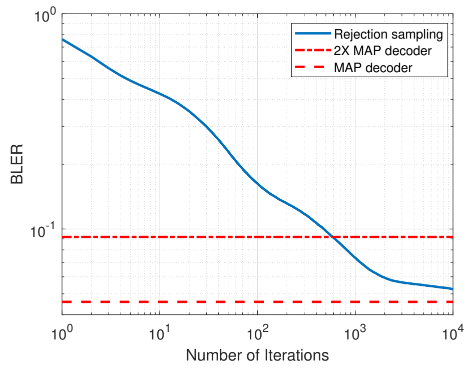

Fig. 2. Performance of the RL decoder in Example 1 for a $(23, 12)$ Golay code over BSC(0.06)

Remark 1. The RL decoder given in Example 1 replaces the space complexity of syndrome decoding by the time complexity. This time/space substitution may be particularly handy when different random codes are used per session (for example, for secrecy, anti-jamming, or robustness applications) and hence table lookup becomes infeasible. Moreover, this RL decoder is particularly favorable over syndrome decoding when p is large, the condition of which is typically consistent with large $n - k$.

Remark 2. We also visit two other (out of infinitely many) rejection sampling decoders for the binary symmetric channel. The first one looks at the output of the same binary symmetric channel with $\mathbf{y}$ as the input, and accepts this output if and only if it is a codeword. It is easy to see that expected number of iterations is 2^{n-k}. The second decoder draws $\mathbf{m}$ from $\{0, 1\}^k$ uniformly at random and finds the corresponding codeword $\mathbf{x}(\mathbf{m})$. It then generates a sample u from $\mathrm{Unif}([0, 1])$, and accepts $\mathbf{m}$ if and only if $u < (p/\overline{p})^{d_H(\mathbf{x}(\mathbf{m}),\mathbf{y})}$. For this decoder, the expected number of iterations is $(2\overline{p})^n$.

It is surprising to see that even with such a simple Monte Carlo method, it is possible to construct a decoder that is comparable (even more favorable for large values of p) with the well-known syndrome decoder in terms of the performance and decoding complexity. In search of polynomial-complexity RL decoders, we next investigate more advanced MCMC methods.

III. Gibbs Decoding

MCMC [16] is a collection of Monte Carlo methods in which samples are taken from an ergodic Markov chain that is easy to run. It is typically assumed that the distribution of the process converges to the unique stationary distribution asymptotically and thus that after a sufficient number of iterations, samples are distributed approximately according to the target stationary distribution. Most Monte Carlo methods are inexact and result in some mismatch between the nominal distribution and the actual distribution of the sample. This should be only a minor concern to us, however. If a Monte Carlo method generates a sample from $p'(\mathbf{m}|\mathbf{y})$ within total variation distance δ from the desired posterior distribution $p(\mathbf{m}|\mathbf{y})$, then the probability error of the inexact Monte Carlo decoder

$$\hat{\mathbf{M}}_{\mathrm{RL}}(\mathbf{y}) \sim p'(\mathbf{m}|\mathbf{y}) \tag{11}$$

is upper bounded as

$$\mathsf{P}(\mathbf{M} \neq \hat{\mathbf{M}}_{\mathrm{RL}}(\mathbf{Y})) \leq 2p_e^* + \delta, \tag{12}$$

which is once again comparable to the MAP decoding performance, provided that δ is sufficiently small. The key here is the speed of convergence (mixing time) [17] of the Markov chain which can be exponentially faster than non-MCMC methods such as rejection sampling. We now study the use of two of the most classical and popular MCMC techniques, Gibbs sampling and Metropolis sampling, for the decoding problem.

Consider a d-variate probability density function $p(\mathbf{v})$, $d \geq 2$, from which we wish to take a sample. Suppose that sampling each variable from the conditional marginal distribution is easier than sampling all variables from the joint distribution. Gibbs sampling algorithm can then be summarized in the following steps.

