A Sampling Technique Based on LDPC Codes

by

Xuhong Zhang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2015

Signature redacted

Author ....
Department of Electrical Engineering and Computer Science
January 30, 2015

Signature redacted

Certified by.. ....
Gregory W. Wornell
Sumitomo Professor of Engineering
Thesis Supervisor

Signature redacted

Accepted by ....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

A Sampling Technique Based on LDPC Codes

by

Xuhong Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on January 30, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Given an inference problem, it is common that exact inference algorithms are computationally intractable and one has to resort to approximate inference algorithms. Monte Carlo methods, which rely on repeated sampling of the target distribution to obtain numerical results, is a powerful and popular way to tackle difficult inference problems. In order to use Monte Carlo methods, a good sampling scheme is vital.

This thesis aims to propose a new sampling scheme based on Low Density Parity Check codes and compare it with existing sampling techniques. The proposed sampling scheme works for discrete variables only, but makes no further assumption of the structure of target distribution.

The main idea of the proposed sampling method relies on the concept of *typicality*. By definition, a strong typical sequence with respect to a distribution can closely approximate the distribution. In other words, if we can find a strong typical sequence, the symbols in the sequence can be used as samples from the distribution. According to asymptotic analysis, the set of typical sequences dominates the probability and all typical sequences are roughly equi-probable. Thus samples from the distribution can be obtained by associating each typical sequence with an index, uniformly randomly picking an index, and finding the typical sequence that corresponds to the chosen index. The symbols in that sequence are the desired samples.

To simulate this process in practice, an LDPC code is introduced. Its parity check values are uniformly randomly generated, and can be regarded as the index. Then we look for the most likely sequence that satisfies all the parity checks, and it will be proved that this sequence is a typical one with high probability if the LDPC has appropriate rate. If the most likely sequence found is a typical one, it can be regarded as the one corresponding to the chosen index. In practice, finding the most likely sequence can be computationally intractable. Thus Belief Propagation algorithm is implemented to perform approximate simulation of the sampling process.

The proposed LDPC-based sampling scheme is formally defined first. After proving its correctness under maximum-likelihood simulation, we empirically examine the performance of the scheme on several distributions, namely Markov chain sources, Single loop sources, and 2-Dimensional Ising models. The results show that the proposed scheme can produce good quality samples for the aforementioned distributions.

Thesis Supervisor: Gregory W. Wornell
Title: Sumitomo Professor of Engineering

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Inference-related problems arise in a wide range of fields such as control, communication, machine learning and signal processing. Generally speaking, we are interested in learning about some hidden variable – let it be the codeword sent through a channel, a command issued by a user, or the position of a spacecraft – from some observations related to the hidden variable. We might ask questions like, given the observations, what is the distribution of the hidden variable? Or, what is the most likely value of the hidden variable? However, exact inference is provably hard and computationally intractable unless the underlying problem has very special structure, and therefore we often have to resort to approximate inference algorithms. One of the most powerful and popular approximate inference algorithms is *sampling.*

Consider a random variables with probability distributions for which exact inference is infeasible. A common task in practice is to evaluate some quantities about the variable, e.g. mean, marginals or some parameter of the model. Since probability distribution contains all information about the variable, these quantities of interest can usually be written as expectation of some functions with respect to the distribution. For example, given target distribution $p_{\mathsf{x}}(\cdot)$ and a particular symbol a, the marginal $p_{\mathsf{x}}(a)$ can be written as $\mathbb{E}_{p_{\mathsf{x}}}[\mathbb{1}(\mathbf{x} = a)]$. If samples can be generated efficiently from the distribution, these expectations can then be approximated in a reasonable amount of time. This is the basic idea of *Monte Carlo methods*, a class of computational algorithms that rely on repeated sampling of the target distribution.

The correctness of Monte Carlo methods in the limit of large number of good quality samples is ensured by Law of Large Numbers: given any function $f(\cdot)$, and independent, identically distributed samples $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ..., $\mathbf{x}^{(S)}$ from $p_{\mathbf{x}}(\cdot)$, we have

$$\frac{1}{S}\sum_{i=1}^{S} f(\mathbf{x}^{(i)}) \rightarrow \mathbb{E}_{p_{\mathbf{x}}}[f(\mathbf{x})] \text{ as } S \rightarrow \infty$$

That is, the expectation computed from the samples converges to the true value asymptotically. Thus the problem becomes how to obtain independent good quality samples from a given distribution at practically feasible computational costs. In some situations where random bit generation is expensive, economic utilization of randomness is also important in addition to computational efficiency.

A large number of sampling techniques have been proposed and utilized, including (but not limited to) Rejection Sampling, Importance Sampling, Markov Chain Monte Carlo methods etc. Each has its own strengths and weaknesses, and thus is suitable for different applications. Notice that one should not expect to find a 'perfect' sampling scheme, since no sampling scheme can generate exact and independent samples from all given target distributions in polynomial time. The existence of such a sampling technique would lead to polynomial time algorithms to solve the general inference problem with arbitrary accuracy, which is provably NP-hard [5]. In other words, compromises need to be made somewhere.

Among the commonly used sampling techniques, Importance Sampling and Rejection Sampling both require the availability of an 'easy-to-sample-from' proposal distribution that is close enough to the target distribution, and lack the ability to deal with high-dimensional data. Without a satisfactory proposal distribution, both methods may require exponentially many iterations, as well as exponentially many random bits to obtain the desired results. For Markov Chain Monte Carlo methods, on the other hand, ensuring that the chain explores the distribution well and determining when the algorithm converges are not straight-forward tasks. As a result, both computational cost and usage of randomness can be difficult to quantify.

This thesis aims to devise a new sampling scheme which makes use of Low Density Parity Check codes (LDPC codes). The proposed sampling scheme will only deal with discrete random variables with a finite-size alphabet, but the distribution can be arbitrary otherwise. In fact, since any discrete random variable with finite alphabet size can be represented as a sequence of binary random variables, without loss of generality, we will focus on distribution over sequences of binary variables. In particular, the proposed sampling technique scales well with dimension and thus can deal with high-dimensional variables. It has provably polynomial computational cost, both in time and space, in addition to making efficient use of random bits. Moreover, in contrast to samples produced by Markov Chain Monte Carlo methods, the ones obtained from the new sampling method have good independence properties.

LDPC codes were first introduced in the 1960s by Gallager in his doctoral thesis [4] as a *channel coding* technique. It was largely forgotten for decades, until its rediscovery in the mid 1990s by Mackay and Spielman (independently). Spielman proved that in theory, LDPC codes could approach capacity of the channel with linear decoding complexity as block length goes to infinity [10]; while Mackay showed that in practice LDPC codes with moderate block length could attain near-Shannon-limit performance [9]. Nowadays, LDPC codes have found extensive use in various applications, and have been part of many channel coding standards.

There is a duality between channel coding and source coding. Thus it is not surprising that LDPC codes have also been applied to source coding problems. Examples include [17], which looked at using LDPC codes for lossless data compression and [15], which showed that in lossy compression settings, it is possible to approach the binary rate-distortion bound using LDPC-like codes.

Recently [29] proposed an architecture for compression that has ‘model-free’ encoders, and LDPC code is used as a key component of the system. Basically, the source is described by a factor graph and compression/encoding is accomplished by applying an LDPC code with appropriate rate to the source variables. The decoder uses side information of the source (provided by the source factor graph) together with the compressed content (which can be seen as the output of a hashing function) to recover the original message. The scheme proposed in this thesis is inspired by the architecture in [29], but aims to solve a different task, namely that of obtaining samples from a given probability distribution.

In LDPC code, sending parity check bits in addition to the actual codeword enables reliable transmission of the codeword over a noisy channel. Given enough parity check bits and enough information about the codeword[1], it should not be surprising that one can recover the codeword being sent. The basic idea of the proposed sampling technique is to view the desired sample as an unknown ‘codeword’, apply an LDPC code and use random coin flips to determine the values of ‘parity check bits’. Then rely on both the parity check bits and 'information about the codeword' (the structure of the source distribution) to recover the codeword and a sample is produced. It will be proved that with adequately constructed system, the obtained samples are from the strong typical set with respect to the target distribution, and thus provide nice approximation of the target distribution.

This thesis is organized as follows: Chapter 2 will introduce the necessary background for understanding the LDPC-based sampling scheme, including Monte Carlo methods, factor graphs, message passing algorithms, LDPC codes and Onsager's exact solutions for 2-dimensional Ising model. A formal description of the problem formulation, as well as the sampling scheme will be presented in Chapter 3. Chapter 4 proves the correctness of the scheme assuming Maximum-likelihood simulation,

[1]In the channel coding context, the information being a corrupted version of the codeword

while Chapter 5 discusses the relevant details of implementing the scheme in practice where approximate simulation is performed via Belief Propagation algorithm. The (approximate) simulation results on three different types of target distributions, followed by an analysis of various phenomena observed, are reported in Chapter 6. A range of discussions over more general topics, such as comparisons among different sampling methods and how to choose a suitable one, or alternative ways to generate the code graph, are included in Chapter 7. Finally, Chapter 8 concludes the thesis and points to several directions of possible future work.

Chapter 2

Preliminaries

This chapter introduces some relevant concepts and results that will be used later in the thesis. Section 2.1 provides a brief review of several sampling methods that are commonly used in practice, including Rejection Sampling, Importance Sampling and Markov Chain Monte Carlo methods. Section 2.2 and Section 2.3 contain a description of factor graphs and a high-level introduction of message passing algorithms, respectively. Section 2.4 serves as a short introduction to LDPC Codes. Section 2.5 presents Onsager's exact solution, which analytically computes various quantities, such as entropy and partition function, for 2-dimensional Ising model.

2.1 Sampling Methods

Depending on the application, Monte Carlo methods aim to achieve one or both of the following tasks:

1. Given a target distribution $p_{\mathbf{x}}(\cdot)$, obtain independent samples $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ..., $\mathbf{x}^{(S)}$ from $p_{\mathbf{x}}(\cdot)$.

2. Given a target distribution $p_{\mathbf{x}}(\cdot)$ and a function $f(\cdot)$, estimate $E_{p_{\mathbf{x}}}[f(\mathbf{x})]$.

If the first task is solved, i.e. there exists a sampling method that produces samples that are independent of each other and can approximate the distribution closely, it will be straightforward to tackle task 2:

$$\frac{1}{S}\sum_{s=1}^{S} f(\mathbf{x}^{(s)}) \rightarrow E_{p_{\mathbf{x}}}[f(\mathbf{x})] \text{ as } S \rightarrow \infty,$$

according to Law of Large Numbers. On the other hand, some applications only require estimation of some expectation and task 2 may be accomplished without worrying about task 1.

Recall from previous discussion that one should not expect to find a 'perfect' sampling scheme. Different sampling techniques will make different compromises. In this section, several sampling techniques that are commonly used in practice are briefly described. Since the results are well known, relevant proofs are omitted here. A discussion on their relative pros and cons, and a comparison with the LDPC-based sampling technique proposed in this thesis, will be deferred to Section 7.1.

2.1.1 Rejection Sampling

Rejection sampling assumes the availability of a proposal density $q_{\mathbf{x}}(\cdot)$, which can be efficiently sampled from[1]. Assume

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{p^*_{\mathbf{x}}(\mathbf{x})}{Z_p} \text{ and } q_{\mathbf{x}}(\mathbf{x}) = \frac{q^*_{\mathbf{x}}(\mathbf{x})}{Z_q}$$

where Z_p and Z_q are constants, and $p^*_{\mathbf{x}}(\mathbf{x})$ and $q^*_{\mathbf{x}}(\mathbf{x})$ are easy to evaluate. Furthermore, assume there exists a constant $c > 0$, such that for any $\mathbf{x}$, $cq^*_{\mathbf{x}}(\mathbf{x}) \geq p^*_{\mathbf{x}}(\mathbf{x})$. Samples from $p_{\mathbf{x}}(\cdot)$ will be generated in 2 steps:

1. Generate $\mathbf{x}$ from $q_{\mathbf{x}}(\cdot)$, and compute $cq^*_{\mathbf{x}}(\mathbf{x})$.
2. Generate a sample u from the uniform distribution on $[0,\, cq^*_{\mathbf{x}}(\mathbf{x})]$. Accept $\mathbf{x}$ as a sample of $p_{\mathbf{x}}(\cdot)$ if $u < p^*_{\mathbf{x}}(\mathbf{x})$, otherwise reject.

It can be proved that the above procedure results in independent samples from $p_{\mathbf{x}}(\cdot)$.

[1] A naive examples of 'easy-to-sample-from' distributions is the uniform distribution.

2.1.2 Importance Sampling

Importance sampling is a method to accomplish task 2 without actually generating samples from $p_{\mathbf{x}}(\cdot)$. Importance sampling also assumes a proposal distribution $q_{\mathbf{x}}(\cdot)$ that can be sampled from efficiently.

Let $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ..., $\mathbf{x}^{(S)}$ be independent samples from distribution $q_{\mathbf{x}}(\cdot)$. $E_{p_{\mathbf{x}}}[f(\mathbf{x})]$ will be estimated by:

$$\frac{\sum\limits_{s=1}^{S} w^{(s)} f(\mathbf{x}^{(s)})}{\sum\limits_{s=1}^{S} w^{(s)}}, \text{ where } w^{(s)} = \frac{p^*_{\mathbf{x}}(\mathbf{x}^{(s)})}{q^*_{\mathbf{x}}(\mathbf{x}^{(s)})} \text{ is known as the weight.}$$

It can be proved that as $S \to \infty$,

$$\frac{\sum\limits_{s=1}^{S} w^{(s)} f(\mathbf{x}^{(s)})}{\sum\limits_{s=1}^{S} w^{(s)}} \to E_{p_{\mathbf{x}}}[f(\mathbf{x})]$$

The intuition here is that the weights $w^{(1)}$, ..., $w^{(S)}$ are used to compensate for the difference between the target distribution and the proposal distribution, by regarding some samples as more important than others.

2.1.3 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo methods generate samples from target distributions $p_{\mathbf{x}}(\cdot)$ by constructing a Markov chain whose stationary distribution happens to be $p_{\mathbf{x}}(\cdot)$. There are many algorithms that fall in this category, and two most famous variants are briefly described here.

Metropolis-Hastings algorithm

Metropolis-Hastings algorithm again makes use of a proposal distribution $q(\cdot;\cdot)$ where $q(\mathbf{x}';\mathbf{x})$ denotes the probability of proposing new state $\mathbf{x}'$ given that current state is

$\mathbf{x}$. The proposal distribution can be chosen arbitrarily, as long as $q(\mathbf{x}, \mathbf{x}) > 0$ for any $\mathbf{x}$, and from any state $\mathbf{x}$, any other state $\mathbf{x}'$ can be reached in a finite number of steps. Basically, these regularity conditions are sufficient conditions for the corresponding Markov chain to be irreducible and aperiodic. The algorithm proceeds as follows:

1. Start in some arbitrary initial state $\mathbf{x}^{(0)}$.
2. At time step $t+1$, propose a new state $\mathbf{x}'$ according to distribution $q(\cdot; \mathbf{x}^{(t)})$.
3. Compute $r = \dfrac{p^*_{\mathbf{x}}(\mathbf{x}')q(\mathbf{x}^{(t)}; \mathbf{x}')}{p^*_{\mathbf{x}}(\mathbf{x}^{(t)})q(\mathbf{x}'; \mathbf{x}^{(t)})}$.
4. If $r \geq 1$, always accept the new state by setting $\mathbf{x}^{(t+1)} = \mathbf{x}'$.
   If $r < 1$, accept and set $\mathbf{x}^{(t+1)} = \mathbf{x}'$ with probability r, otherwise reject and set $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)}$ with probability $1 - r$.
5. Increase t and go back to step 2.

It can be shown that the Markov Chain constructed this way has unique stationary distribution $p_{\mathbf{x}}(\cdot)$. Thus in theory, as the number of iterations goes to infinity, the empirical distribution of the obtained samples converges to $p_{\mathbf{x}}(\cdot)$.

Gibbs Sampling

Gibbs Sampling algorithm can be viewed as a special case of Metropolis-Hastings algorithm where the proposal distribution is chosen in a particular way:

1. Start in some initial state $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$.
2. At time step $t+1$, sample each variable in turn:

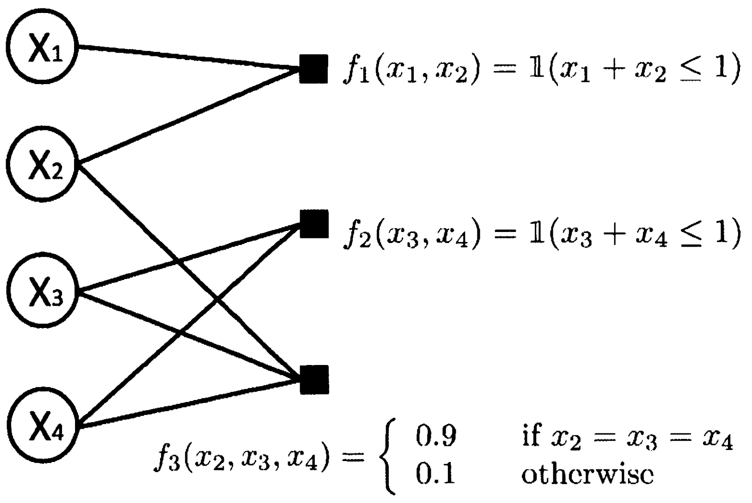
$$\begin{aligned} x_1^{(t+1)} &\sim p(x_1 | x_2^{(t)}, x_3^{(t)}, ..., x_n^{(t)}) \\ x_2^{(t+1)} &\sim p(x_2 | x_1^{(t+1)}, x_3^{(t)}, ..., x_n^{(t)}) \\ &\cdots \\ x_n^{(t+1)} &\sim p(x_n | x_1^{(t+1)}, x_2^{(t+1)}, ..., x_{n-1}^{(t+1)}) \end{aligned}$$

3. The state at time $t+1$ is then $\mathbf{x}^{(t+1)} = (x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_n^{(t+1)})$. Increase t and go back to step 2.