1. Initialize a starting point $\mathbf{v}^{(0)} = (v_1^{(0)}, v_2^{(0)}, \ldots, v_d^{(0)})$.
2. On the t-th iteration ($t = 1, 2, \ldots$),
   (a) randomly pick a coordinate $i \sim \mathrm{Unif}[d]$.
   (b) sample the i-th coordinate from the conditional distribution $V_i^{(t)} \sim p(v_i|\mathbf{v}_{\neg i}^{(t-1)})$, where
   $$\mathbf{v}_{\neg i}^{(t-1)} = (v_1^{(t-1)}, \ldots, v_{i-1}^{(t-1)}, v_{i+1}^{(t-1)}, \ldots, v_d^{(t-1)}).$$
   (c) fix the remaining coordinates $v_j^{(t)} = v_j^{(t-1)}$ for all $j \in [d]\backslash\{i\}$.

It can be proven that the distribution of the Markov chain generated by the above algorithm converges to the target distribution $p(\mathbf{v})$.

The multivariate structure of Gibbs sampling fits our decoding problem very well—in order to sample from posterior distribution $p(\mathbf{m}|\mathbf{y})$, our Gibbs decoder performs a random walk on the state space of 2^k messages by generating the i-th bit $M_i^{(t)} \sim p(m_i|\mathbf{y}, \mathbf{m}_{\neg i}^{(t-1)})$, keeping all other positions the same and iterating over the k bits.

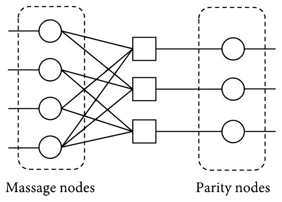

Fig. 3. A Tanner graph for a code with 4 message bits and 3 parity bits.

We note that there is some intriguing connection between Gibbs decoding and conventional BP decoding. Both involve iterative "message passing" between the check nodes and the variable nodes. The main difference is that, while the update rule for BP is deterministic, the one for parallel Gibbs is random. Fig. 3 illustrates a typical Tanner graph of a linear block code with the parity-check matrix chosen to be systematic so that the variable nodes are divided into message nodes and parity nodes. Suppose that, for example, the Gibbs decoder wishes to update the first message node. The Gibbs message-passing algorithm takes the following steps.

1. Every message node j other than node 1 sends its current estimate $m_j^{(t)}$, and every parity node i sends the channel output bit y_i to the check nodes.
2. From each check node i to which information node 1 is connected, a soft likelihood $p(m_1|y_i, \mathbf{m}_{\neg 1}^{(t)})$ is sent to message node 1.
3. Message node 1 generates a new estimate

$$\begin{aligned} M_1^{(t+1)} &\sim p(m_1|\mathbf{y}, \mathbf{m}_{\neg 1}^{(t)}) \\ &\propto p(m_1|y_1) \prod_{i \in I_1} p(m_1|y_i, \mathbf{m}_{\neg 1}^{(t)}), \end{aligned} \tag{13}$$

where $I_1 = \{i\colon \text{parity bit } i \text{ in which message bit 1 participates}\}$.

A recent paper [18] presents a hybrid approach of MCMC and BP decoding.

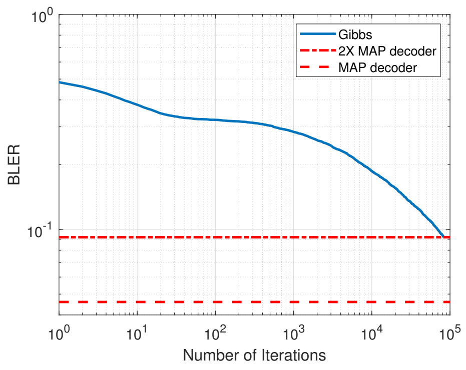

Fig. 4. Performance of the Gibbs decoder for a $(23, 12)$ Golay code over BSC(0.06)

Fig. 4 shows the performance of the Gibbs decoder for a $(23, 12)$ Golay code over a BSC with crossover probability 0.06. The generator matrix is chosen to be systematic so that the starting point of the chain can be set as the message bits of the channel output, which are the bitwise MAP estimates given those output bits. As seen in this plot, the convergence time is not satisfactory. So in its naive form, Gibbs sampling is not practical due to slow dynamics.