Notice that Gibbs sampler always accepts the proposed state in each iteration. It is assumed that the conditional distributions are easy to sample from. Since Gibbs sampling is a special case of Metropolis-Hastings algorithm, it is also provably correct in the limit of infinitely many iterations.

2.2 Factor Graphs

Graphical models are useful tools that capture the structure of the variables of interest. A graph contains nodes and edges, where nodes represent variables/functions and edges indicate dependence between the nodes they connect to. This thesis focuses on a particular type of graphical models, namely factor graphs.



X1
X2
X3
X4
$f_1(x_1, x_2) = \mathbb{1}(x_1 + x_2 \leq 1)$
$f_2(x_3, x_4) = \mathbb{1}(x_3 + x_4 \leq 1)$
$f_3(x_2, x_3, x_4) = \begin{cases} 0.9 & \text{if } x_2 = x_3 = x_4 \\ 0.1 & \text{otherwise} \end{cases}$

Figure 2-1: An example factor graph.

A factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ is a bipartite graph, which contains a set of variable nodes $\mathcal{V}$, a set of factor nodes $\mathcal{F}$, and a set of undirected edges $\mathcal{E}$. Each edge connects a variable node and a factor node. Let the variable nodes represent

$\mathbf{x} = (x_1, x_2, ..., x_n)$. j^{th} factor node is associated with a non-negative function $f_j(\mathbf{x}_{f_j})$, where $j \in \{1, 2, ..., K\}$, $K = |\mathcal{F}|$ and $\mathbf{x}_{f_j}$ is the subset of variable nodes that j^{th} factor node connects to. Then the joint probability distribution associated with the factor graph is

$$p_{\mathbf{x}}(x_1, x_2, ..., x_N) \propto \prod_{j=1}^{K} f_j(\mathbf{x}_{f_j})$$

A simple example of a factor graph can be found in Figure 2-1.

2.3 Message Passing Algorithms

Nowadays, many applications are dealing with systems of ever-increasing size. Examples include the Internet, sensor networks or multi-core architectures. Such systems are usually modelled as graphs, where the entities (i.e. computers, sensors and cores respectively in the aforementioned examples) are viewed as nodes, and two entities that can communicate with each other are connected by an edge. The tasks in the applications are usually global ones, e.g. finding some globally optimal configuration. Yet the computational cost to accomplish these tasks via centralized computing, where one processor takes care of all the computation, is usually exponential in the total number of nodes, and thus infeasible for large systems. Instead, one resorts to distributed computing.

In distributed computing, each node in the graph is viewed as a processor and carries out computation locally. Obviously, the local processors need to communicate with each other to achieve/approximately achieve the global objective. Message passing algorithms are a family of algorithms that perform such communication among nodes using 'messages'.

In general, message passing algorithms are iterative algorithms. In each iteration, each node receives messages from neighbours (i.e. other nodes that it directly connects to), performs local computation, and sends out new messages to its neighbours.

After t iterations, each node has information from its size-t neighbourhood. Depending on what messages are communicated among nodes, there are many types of message passing algorithms. In this thesis, we focus on a particular one called Belief Propagation algorithm. Its general derivation can be found in many standard textbooks (e.g. [6], [27]) and thus is omitted here. Details of the Belief Propagation algorithm specialized to the proposed LDPC-based sampling scheme will be discussed in Section 5.2.

2.4 LDPC Codes

This section introduces LDPC codes and some related definitions.

Only binary LDPC codes will be considered here, i.e. the codebook $\mathcal{C}$ is a subset of $\{0,1\}^N$. N is called the block length of the code. LDPC codes are linear codes, that is, $\mathcal{C}$ can be written as

$$\mathcal{C} = \{\mathbf{x} \in \{0,1\}^N : \mathbb{H}\mathbf{x} = \mathbf{0}\},$$

where parity check matrix $\mathbb{H}$ is a $K \times N (K \leq N)$ matrix with binary elements $\mathbb{H}_{ij} \in \{0,1\}$, and $\mathbb{H}\mathbf{x}$ is computed in modulo 2 sense.

rank($\mathbb{H}$) denotes the number of linearly independent rows in $\mathbb{H}$ and

$$R = \frac{\text{rank}(\mathbb{H})}{N}$$

is defined as the rate of the code. Clearly, $R \leq \frac{K}{N}$.

$\mathbb{H}$ will be a sparse matrix, i.e. the number of 0's in $\mathbb{H}$ is much larger than the number of 1's, hence the name 'low density'. To be more precise, the number of 1's grows only linearly with block length N.

Each LDPC code can be described by its parity check matrix $\mathbb{H}$ and each $\mathbb{H}$ in turn can be associated with a factor graph. Create a variable node for each column of $\mathbb{H}$ and create a factor node for each row of $\mathbb{H}$. As each of them corresponds to a parity check, the factors nodes in the factor graph for an LDPC code are also referred to as 'check nodes'. Variable node j is connected to check node i if and only if $\mathbb{H}_{ij} = 1$. The function associated with check node i is

$$\mathbb{1}(x_{j_1} + x_{j_2} + \ldots + x_{j_l} = 0 (\text{mod } 2))$$

assuming there are l variables connected to check node i, whose corresponding indices are j_1, j_2, ..., j_l.

In fact, we will be using a slightly different definition of LDPC codes:

$$\mathcal{C}' = \{\mathbf{x} \in \{0,1\}^N : \mathbb{H}\mathbf{x} = \mathbf{z} (\text{mod } 2)\}$$

where $\mathbf{z} \in \{0,1\}^K$ is a fixed binary sequence. Notice that for any $\mathbf{x}_0 \in \mathcal{C}'$, and for any $\mathbf{x} \in \mathcal{C}$, $\mathbf{x}_0 + \mathbf{x} \in \mathcal{C}'$. The function associated with check node i now becomes

$$\mathbb{1}(x_{j_1} + x_{j_2} + \ldots + x_{j_l} = z_i (\text{mod } 2))$$

It can be proved that most of the properties of $\mathcal{C}$ are satisfied by $\mathcal{C}'$ as well.

An important property of an LDPC code is its degree distribution. The degree of a node (variable or check node) is defined as the number of other nodes that it is connected to. Degree distribution, as the name suggests, is the distribution of the degree of variables nodes and check nodes. More concretely, given the factor graph associated with an LDPC code, denote by λ_d and ρ_d the fractions of variable nodes and check nodes, respectively, of degree d (d = 0,1,2,...). The pair (Λ, P) is called the degree distribution of the LDPC, where $\Lambda = \{\lambda_d\}_{d=0}^{\infty}$ and $P = \{\rho_d\}_{d=0}^{\infty}$. A practical representation of the degree distribution is provided by the generating functions

$$\Lambda(x) = \sum_{d\geq 0} \lambda_d x^d \text{ and } P(x) = \sum_{d\geq 0} \rho_d x^d$$

If both the variable node degree and the check node degree are constants (i.e. there exist integers d_v and d_c such that $\lambda_{d_v} = 1$ and $\rho_{d_c} = 1$), the LDPC is said to be regular. Otherwise the LDPC is called irregular. Irregular LDPC codes are more flexible than regular ones and shown to have better performance when used as channel codes [11]. However, finding the optimal degree distribution is a non-trivial task. This thesis will stick to regular LDPC codes. Section 5.1 will describe how the LDPC codes used in the simulations are generated, and Section 6.2.2 will discuss how degree distribution of the LDPC codes can impact the performance of the proposed sampling scheme.

2.5 Onsager's Exact Solution for 2D Ising Model

One of the models the proposed sampling scheme will be tested on is 2-Dimensional homogeneous Ising model, a schematic of which is shown in Figure 2-2.

Denote the set of edges in the 2D Ising model as $\mathcal{E}$. The model is defined as

$$p_{\mathsf{x}}(\mathbf{x}) = \frac{1}{Z} \exp(-\beta J \sum_{(i,j)\in\mathcal{E}} x_i x_j)$$

where $Z = \sum_{\mathbf{x}} \exp(-\beta J \sum_{(i,j)\in\mathcal{E}} x_i x_j)$ is called the partition function and $x_i \in \{-1, 1\}$ represents the spin at location i. β and J have physical meanings, but for our purposes, they are just constants.

Chapter 6 will need to calculate the entropy and partition function of the 2-Dimensional Ising model. Fortunately, both quantities can be analytically computed using Onsager's Exact Solution [1], which is presented in this section. The relevant proofs are very involved and beyond the scope of this thesis, and thus are omitted here.

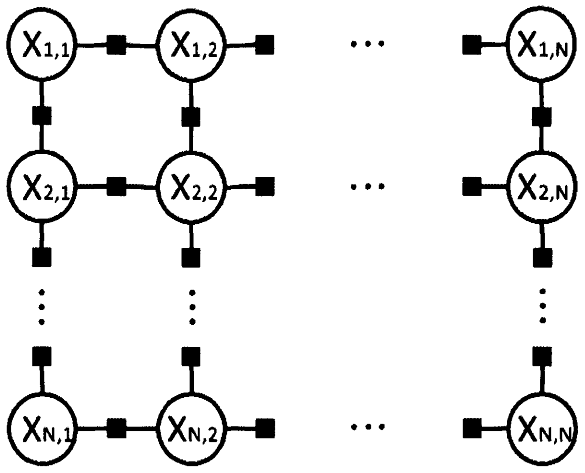


Figure 2-2: Schematic of Ising model.

It should be noted that Onsager's Exact Solution assumes periodic boundary conditions, i.e. a factor node is attached to the first and last variable in each row and column as well. The graph is a toroidal grid as shown in Figure 2-3, and all variables in the grid have the same number of neighbours. As the size of Ising model increases, the assumption about boundary conditions should have vanishing impact.

Free energy of the 2-Dimensional Ising model is defined as

$$F(\beta) \triangleq -\frac{1}{\beta}\ln[Z(\beta)] = -\frac{1}{\beta}\ln[\sum_{\mathbf{x}} \exp(-\beta J \sum_{(i,j)\in\mathcal{E}} x_i x_j)]$$

Free energy density is free energy per node, as the size of the model goes to infinity

$$f(\beta) = \lim_{N\to\infty} \frac{F_N(\beta)}{N^2}$$

where N is the number of nodes in each row/each column.

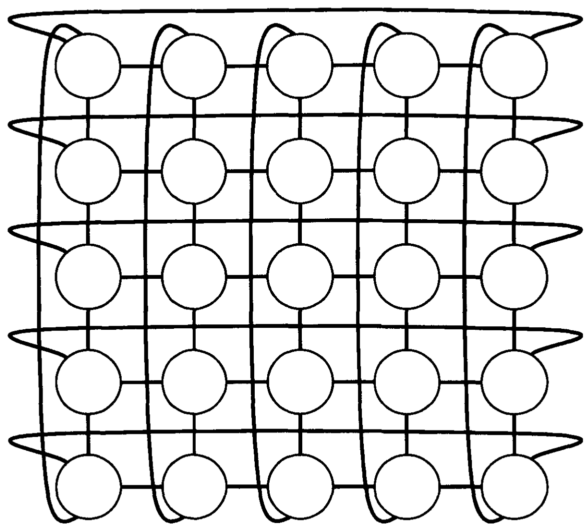


Figure 2-3: 2D Ising model with periodic boundary condition.

Entropy of the Ising model is defined as[2]

$$S(\beta) \triangleq -\sum_{\mathbf{x}} p_{\mathbf{x}}(\mathbf{x}) \ln(p_{\mathbf{x}}(\mathbf{x})) = \beta^2 \frac{\partial F(\beta)}{\partial \beta}$$

Similarly entropy density is defined as $s(\beta) = \lim_{N\to\infty} \frac{S_N(\beta)}{N^2}$. Thus

$$s(\beta) = \beta^2 \frac{\partial f(\beta)}{\partial \beta}$$

Onsager's Exact Solution states that

$$f(\beta) = -\frac{1}{\beta}\ln(2) - \frac{1}{8\beta\pi^2} \int_0^{2\pi}\int_0^{2\pi} \ln[a^2 - b\cos(\theta_1) - b\cos(\theta_2)] d\theta_1 d\theta_2$$

where $a = \cosh(2\beta J)$ and $b = \sinh(2\beta J)$

For large N, by definition of $F(\beta)$ and $f(\beta)$, partition function can be computed from

$$Z \sim \exp(-\beta f(\beta) N^2)$$

[2]The usual definition of entropy uses $\log_2$ instead of ln. But for clarity of presentation we use ln here. The answer only differs by a constant factor ln(2).

Plug in $s(\beta) = \beta^2 \frac{\partial f(\beta)}{\partial \beta}$, entropy density can be computed as

$$s(\beta) = \ln(2) + \frac{1}{8\pi^2} \int_0^{2\pi}\int_0^{2\pi} \ln[a^2 - b(\cos(\theta_1) + \cos(\theta_2))] d\theta_1 d\theta_2$$
$$- \frac{\beta J}{4\pi^2} \int_0^{2\pi}\int_0^{2\pi} \frac{2ab - a(\cos(\theta_1) + \cos(\theta_2))}{a^2 - b(\cos(\theta_1) + \cos(\theta_2))} d\theta_1 d\theta_2$$

Notice β and J always appear together in the expressions, thus without loss of generality, one can assume $\beta = 1$. Also, notice that neither free energy nor entropy will change if all -1's in the Ising model are replaced by 0's.

It is also common to describe the Ising model using factors and flipping probability p (i.e. the probability of neighbouring variables taking different values). Then the function associated with each factor in Figure 2-2 takes the form:

$$\psi_{x_i,x_j} = \begin{cases} p & \text{if } x_i \neq x_j \\ 1-p & \text{if } x_i = x_j \end{cases}$$

Now let us relate J to p:

$$\frac{e^{-J}}{e^J} = \frac{1-p}{p} \Rightarrow J = \frac{1}{2}\ln(\frac{p}{1-p})$$

Thus $a = \cosh(2J) = \frac{2p^2-2p+1}{2p-2p^2}$ and $b = \sinh(2J) = \frac{2p-1}{2p-2p^2}$.

Chapter 3

LDPC-Based Sampling Scheme

In this chapter, the LDPC-based sampling scheme is formally introduced. To begin with, Section 3.1 defines the sampling problem and lists the goals of the proposed scheme. Next, Section 3.2 aims to provide a description of the basic idea of the scheme, as well as an intuitive argument as why the scheme should work. Finally, a precise statement of the scheme is presented in Section 3.3.

3.1 Problem Description

The goal is to sample from distributions over discrete variables defined over a finite alphabet. As already mentioned in Chapter 1, such a distribution can always be converted into a distribution over a sequence of binary random variables. Thus without loss of generality, the rest of the thesis will assume $\mathbf{x} = (x_1, x_2, ..., x_N) \in \{0,1\}^N$. Moreover, the target distribution $p_{\mathbf{x}}(\cdot)$ is described by a factor graph.

Ideally, independent samples $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(S)}$ from $p_{\mathbf{x}}(\cdot)$ are desired. Since the alphabet size is finite, in theory $p_{\mathbf{x}}(\mathbf{x})$ for every $\mathbf{x} \in \{0,1\}^N$ can be computed. Given probabilities of all possible configurations, one can partition [0,1] into small intervals, each corresponds to one configuration and the length of the interval equals to the probability of that configuration. Then independent samples can be obtained by simply generating a random number from the uniform distribution on [0,1]. However, the amount of computation needed for this naive approach will be exponential in N[1].

The proposed LDPC-based sampling scheme, which will be described in detail in Section 3.3, claims to produce samples $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(S)}$:

1. in amount of time that is linear with respect to the number of samples S and polynomial with respect to the size of the source N;
2. without requiring an 'easy-to-sample-from' proposal distribution;
3. although the samples produced are not necessarily independent, for any pre-specified accuracy, the empirical distribution formed by $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(S)})$ can approximate $p_{\mathbf{x}}(\cdot)$ with that accuracy given large enough S.

3.2 Basic Idea and Why It Should Work

Before presenting the details of the proposed sampling scheme, it is useful to describe the basic idea and provide some intuition as why this scheme should work.

Given a target distribution $p_{\mathbf{x}}(\cdot)$ over alphabet $\mathcal{X}$, a δ-strong typical set of size t with respect to $p_{\mathbf{x}}(\cdot)$ is defined as:

$$T_{\delta}^{(t)} = \{\mathbf{y} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)}) \in \mathcal{X}^t : \forall a \in \mathcal{X}, |\frac{N(a|\mathbf{y})}{t} - p_{\mathbf{x}}(a)| \leq \delta\}$$

where $N(a|\mathbf{y})$ denotes the number of occurrences of symbol a in sequence $\mathbf{y}$. In other words, if $\mathbf{y}$ is a strong typical sequence for $p_{\mathbf{x}}(\cdot)$, the empirical distribution formed by $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)}$ is a nice approximation of $p_{\mathbf{x}}(\cdot)$ and the approximation precision is controlled by δ. On the other hand, typicality analysis shows that the sequences in the strong typical set will dominate the probability when k is large enough and they each have roughly the same probability in asymptotic sense. Thus good quality samples of $p_{\mathbf{x}}(\cdot)$ can be obtained by associating with each typical sequence an index,

[1]Recall that in general, computing the normalization constant/partition function is NP-hard for distributions described by a factor graph.

uniformly randomly picking an index, and then finding the sequences corresponding to that index.

The challenge is how to simulate this process using a scheme that can be implemented in practice. Inspired by [29], we consider attaching an LDPC code to the variables. The LDPC code essentially serves as a hash function and partition all sequences into groups that will be referred to as 'bins'. The results in [29] shows that when the LDPC code has appropriate rate, the hash function is a very good one for the typical sequences: almost all bins contain one and only one typical sequence. Therefore given hash index and side information provided by the factor graph of the distribution, one can expect to find the corresponding typical sequence with high probability. In particular, the analysis in Chapter 4 assumes that the most likely sequence in the chosen bin is picked and it will be proved to be a typical one with high probability; the experiments in Chapters 5 and 6 uses Belief Propagation algorithm to search for the typical sequence and obtains good samples empirically.