To speed up Gibbs sampling certain methods have previously been shown to be effective. The first one is simulated annealing [19], a popular technique used for avoiding local extrema in random walks (getting trapped in which leads to slow convergence). Instead of generating sample from $p(\mathbf{m}|\mathbf{y})$, we run a Markov chain that generates a sample form $p^\alpha(\mathbf{m}|\mathbf{y})$, where $\alpha > 0$. As α tends to infinity, the behavior of the Markov chain becomes more like finding the steepest ascent in a lazy manner – if flipping a bit yields higher likelihood, then it flips that bit; otherwise, it stays at the current state. On the contrary, as α goes to 0, the Markov chain behaves more like a lazy simple random walk. Fig. 5 compares the performance of annealed Gibbs decoders with various temperature parameters for a $(40, 20)$ irregular LDPC code over a BSC with crossover probability 0.04. The parameter $k = 20$ is chosen to allow accurate simulation of the MAP decoding error probability. While smaller α (higher temperature) makes the Markov chain move more freely, it also flattens the distribution we sample from. Among the values considered, $\alpha = 0.25$ is the optimal in terms of the speed of convergence.

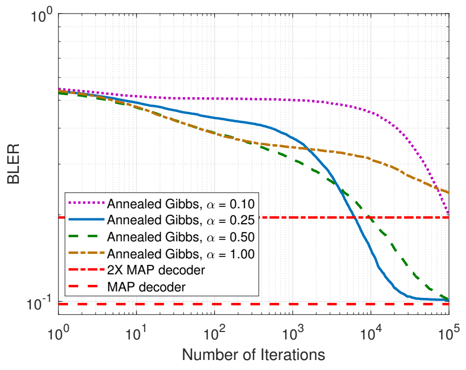

Fig. 5. Performance of annealed Gibbs decoders for a $(40, 20)$ irregular LDPC code over BSC(0.04)

In [9], Neal considered a different Gibbs decoder in which a random walk is performed over $\mathbb{F}_2^n$, i.e. the codeword space (as opposed to the message space $\mathbb{F}_2^k$), and a non-codeword state is penalized depending on how many parity-check equations are dissatisfied by it. Neal also used simulated annealing. The resulting doubly annealed posterior distribution has the form

$$p_{\alpha,\beta}(\mathbf{x}|\mathbf{y}) \propto p^\alpha(\mathbf{x}|\mathbf{y}) \prod_{i=1}^{n-k} r_\beta([\mathsf{H}\mathbf{x}]_i),$$

where $r_\beta(y) = (1-y)\beta + (1-\beta)/2$, α is the heuristic temperature control parameter, and β is the soft-parity annealing parameter. The performance of the soft-parity Gibbs decoder is plotted in Fig. 6. For each simulation, β is increased linearly with the number of iterations from 0.01 to 1, which is annealing schedule used in [9].

Another favorable technique is using Gibbs sampling with block updates. Instead of updating one coordinate per iteration, for block Gibbs a set B of multiple coordinates is randomly chosen and updated according to the conditional marginal distribution $p(\mathbf{m}_B|\mathbf{y}, \mathbf{m}_{\neg B}^{(t-1)})$. As long as the blocksize is not too large so that the conditional marginal distribution is easy to sample from, empirically the block Gibbs algorithm can substantially improve the speed of convergence, which can be seen in Fig. 7. Compared to the belief propagation (BP) decoder, Gibbs decoders have slower convergence speed. However, while there is a gap between BP decoding and MAP decoding, the error probability of Gibbs decoders converges to the MAP error probability after sufficient iterations. For $\alpha = 0.25$ and blocksize $= 3$,

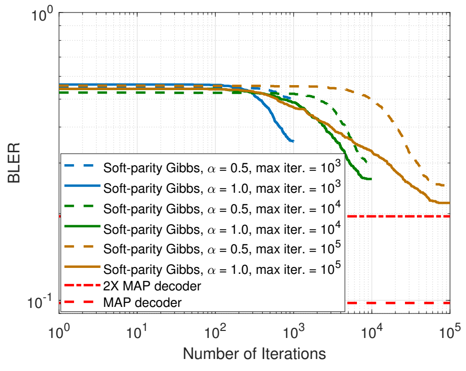

Fig. 6. Performance of soft-parity Gibbs decoders for a $(40, 20)$ irregular LDPC code over BSC(0.04)

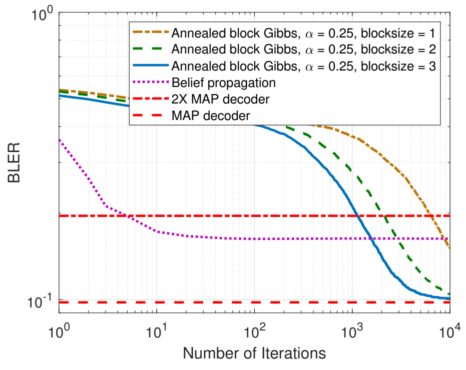

Fig. 7. Performance of annealed block Gibbs decoders for a $(40, 20)$ irregular LDPC code over BSC(0.04)

the error probability of the annealed block Gibbs decoder achieves twice the MAP error probability within about 10^3 iterations, which is exponentially better than 2^{20}, the time complexity of MAP/ML decoding (or the space complexity of syndrome decoding, or the time complexity of the rejection sampling based RL decoder).

Remark 3. Due to the resemblance between Gibbs and BP decoders, it is natural to consider parallel Gibbs sampling, in which some of all the coordinates are updated simultaneously. In our setting, parallel Gibbs updates become

$$M_i^{(t)} \sim p(m_i|\mathbf{y}, \mathbf{m}_{\neg i}^{(t-1)}), \quad i = 1, 2, \ldots, k.$$

Nevertheless, the correctness is lost through the parallelism and the algorithm is not guaranteed to converge to the target distribution. Analysis can be found in [20], [21].

IV. Metropolis Decoding

Let $Q(v, v')$ be a symmetric transition matrix of a Markov chain and $p(v)$ be an unrelated, target distribution one wishes to sample from. The Metropolis algorithm modifies the underlying Markov chain Q suitably to obtain a new "Metropolized" chain P with stationary distribution p. The algorithm works as follows. Given the current state $V_i = v_i$, a new state V_{i+1} is generated as per the distribution $Q(v_i, \cdot)$. If $p(v_{i+1}) \geq p(v_i)$, then the new state $V_{i+1} = v_{i+1}$ is accepted. Otherwise, it is accepted with probability $p(v_{i+1})/p(v_i)$. In summary, the Metropolized chain has the transition matrix

$$P(v, v') = \begin{cases} Q(v, v') \min\left\{1, \frac{p(v')}{p(v)}\right\}, & v' \neq v, \\ 1 - \sum\limits_{v' : v' \neq v} Q(v, v') \min\left\{1, \frac{p(v')}{p(v)}\right\}, & v' = v, \end{cases} \tag{14}$$

which can be shown to have $p(v)$ as its stationary distribution. When the underlying Markov chain Q is asymmetric, a straightforward generalization of (14) called the Metropolis–Hastings algorithm is used to alter Q, so that the final stationary distribution is $p(v)$. Note that Gibbs sampling discussed in Section III is a special case of the Metropolis–Hastings algorithm (see, for example, Exercise 29.2 of [15]).