However, the sampling scenario considered in this thesis has its own feature: in the compression problem in [29], the sequence to be compressed is produced according to the source distribution and thus is a typical one with high probability; while in the sampling problem, typicality is not naturally in the system: the factor graph describing the target distribution has to be repeated to impose the notion of typicality.

Notice all information about the target distribution is (implicitly) available once the factor graph of target distribution is given. For example, algorithms such as Belief Propagation or Max-Product can be run to estimate the marginal distributions or maximizing configuration. However, these algorithms are deterministic. In order to sample from $p_{\mathsf{x}}(\cdot)$, randomness is needed. In the proposed scheme, randomness is introduced by 'uniformly randomly picking an index'. Introducing the right amount of randomness is then equivalent to picking an appropriate rate for the LDPC code. The amount of randomness should be enough such that rich samples can be produced, but only to the extent where the structure in the target distribution is still respected.

3.3 Sampling Scheme Based on LDPC Codes

The proposed sampling scheme will be defined on a *combined source-code graph* (which will be referred to as *combined graph* for concise presentation). Thus the construction of the combined graph will be described first.

Given a target distribution $p_{\mathsf{x}}(\cdot)$, denote the corresponding factor graph as $\mathcal{S}$. The *source graph* $\mathcal{G}_S$ will contain t identical copies of $\mathcal{S}$. The copies are independent of each other, i.e. there is no edge connecting any two nodes from different copies. As mentioned in Section 3.2, the factor graph of $p_{\mathsf{x}}(\cdot)$ is repeated for typicality arguments to work. Further justification for this and what value of t should be used will be discussed in Chapter 4.

Given the source graph $\mathcal{G}_S$, attach to it an LDPC code, which has Nt variable nodes and with appropriate rate[2]. The factor graph describing the chosen LDPC code is referred to as *code graph* $\mathcal{G}_C$. In what follows, the factor nodes in code graph $\mathcal{G}_C$ will be called 'check nodes', and the term 'factor nodes' will be reserved for factor nodes in the source graph $\mathcal{G}_S$.

Notice that the variable nodes are shared by source graph $\mathcal{G}_S$ and code graph $\mathcal{G}_C$, and form a boundary between the two. $\mathcal{G}_S$ and $\mathcal{G}_C$ together are called combined graph, denoted by $\mathcal{G}$. An example of combined graph is shown in Figure 3-1. Now we are ready to present the LDPC-based sampling scheme:

1. Given the factor graph for $p_{\mathsf{x}}(\cdot)$, construct the corresponding combined graph. Denote the number of check nodes in the combined graph as K, the number of variables in the target distribution as N, and the number of repetitions of the factor graph as t.

[2]Discussion on how to choose the rate will be deferred to Section 5.5.

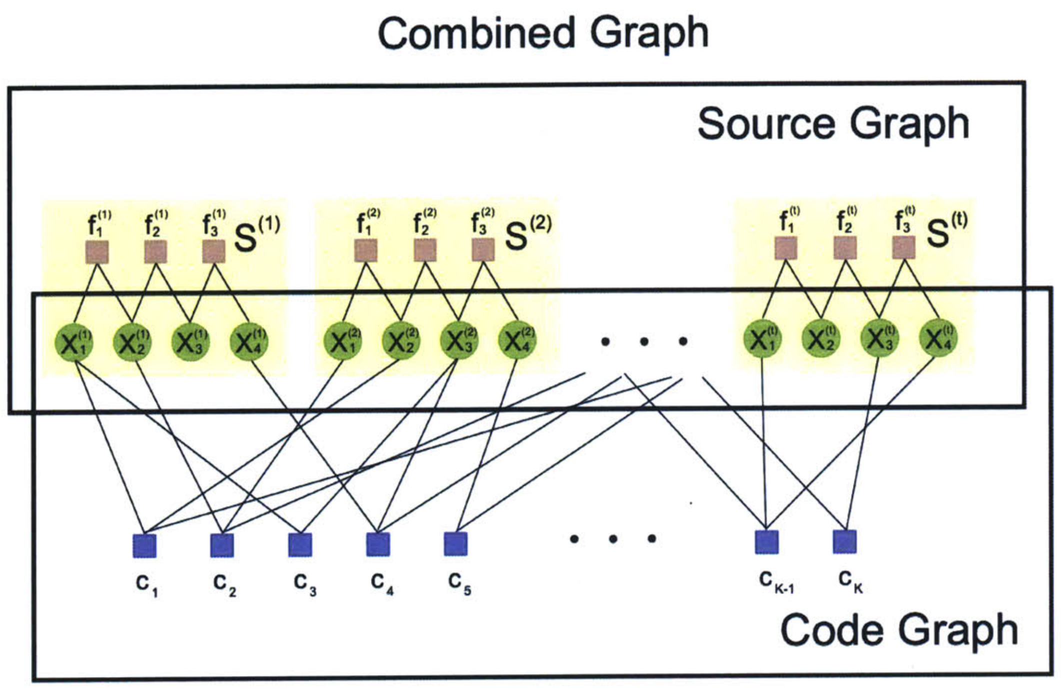

Combined Graph
Source Graph
Code Graph
$S^{(1)}$
$S^{(2)}$
$S^{(t)}$
$f_1^{(1)}$ $f_2^{(1)}$ $f_3^{(1)}$
$f_1^{(2)}$ $f_2^{(2)}$ $f_3^{(2)}$
$f_1^{(t)}$ $f_2^{(t)}$ $f_3^{(t)}$
$X_1^{(1)}$ $X_2^{(1)}$ $X_3^{(1)}$ $X_4^{(1)}$
$X_1^{(2)}$ $X_2^{(2)}$ $X_3^{(2)}$ $X_4^{(2)}$
$X_1^{(t)}$ $X_2^{(t)}$ $X_3^{(t)}$ $X_4^{(t)}$
c_1 c_2 c_3 c_4 c_5 c_{K-1} c_K

Figure 3-1: The schematic of an example combined graph.

2. Flip K independent fair coins and use the results ('head' as 1 and 'tail' as 0) as parity values at the check nodes.

3. Find $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)}) \in \{0,1\}^{Nt}$ that maximizes $\prod_{i=1}^{t} p_{\mathbf{x}}(\mathbf{x}^{(i)})$ while keeping the K parity checks satisfied. In other words, find the codeword of the attached LDPC code that has the highest probability under $p_{\mathbf{x}}(\cdot)$.

In practice, finding the Maximum-likelihood solution in step 3 will be NP-hard and Belief Propagation algorithms are used instead to provide an approximate solution.

Now that the LDPC-based sampling algorithm has been formally described, following chapters move on to theoretically justify the correctness of the proposed scheme under Maximum-likelihood simulation (Chapter 4) and empirically examine the performance of the scheme under approximate simulation (Chapters 5 and 6).

Chapter 4

Maximum-Likelihood Simulation

In this chapter, performance of the proposed LDPC-based sampling scheme is analyzed assuming maximum-likelihood simulation is possible, i.e. one can find the most likely sequence that satisfies all parity checks. In practice, maximum-likelihood simulation will be computationally intractable and approximate simulation will be implemented via Belief Propagation algorithm. Yet it is useful to understand the behavior of the scheme in the maximum-likelihood simulation case, which provides an upper bound for the performance of the scheme.

Section 4.1 will first prove the correctness of the proposed scheme assuming maximum-likelihood simulation. That is, with maximum-likelihood simulation and large enough t (number of repetitions of the source), the proposed sampling scheme will provide samples that approximate the target distribution arbitrarily well, with probability arbitrarily close to 1. Clearly, smaller t is desirable since the computational cost will grow with t. Section 4.2 discusses how many repetitions are required, i.e. how large t should be, as a function of the target distribution and the desired accuracy.

4.1 Correctness Under Maximum-Likelihood Simulation

Section 3.2 argues that if $\mathbf{y} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)})$ is in the δ-strong typical set defined with respect to target distribution $p_{\mathbf{x}}(\cdot)$, by definition, the empirical distribution computed from $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)}$ will be a good approximation of $p_{\mathbf{x}}(\cdot)$ in the sense that

$$\forall a \in \mathcal{X}, |\frac{N(a|\mathbf{y})}{t} - p_{\mathbf{x}}(a)| \leq \delta$$

where $\mathcal{X}$ is the alphabet of the target distribution and $N(a|\mathbf{y})$ is the number of times symbol a appears in $\mathbf{y} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)})$.

Thus in order to produce good quality samples for $p_{\mathbf{x}}(\cdot)$, the goal is to find typical sequences. However, maximum-likelihood simulation of the proposed sampling scheme, as described in Section 3.3, looks for a 'most likely' sequence that satisfies all parity checks. The main result of the current section proves that when the LDPC code is generated with appropriate rate, the 'most likely' sequence found by maximum-likelihood simulation will be a typical sequence with high probability.

Before a precise statement of the result can be made, it is necessary to introduce a few definitions first.