For the decoding problem, the target stationary distribution would be chosen to be $p(\mathbf{m}|\mathbf{y})$. Note that the knowledge of the target distribution is necessary only up to a constant multiplicative factor in the accept/reject step and hence that the Metropolis decoder only needs to compute the likelihood $p(\mathbf{y}|\mathbf{x}(\mathbf{m}))$. Upon fixing the target stationary distribution, to design a Metropolis decoder a suitable Q needs to be chosen. This design choice is critical since the underlying Markov chain in the Metropolis algorithm has a major role in the speed of convergence of the Metropolized chain.

Apart from a good choice of Q, another heuristic method shown to be effective in speeding up convergence is the use of *temperature control*. At a high level, temperature control entails changing the target stationary distribution p as the algorithm progresses. Typically, the time-varying stationary distribution of the Markov chain is altered as p^{α_t}. Here, α_t is referred to as the "inverse temperature" and gradually increases from 0 to 1 with t. This method is known as simulated annealing. While inexact, it has been shown to be effective for a variety of combinatorial optimization problems, and there have been several proposals on the optimal pattern of varying α_t [22], [23].

Theoretical analyses of most Metropolis algorithms are elusive, and one of the reasons for this difficulty is that the mixing time of the Metropolized chain P does not appear to be directly related to the mixing time of the underlying Q. However, there have been some analytical studies on MCMC methods (see [11] and the references therein). One study in particular that is pertinent to our scenario is the analysis by Diaconis and Hanlon [24]. They considered the underlying Markov chain to be the symmetric random walk on the n-dimensional hypercube (see below) and the target stationary distribution to be

$$p(\mathbf{x}) = \frac{\theta^{\mathrm{wt}(\mathbf{x})}}{(1+\theta)^n}$$

for $0 < \theta < 1$, where $\mathrm{wt}(\mathbf{x})$ denotes the Hamming weight of $\mathbf{x}$. For $\theta \leq \frac{1}{2}$, this is the chain that is formed when we wish to decode the output over a BSC with parameter θ, assuming the all-zero codeword is transmitted and the trivial rate-1 code is employed. Interestingly, in this case the mixing time is $O(n \log n)$, which is also the mixing time of the symmetric random walk on the hypercube. The analysis relies on calculation of the spectral gap of the Metropolized chain, and the corresponding mixing time bound in terms of the spectral gap [17].

We now report empirical results of three methods used to reach the goal of designing a Metropolis decoder competitive in terms of the tradeoff between high performance and low-complexity: 1) to vary the underlying chain Q, 2) to employ temperature-control, and 3) to parallelize the sampling process to replace time complexity by space complexity.

To study the effect of the underlying Q on convergence, we start with one of the simplest chains, the symmetric random walk on the hypercube. In the symmetric random walk, a new state $\mathbf{m}'$ is generated by flipping one of the k bits in $\mathbf{m}$ at random and accepting/rejecting based on the ratio of the likelihoods $p(\mathbf{y}|\mathbf{x}(\mathbf{m}))$ and $p(\mathbf{y}|\mathbf{x}(\mathbf{m}'))$. In order to avoid periodicity of the Markov chain, we make the random walk lazy and add some positive probability of staying in the same state. Fig. 8 shows the performance of this Hypercube Metropolis decoder for a $(23, 12)$ systematic Golay code over a BSC with crossover probability 0.06. The chain has "laziness" probability 0.01. The straight dash/dot line at the bottom of the plot is twice the MAP decoding error probability.

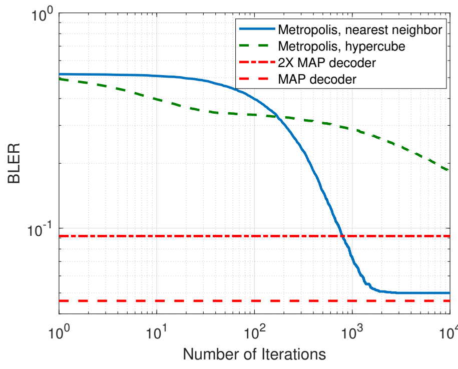

Fig. 8. Performance of Metropolis decoders for a $(23, 12)$ Golay code over BSC(0.06)

To study a slightly larger code, we consider again the irregular $(40, 20)$ LDPC code. The performance of the Metropolis decoders is shown in Fig. 9. The dashed curve shows the error probability of this Metropolis decoder for a BSC with $p = 0.04$. The convergence is slow and it is still far off from the theoretical guarantee when the number of iterations is limited at 2^{20}. This is in contrast to the mixing time of the lazy random walk on the hypercube, despite its exponentially large state space, is $O(k \log k)$ [17].

As an alternative for the underlying Markov chain, we considered Q to be a variant of random walk on the *nearest-neighbor graph*. This is the random walk in which a codeword jumps to one of its closest neighbor (in terms of Hamming distance). For example, the $(15, 11)$ Hamming code has 35 codewords of Hamming weight 3; hence each codeword jumps to one of its 35 neighbors uniformly at

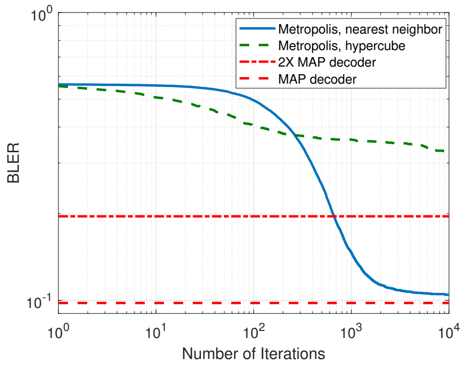

Fig. 9. Performance of Metropolis decoders for a $(40, 20)$ irregular LDPC code over BSC(0.04)

random (with laziness parameter 0.01). The performance of this nearest-neighbor walk can be seen to significantly outperform the hypercube random walk in Fig.s 8 and 9, and especially so for the LDPC code. The solid curve in Fig. 9 shows the error probability of the Metropolis decoder that runs over the third-level nearest-neighbor random walk, with beginning state chosen as the first k bits of the output. The error probability converges to the target of twice the MAP error probability very fast and even approaches the MAP error probability. This decoder is exponentially better than both the naive hypercube Metropolis decoder and the rejection sampling decoder in Section II.

If the underlying distribution in a Markov chain is highly concentrated on a few values (i.e. is highly non-uniform), the chain tends to get stuck in states which are local optima, leading to many rejections of proposed states and subsequently a prohibitively large mixing time. *Temperature control* methods counter this by making the accept/reject decision at each step based on the likelihood ratio $p^{\alpha_t}(\mathbf{y}|\mathbf{m})$ where $0 \le \alpha_t < 1$. Typically, α_t is gradually increased to 1. Thus, in the initial steps of the Markov chain, there is a higher probability of acceptance and more of the solution space can be explored, thereby helping avoid local optima. Another approach which was used for example in [14] was to *fix* an optimal α and use it throughout the algorithm—so $\alpha_t = \alpha$ for all t. Out of the α values considered, $\alpha = 0.25$ appeared to be the optimal in terms of speed of convergence, as seen in Fig. 10

Another method to decrease the number of iterations is at each iteration to take $L > 1$ samples from the underlying Markov chain Q, and keep the most likely one among those samples as the new state $\mathbf{m}'$. The standard accept/reject step is then performed for the new state $\mathbf{m}'$. This process, which is referred to as the parallel-Metropolis, can be viewed as running Metropolis sampling on a new chain $\tilde{Q}$ derived from Q, where

$$\tilde{Q}(\mathbf{m}, \mathbf{m}_1)$$
$$= \sum_{\mathbf{m}_2,\ldots,\mathbf{m}_L} \prod_{i=1}^{L} Q(\mathbf{m}, \mathbf{m}_i) \frac{\mathsf{P}(p(\mathbf{m}_1|\mathbf{y}) \ge p(\mathbf{m}_k|\mathbf{y}), k \in [2:L])}{|\arg\max_j \; p(\mathbf{m}_j|\mathbf{y})|}$$