Definition 1. *Given a target distribution $p_{\mathbf{y}}(\cdot)$ and a parity check matrix $\boldsymbol{A} \in \{0,1\}^{k \times n}$, an **ML-sequence set** M is defined as the set of all sequences that has the largest probability among sequences with the same parity check values:*

$$M \triangleq \{\mathbf{y} \in \{0,1\}^n : \exists \mathbf{z} \in \{0,1\}^k s.t. \boldsymbol{A}\mathbf{y} = \mathbf{z} \textit{ and } \forall \mathbf{y}' \textit{ where } \boldsymbol{A}\mathbf{y}' = \mathbf{z}, p_{\mathbf{y}}(\mathbf{y}) \geq p_{\mathbf{y}}(\mathbf{y}')\}$$

Technically, it is possible that there are several sequences that are most likely (they all have the same probability). In those situations, one of them can be arbitrarily but deterministically chosen to be in set M to make sure $|M| \leq 2^k$.

Definition 2. *A model that defines a probability distribution over string* $\mathbf{y}$ *of any length* n, $p_{\mathbf{y}}(\cdot;n)$, *is said to have* ***mean entropy*** $H_{\mathbf{y}}$ *if for any* $\epsilon > 0$, *for any* $\delta > 0$, *there exists* n^* *such that for all* $n > n^*$*:*

$$\mathbb{P}(|\frac{1}{n}log(p_{\mathbf{y}}(\mathbf{y};n)) + H_{\mathbf{y}}| > \delta) < \epsilon$$

Definition 3. *Given target distribution* $p_{\mathbf{y}}(\cdot)$, *a* $(k, \lambda, w, \epsilon)$ ***satisfactory matrix A*** *for* $p_{\mathbf{y}}(\cdot)$ *is a matrix with k row, n columns* $(n \geq \lambda k)$ *with weight w or less per column, and* $p_{\mathbf{y}}(M) \geq 1 - \epsilon$, *where M is the ML-sequence set defined with respect to* $p_{\mathbf{y}}(\cdot)$.

The main result of this section is summarized in Theorem 1:

Theorem 1. *Given any target distribution* $p_{\mathbf{x}}(\cdot)$ *over* $\{0,1\}^N$ *with mean entropy* $H_{\mathbf{x}}$, *for any* $\epsilon > 0$, $\delta > 0$, *there exists large enough t, such that there exists an LDPC code with parity check matrix* $\mathbb{H} \in \{0,1\}^{K \times Nt}$, *where* $K \sim t(H(p_{\mathbf{x}}) - \eta)$, *for which maximum-likelihood simulation will produce a sequence in* $T_\delta^{(t)}$ *with probability at least 1 - ϵ. Here* $T_\delta^{(t)}$ *is the δ-strong typical set defined for distribution* $p_{\mathbf{x}}(\cdot)$, $H(p_{\mathbf{x}}) = -\sum_{a \in \mathcal{X}} p_{\mathbf{x}}(a)\log(p_{\mathbf{x}}(a)) \sim NH_{\mathbf{x}}$, *and* $\eta = -\delta \sum_{a \in \{0,1\}^N} \log(p_{\mathbf{x}}(a))$. *In fact, the result holds for almost any reasonably well constructed LDPC code*[1].

The theorem contains several important messages:

1. The 'appropriate' rate of the LDPC code is

$$\frac{K}{Nt} \sim \frac{t(H(p_{\mathbf{x}}) - \eta)}{Nt} = \frac{H(p_{\mathbf{x}}) - \eta}{N} \sim \frac{H(p_{\mathbf{x}})}{N} \sim H_{\mathbf{x}}$$

   which is the mean entropy of the target distribution.

2. With large enough t, the 'most likely' sequence satisfying all parity checks found by maximum-likelihood simulation is a typical sequence with probability arbitrarily close to 1 (controlled by constant ϵ). If it is typical, the obtained sequence can be viewed as concatenation of t samples, and the samples approximate target distribution $p_{\mathbf{x}}(\cdot)$ with accuracy controlled by constant δ.

[1]Section 5.1 discusses how to construct an LDPC code.

Theorem 1 can be broken into three steps:

1. The parity check matrix of the LDPC code $\mathbb{H}$ partitions the space $\{0,1\}^{Nt}$ into 2^K bins, each containin the same number of Nt-bit sequences and corresponds to a different parity check value $\mathbf{z} \in \{0,1\}^K$.
2. Only a vanishingly small portion of all bins have a sequence that has higher probability than a typical sequence.
3. The typical sequences spread across the bins nicely, i.e. almost all bins contain a typical sequence.

Since the proposed sampling scheme is basically uniformly picking a $\mathbf{z} \in \{0,1\}^K$ and then finding the most likely $\mathbf{y} \in \{0,1\}^{Nt}$ such that $\mathbb{H}\mathbf{y} = \mathbf{z}(\text{mod } 2)$, it is clear that if all three arguments are true, Theorem 1 follows directly. These three steps will be established in the lemmas below:

Lemma 1. *Let $\boldsymbol{A} \in \{0,1\}^{k\times n}$ be full rank and $k < n$, then for any $\mathbf{z} \in \{0,1\}^k$, there are 2^{n-k} sequences $\mathbf{y} \in \{0,1\}^n$ that satisfy $\boldsymbol{A}\mathbf{y} = \mathbf{z}(\text{mod } 2)$.*

Proof. Since $\mathbf{A}$ is full rank, i.e. $\text{rank}(\mathbf{A}) = k$, according to Rank-Nullity Theorem, the dimension of the kernel of $\mathbf{A}$ (i.e. the set of $\mathbf{y} \in \{0,1\}^n$ such that $\mathbf{Ay} = \mathbf{0}$ (mod 2)) is $n-k$. In other words, there are 2^{n-k} binary sequences of size n for which $\mathbf{Ay} = \mathbf{0}$ (mod 2) holds. Clearly, the all-zero sequence is in the kernel.

Given some $\mathbf{z}$, if $\mathbf{y}_0 \in \{0,1\}^n$ satisfies $\mathbf{Ay}_0 = \mathbf{z}(\text{mod } 2)$, we have $\mathbf{A}(\mathbf{y}_0 + \mathbf{u}) = \mathbf{z}$ if and only if $\mathbf{u}$ is in the kernel of $\mathbf{A}$. Thus if such a $\mathbf{y}_0$ exists, there will be exactly 2^{n-k} binary sequences $\mathbf{y}$ that satisfies $\mathbf{Ay} = \mathbf{z}(\text{mod } 2)$. Since $\mathbf{A}$ is full rank, $\mathbf{Ay}$ can achieve any sequence in $\{0,1\}^k$ as $\mathbf{y}$ sweeps the whole space of $\{0,1\}^n$, which guarantees the existence of such an $\mathbf{y}_0$. Thus for any $\mathbf{z} \in \{0,1\}^k$, there are 2^{n-k} sequences $\mathbf{y} \in \{0,1\}^n$ that satisfy $\mathbf{Ay} = \mathbf{z}(\text{mod } 2)$. □

We will be working with finite field $GF(2)$ throughout this thesis, and from here on, 'mod 2' will be omitted in the equations. Also, the parity check matrices are assumed to be full rank unless otherwise stated. Such a matrix $\mathbf{A}$ partitions the space of $\{0,1\}^n$ into 2^k bins. Each bin corresponds to a different $\mathbf{z}$, i.e. for any $\mathbf{y}$ in this bin, $\mathbf{Ay} = \mathbf{z}$. Each bin contains 2^{n-k} of these size-n binary sequences. As a result of Lemma 1, the ML-sequence set M defined with respect to a full-rank matrix $A \in \{0,1\}^{k \times n}$ and any target distribution will contain exactly 2^k elements. Clearly, Lemma 1 proves argument 1.

Given target distribution $p_{\mathbf{x}}(\cdot)$ defined over alphabet $\mathcal{X}$, if $\mathbf{y} \in \mathcal{X}^t$ is in $T_\delta^{(t)}$, i.e. $\mathbf{y}$ is a typical sequence of $p_{\mathbf{x}}(\cdot)$, by definition we have

$$p_{\mathbf{x}}(\mathbf{y}) = \prod_{a \in \mathcal{X}} (p_{\mathbf{x}}(a))^{N(a|\mathbf{y})} \Rightarrow \log(p_{\mathbf{x}}(\mathbf{y})) = \sum_{a \in \mathcal{X}} N(a|\mathbf{y}) \log(p_{\mathbf{x}}(a))$$

Plug in $|\frac{N(a|\mathbf{y})}{t} - p_{\mathbf{x}}(a)| \leq \delta$, then for any $\mathbf{y} \in T_\delta^{(t)}$

$$\sum_{a \in \mathcal{X}} t(p_{\mathbf{x}}(a) + \delta) \log(p_{\mathbf{x}}(a)) \leq \log(p_{\mathbf{x}}(\mathbf{y})) \leq \sum_{a \in \mathcal{X}} t(p_{\mathbf{x}}(a) - \delta) \log(p_{\mathbf{x}}(a))$$

$$\therefore 2^{-t(H(p_{\mathbf{x}})+\eta)} \leq p_{\mathbf{x}}(\mathbf{y}) \leq 2^{-t(H(p_{\mathbf{x}})-\eta)}$$

where $H(p_{\mathbf{x}}) = -\sum_{a \in \mathcal{X}} p_{\mathbf{x}}(a) \log(p_{\mathbf{x}}(a))$ and $\eta = -\delta \sum_{a \in \mathcal{X}} \log(p_{\mathbf{x}}(a))$ are deterministic functions of $p_{\mathbf{x}}(\cdot)$. The above computation provides upper and lower bounds on the probability of a typical sequence.

Lemma 2. *Given a distribution $p_{\mathbf{x}}(\cdot)$, define set $L_\delta^{(t)} \triangleq \{\mathbf{y} \in \mathcal{X}^t : p_{\mathbf{x}}(\mathbf{y}) > 2^{-t(H(p_{\mathbf{x}})-\eta)}\}$. In other words, $L_\delta^{(t)}$ is the set of all sequences that are more likely than a δ-typical sequence. Then for any $\epsilon > 0$, for any $\delta > 0$, there exists $T \in Z^+$, such that for all $t > T$,*

$$|L_\delta^{(t)}| < \epsilon \cdot 2^{t(H(p_{\mathbf{x}})-\eta)},$$

where $H(p_{\mathbf{x}}) = -\sum_{a \in \mathcal{V}} p_{\mathbf{x}}(a) \log(p_{\mathbf{x}}(a))$ and $\eta = -\delta \sum_{a \in \mathcal{V}} \log(p_{\mathbf{x}}(a))$.

Proof. Using properties of the strong typical set, we know that for any $\epsilon > 0$, there exists large enough t such that $\mathbb{P}(T_\delta^{(t)}) > 1 - \epsilon$. Since $T_\delta^{(t)} \cap L_\delta^{(t)} = \emptyset$, we have

$$\mathbb{P}(L_\delta^{(t)}) \leq \epsilon$$

$$\therefore p_{\mathbf{x}}(\mathbf{y}) > 2^{-t(H(p_{\mathbf{x}})-\eta)}, \forall \mathbf{y} \in L_\delta^{(t)} \Rightarrow |L_\delta^{(t)}| < \epsilon \cdot 2^{t(H(p_{\mathbf{x}})-\eta)}$$

□

If $K \geq t(H(p_{\mathbf{x}}) - \eta)$, Lemma 2 proves argument 2, i.e. number of sequences that are more probable than a typical one is only a small proportion of the number of bins 2^K. In other words, only a small portion of the bins contain sequences that are more likely than a typical one.

Lemma 3 (Theorem 2 in [8]). *Given a distribution $p_{\mathbf{x}}(\cdot)$ of mean entropy $H_{\mathbf{x}} < 1$, and a desired $\lambda < \frac{1}{H_{\mathbf{x}}}$, there exists an integer $w(H_{\mathbf{x}}, \lambda) \geq 3$ such that for any desired block error probability $\epsilon > 0$, there is an integer $k_{\min}$, such that for any $k > k_{\min}$, there is a $(k, \lambda, w(H_{\mathbf{x}}, \lambda), \epsilon)$ satisfactory matrix A for $p_{\mathbf{x}}(\cdot)$.*

In fact, [8] comments that the parity check matrix of almost any well-constructed LDPC code of the appropriate rate and column weight is a $(k, \lambda, w(H_{\mathbf{x}}, \lambda), \epsilon)$ satisfactory matrix. Lemma 3 effectively says that for any given distribution, almost any LDPC code of appropriate rate and column weight satisfies the property that the ML-sequence set has probability close to 1.

Think about playing a game, where the sender generates a sequence $\mathbf{y} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)})$ according to $p_{\mathbf{x}}(\cdot)$ and sends $\mathbf{z} = \mathbb{H}\mathbf{y}$ to the receiver, where $\mathbb{H}$ is the parity check matrix of some LDPC code. The receiver's task is to figure out what $\mathbf{y}$ was sent and wins the game if and only if he correctly guessed $\mathbf{y}$. If the receiver answers with $\hat{\mathbf{y}}$ where $\hat{\mathbf{y}}$ is the most likely $\mathbf{y}$ such that $\mathbb{H}\mathbf{y} = \mathbf{z}$, according to Lemma 3, the receiver will almost always win when the number of repetition of the source (thus the size of the sequence) is allowed to increase.

The proof of Lemma 3 can be found in [8] and thus is omitted here.

Lemma 4. *Given a distribution $p_{\mathsf{x}}(\cdot)$ of mean entropy $H_{\mathsf{x}} < 1$, $\eta = -\delta \sum_{a \in \mathcal{X}} \log(p_{\mathsf{x}}(a))$, M and $T_\delta^{(t)}$ are the ML-sequence set and δ-strong typical set defined with respect to $p_{\mathsf{x}}(\cdot)$ respectively. Then for any $\epsilon > 0$, there exists large enough t such that*

$$(1-\epsilon)2^{t(H(p_{\mathsf{x}})-\eta)-K} \le \frac{|M \cap T_\delta^{(t)}|}{|M|} \le 1$$

Proof. One side is easy:

$$|M \cap T_\delta^{(t)}| \le |M| = 2^K \Rightarrow \frac{|M \cap T_\delta^{(t)}|}{2^K} \le 1$$

From Lemma 3 we know that for any $\epsilon > 0$, there is a large enough t, such that

$$\mathbb{P}(M) > 1 - \frac{\epsilon}{2}$$

On the other hand, from properties of typical sets we have, for any $\epsilon > 0$, $\delta > 0$, there is large enough t such that

$$\mathbb{P}(T_\delta^{(t)}) \ge 1 - \frac{\epsilon}{2}$$

$$\therefore \mathbb{P}(T_\delta^{(t)} \cap M) = 1 - \mathbb{P}((T_\delta^{(t)})^C \cup M^C) \ge 1 - \mathbb{P}((T_\delta^{(t)})^C) - \mathbb{P}(M^C) \ge 1 - \epsilon$$

We have mentioned that the probability of any sequence in $T_\delta^{(t)}$ is at most $2^{-t(H(p_{\mathsf{x}})-\eta)}$

$$\therefore |M \cap T_\delta^{(t)}| \ge (1-\epsilon)2^{t(H(p_{\mathsf{x}})-\eta)}$$

$$\therefore \frac{|M \cap T_\delta^{(t)}|}{|M|} \ge (1-\epsilon)2^{t(H(p_{\mathsf{x}})-\eta)-K}$$

□

If we set $K = t(H(p_{\mathsf{x}}) - \eta)$, we have $1 - \epsilon \le \frac{|M \cap T_\delta^{(t)}|}{|M|} \le 1$.

Remember that maximum-likelihood simulation uniformly picks a bin and look for the most probable sequence in that bin, which is equivalent to uniformly pick one element in the ML-sequence set M. Lemma 4 states that with probability arbitrarily close to 1, the sequence picked is also a strongly typical one and thus provides a good approximation to the target distribution.

Now the proof of Theorem 1 becomes relatively straightforward.

Proof. Given a target distribution $p_{\mathbf{x}}(\cdot)$ over $\{0,1\}^N$, let us set $\mathbf{y}$ to be a concatenation of t independent, identically distributed copies of $\mathbf{x}$, i.e. $\mathbf{y} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(t)})$ and $p_{\mathbf{y}}(\mathbf{y}) = \prod_{i=1}^{t} p_{\mathbf{x}}(\mathbf{x}^{(i)})$. Clearly the model $p_{\mathbf{y}}(\cdot)$ has mean entropy.

From Lemma 4, we know that for large enough t and correctly chosen rate,

$$1 - \epsilon \leq \frac{|M \cap T_\delta^{(t)}|}{|M|} \leq 1$$

In the Maximum-likelihood Simulation, we pick an element of M uniformly at random. Thus the probability that the chosen element is a δ-strong typical sequence is lower bounded by $1 - \epsilon$. □

4.2 Rate of Convergence

In the discussions so far, a 'large enough' t is assumed for asymptotic behaviors to kick in without discussing exactly how large t needs to be. Apparently, the scheme would be of little use if t grows exponentially with N because the computational complexity of the scheme would also be exponential in N in that case.

Unfortunately, it is possible to encounter distributions which requires a large number of repetitions. An example would be a distribution where one of the configurations has a constant probability (say, $1/2$) while all other configurations are equi-probable (roughly $1/2^N$). Intuitively, a large t is needed to 'smooth out' the effect of the highly probable configuration.

The following theorem discusses how the necessary size of t depends on properties of the target distribution $p_{\mathsf{x}}(\cdot)$.