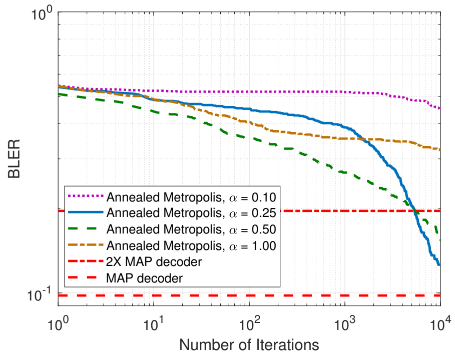

Fig. 10. Effect of temperature control on hypercube Metropolis decoder for a $(40, 20)$ irregular LDPC code over BSC(0.04)

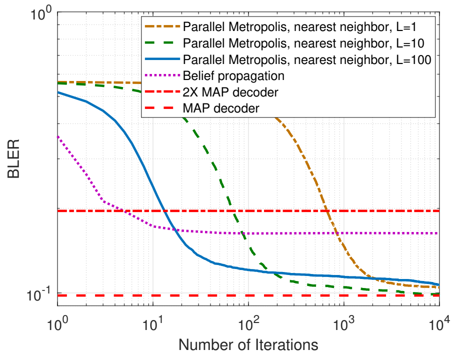

Fig. 11. Performance of parallel-Metropolis decoder for a $(40, 20)$ LDPC code

and $\arg\max_j \; p(\mathbf{m}_j|\mathbf{y}) := \{j\colon p(\mathbf{m}_j|\mathbf{y}) \geq p(\mathbf{m}_k|\mathbf{y}), k \in [1:L]\}$. Since $\tilde{Q}$ is not necessarily symmetric, Metropolis-Hasting update should be used in the accept/reject step to have the convergence guarantee. Even when Q is fixed to the nearest neighbor random walk, however, $\tilde{Q}$ simplifies up to a constant to

$$\tilde{Q}(\mathbf{m}, \mathbf{m}_1) \propto \sum_{\substack{\mathbf{m}_2,\dots,\mathbf{m}_L \\ \in \mathcal{N}(\mathbf{m})}} \frac{\mathsf{P}(p(\mathbf{m}_1|\mathbf{y}) \geq p(\mathbf{m}_k|\mathbf{y}), k \in [2:L])}{|\arg\max_j \; p(\mathbf{m}_j|\mathbf{y})|},$$

which is still difficult to compute, where $\mathcal{N}(\mathbf{m})$ is the set of nearest neighbors of the codeword $\mathbf{x}(\mathbf{m})$. For the $(40, 20)$ LDPC code and BSC with $p = 0.04$, we consider the third level nearest neighbor random walk as the chain Q, and parallelize Metropolis decoding by using L samples at each step. It is experimentally shown that using Metropolis update in the accept/reject step under the symmetric $\tilde{Q}$ assumption has the same performance as using the ideal Metropolis-Hasting update. So, we continue with this assumption to obtain the BLER performance of the parallel-Metropolis decoder for different L values shown in Fig. 11. For comparison, the performance of the belief propagation (BP) and MAP decoders are included. The results have several implications. First, parallel Metropolis can replace time complexity by space complexity. Second, the Metropolis decoders built upon the nearest neighbor walk outperform the BP decoder, which is not be able to achieve the BLER of the MAP decoder.

References

[1] T. M. Cover, "Enumerative source encoding," *IEEE Trans. Inf. Theory*, vol. 19, pp. 73–77, Jan. 1973.

[2] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Trans. Inf. Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[3] M. H. Yassaee, M. R. Aref, and A. Gohari, "A technique for deriving one-shot achievability results in network information theory," in *Proc. IEEE Int. Symp. Inf. Theory*, pp. 1287–1291, IEEE, July 2013.

[4] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, 2012.

[6] J. Liu, P. Cuff, and S. Verdú, "On α-decodability and α-likelihood decoder," in *Proc. 55th Ann. Allerton Conf. Comm. Control Comput.*, 2017.

[7] S. Kudekar, S. Kumar, M. Mondelli, H. Pfister, and R. Urbanke, "Comparing the bit-MAP and block-MAP decoding thresholds of Reed-Muller codes on BMS channels," in *Proc. IEEE Int. Symp. Inf. Theory*, (Barcelona, Spain), pp. 1755–1759, July 2016.

[8] A. Bhatt, J.-T. Huang, Y.-H. Kim, J. J. Ryu, and P. Sen, "Variations on a theme by Liu, Cuff, and Verdú: The power of posterior sampling." to appear in *Proc. IEEE Inf. Theory Workshop*, 2018.

[9] R. M. Neal, "Monte Carlo decoding of LDPC codes," tech. rep., 2001.

[10] M. Mézard and A. Montanari, *Information, Physics, and Computation*. Oxford University Press, 2009.

[11] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Springer, 2012.

[12] A. Doucet and X. Wang, "Monte Carlo methods for signal processing: A review in the statistical signal processing context," *IEEE Signal Proces. Magazine*, vol. 22, no. 6, pp. 152–170, 2005.

[13] H. Zhu, B. Farhang-Boroujeny, and R. R. Chen, "On performance of sphere decoding and Markov chain Monte Carlo detection methods," in *IEEE 6th Workshop Signal Proces. Adv. Wireless Commu.*, pp. 86–90, June 2005.

[14] B. Hassibi, M. Hansen, A. G. Dimakis, H. A. J. Alshamary, and W. Xu, "Optimized Markov Chain Monte Carlo for signal detection in MIMO systems: An analysis of the stationary distribution and mixing time," *IEEE Trans. Signal Process.*, vol. 62, no. 17, pp. 4436–4450, 2014.

[15] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[16] S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, *Handbook of Markov Chain Monte Carlo*, vol. 2. CRC press New York, NY, 2011.

[17] D. A. Levin and Y. Peres, *Markov Chains and Mixing Times*, vol. 107. American Mathematical Soc., 2017.

[18] S.-S. Ahn, M. Chertkov, and J. Shin, "Synthesis of MCMC and belief propagation," in *Adv. Neural Inform. Process. Syst.*, pp. 1453–1461, 2016.

[19] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," in *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications* (M. Mézard, G. Parisi, and M. Virasoro, eds.), pp. 339–348, Singapore: World Scientific, 1987.

[20] M. Johnson, J. Saunderson, and A. Willsky, "Analyzing Hogwild parallel Gaussian Gibbs sampling," in *Adv. Neural Inform. Process. Syst.*, pp. 2715–2723, 2013.

[21] C. De Sa, K. Olukotun, and C. Ré, "Ensuring rapid mixing and low bias for asynchronous Gibbs sampling," in *JMLR Workshop Conf. Proc.*, vol. 48, pp. 1567–1576, NIH Public Access, 2016.

[22] B. Hajek, "Cooling schedules for optimal annealing," *Math. Oper. Res.*, vol. 13, no. 2, pp. 311–329, 1988.

[23] R. Holley and D. Stroock, "Simulated annealing via Sobolev inequalities," *Comm. Math. Phys.*, vol. 115, no. 4, pp. 553–569, 1988.

[24] P. Diaconis and P. Hanlon, "Eigen analysis for some examples of the Metropolis algorithm," in *Hypergeometric Functions on Domains of Positivity, Jack Polynomials, and Applications: Proceedings of an AMS Special Session Held March 22-23, 1991 in Tampa, Florida*, vol. 138, pp. 99–117, American Mathematical Soc., 1992.