Theorem 2. *For any distribution $p_{\mathsf{x}}(\cdot)$ where $\mathbf{x} \in \{0,1\}^N \triangleq \mathcal{X}$, and $\forall \epsilon > 0$, $\forall \delta > 0$, $\mathbb{P}(T_\delta^{(t)}) \geq 1 - \epsilon$ if*

$$t \geq \frac{2\ln(2)}{\delta^2} \ln(\frac{2|\mathcal{X}|}{\epsilon}) \max_{a \in \mathcal{X}} \{p_{\mathsf{x}}(a)(1 - p_{\mathsf{x}}(a))\}$$

where $T_\delta^{(t)}$ is the δ-strong typical set defined for $p_{\mathsf{x}}(\cdot)$

Notice Theorem 2 argues that the number of repetitions needed for typicality arguments to be valid is proportional to $\ln(|\mathcal{X}|)$, i.e. linear in N. The required t is also proportional to $\frac{1}{\delta^2}$ and $\ln\frac{1}{\epsilon}$, as well as $\max_{a\in\mathcal{X}} p_{\mathsf{x}}(a)(1 - p_{\mathsf{x}}(a))$. If probability for individual elements $p_{\mathsf{x}}(a)$ is close to 0 for all $a \in \mathcal{X}$, then

$$\max_{a\in\mathcal{X}} p_{\mathsf{x}}(a)(1 - p_{\mathsf{x}}(a)) \approx \max_{a\in\mathcal{X}} p_{\mathsf{x}}(a)$$

Notice that for sources that possess a high degree of regularity, e.g. Markov chain source or Ising models that we will consider in Section 6.1, $\max_{a\in\mathcal{X}} p_{\mathsf{x}}(a) = c^N$, where c is a constant and $c < 1$. Thus the bound on t scales as $O(Nc^N)$ and becomes negligibly small for large N. In such cases, the increase in computational cost brought about by repeating the source becomes insignificant.

In order to prove Theorem 2, the following notations and lemmas are introduced:

1. If $u \sim B(n, p)$ is a binomial random variable, the probability mass function of u is $f(k; n, p) = \mathbb{P}(u = k) = \binom{n}{k} p^k (1-p)^{n-k}$, and the cumulative distribution function of u is denoted as $F(k; n, p) = \mathbb{P}(u \leq k)$.

2. If u and v are two Bernoulli random variables with $\mathbb{P}(u = 1) = a$ and $\mathbb{P}(v = 1) = b$ respectively, by a slight abuse of notation, we will denote $D(p_u || p_v)$, the KL-divergence between u and v, as $D(a||b)$. Clearly, $D(a||b) = D(1-a||1-b)$.

The following lemma provides an upper bound for the cumulative distribution function of binomial distribution:

Lemma 5. $F(k; n, p) \leq \exp(-nD(\frac{k}{n}\|p))$

This is a well-known result and its proof can be found in [7] and thus is omitted here.

Lemma 6. *Assume* $0 < a < b < 1$*. If the value of* b *is fixed,* $D(a\|b)$ *is locally a decreasing function of* a*.*

Proof. This should be intuitive as divergence is in some sense a measure of 'distance' (square of distance) between two distributions. Thus the larger a is (while $a < b$ still holds), the closer the two Bernoulli distributions are, and the smaller $D(a\|b)$ is. More concretely:

$$\begin{aligned}\frac{\partial D(a\|b)}{\partial a} &= \frac{\partial}{\partial a}(a \log \frac{a}{b} + (1-a)\log \frac{1-a}{1-b}) \\ &= \log a + \frac{1}{\ln 2} - \log b - \log(1-a) - \frac{1}{\ln 2} + \log(1-b) \\ &= \log \frac{a(1-b)}{b(1-a)} = \log \frac{a-ab}{b-ab}\end{aligned}$$

Since $a < b \Rightarrow a - ab < b - ab \Rightarrow \dfrac{\partial D(a\|b)}{\partial a} < 0$
$\therefore$ $D(a\|b)$ is locally a decreasing function of a. $\square$

Now we are ready to prove Theorem 3.

Proof of Theorem 3. $\mathbb{P}(T_\delta^{(t)}) \geq 1 - \epsilon$ is equivalent to:

$$\mathbb{P}(\exists a \in \mathcal{X}, \text{s.t. } |\frac{N(a|\mathbf{y})}{t} - p_{\mathsf{x}}(a)| > \delta) \leq \epsilon$$

Here $\mathsf{y} \in \{0,1\}^{Nt}$, but is viewed as a t-symbol sequence where each symbol consists of N bits and is generated from $p_{\mathsf{x}}(\cdot)$. $N(a|\mathsf{y})$ is the number of appearances of symbol a in the sequence.

From Union Bound:

$$\mathbb{P}(\exists a \in \mathcal{X}, \text{s.t. } |\frac{N(a|\mathbf{y})}{t} - p_{\mathbf{x}}(a)| > \delta) \leq \sum_{a\in\mathcal{X}} \mathbb{P}(|\frac{N(a|\mathbf{y})}{t} - p_{\mathbf{x}}(a)| > \delta)$$

Look at each term in closer detail:

$$\begin{aligned}\mathbb{P}(|\frac{N(a|\mathbf{y})}{t} - p_{\mathbf{x}}(a)| > \delta) &= 1 - \mathbb{P}(|\frac{N(a|\mathbf{y})}{t} - p_{\mathbf{x}}(a)| \leq \delta) \\ &= 1 - \sum_{i=\lceil t(p_{\mathbf{x}}(a)-\delta)\rceil}^{\lfloor t(p_{\mathbf{x}}(a)+\delta)\rfloor} \binom{t}{i} p_{\mathbf{x}}(a)^i (1-p_{\mathbf{x}}(a))^{t-i} \qquad (*)\end{aligned}$$

Denote $\lceil t(p_{\mathbf{x}}(a) - \delta)\rceil$ as c_1, and denote $\lfloor t(p_{\mathbf{x}}(a) + \delta)\rfloor$ as c_2

$$\begin{aligned}(*) &= \sum_{i=0}^{c_1-1} \binom{t}{i} p_{\mathbf{x}}(a)^i (1-p_{\mathbf{x}}(a))^{t-i} + \sum_{i=c_2+1}^{t} \binom{t}{i} p_{\mathbf{x}}(a)^i (1-p_{\mathbf{x}}(a))^{t-i} \\ &= \sum_{i=0}^{c_1-1} \binom{t}{i} p_{\mathbf{x}}(a)^i (1-p_{\mathbf{x}}(a))^{t-i} + \sum_{i=0}^{t-c_2-1} \binom{t}{i} p_{\mathbf{x}}(a)^{t-i} (1-p_{\mathbf{x}}(a))^{i} \\ &= F(c_1 - 1; t, p_{\mathbf{x}}(a)) + F(t - c_2 - 1; t, 1 - p_{\mathbf{x}}(a)) \\ &\leq \exp(-tD(\frac{c_1-1}{t} \| p_{\mathbf{x}}(a))) + \exp(-tD(\frac{t-c_2-1}{t} \| 1 - p_{\mathbf{x}}(a))) \qquad (**)\end{aligned}$$

The last step used in $(**)$ Lemma 5. Notice:

$$\begin{aligned}\frac{c_1-1}{t} &\leq \frac{t(p_{\mathbf{x}}(a)-\delta)+1-1}{t} = p_{\mathbf{x}}(a) - \delta < p_{\mathbf{x}}(a) \\ \frac{t-c_2-1}{t} &\leq \frac{t - t(p_{\mathbf{x}}(a)+\delta)+1-1}{t} = 1 - p_{\mathbf{x}}(a) - \delta < 1 - p_{\mathbf{x}}(a)\end{aligned}$$

Applying Lemma 6, we get:

$$\begin{aligned}(**) &\leq \exp(-tD(p_{\mathbf{x}}(a) - \delta \| p_{\mathbf{x}} a))) + \exp(-tD(1 - p_{\mathbf{x}}(a) - \delta \| 1 - p_{\mathbf{x}}(a))) \\ &= \exp(-tD(p_{\mathbf{x}}(a) - \delta \| p_{\mathbf{x}}(a))) + \exp(-tD(p_{\mathbf{x}}(a) + \delta \| p_{\mathbf{x}}(a)))\end{aligned}$$

Since δ is a small positive constant, let us use Taylor expansion:

$$D(p_{\mathsf{x}}(a)+\delta \| p_{\mathsf{x}}(a)) = (p_{\mathsf{x}}(a)+\delta)\log(\frac{p_{\mathsf{x}}(a)+\delta}{p_{\mathsf{x}}(a)}) + (1-p_{\mathsf{x}}(a)-\delta)\log(\frac{1-p_{\mathsf{x}}(a)-\delta}{1-p_{\mathsf{x}}(a)})$$
$$= p_{\mathsf{x}}(a)\log(1+\frac{\delta}{p_{\mathsf{x}}(a)}) + \delta\log(1+\frac{\delta}{p_{\mathsf{x}}(a)})$$
$$+ (1-p_{\mathsf{x}}(a))\log(1-\frac{\delta}{1-p_{\mathsf{x}}(a)}) + \delta\log(1-\frac{\delta}{1-p_{\mathsf{x}}(a)})$$
$$= \frac{1}{\ln 2}\{p_{\mathsf{x}}(a)(\frac{\delta}{p_{\mathsf{x}}(a)} - \frac{\delta^2}{2p_{\mathsf{x}}(a)^2}) + \delta(\frac{\delta}{p_{\mathsf{x}}(a)} - \frac{\delta^2}{2p_{\mathsf{x}}(a)^2})$$
$$+ (1-p_{\mathsf{x}}(a))(-\frac{\delta}{1-p_{\mathsf{x}}(a)} - \frac{\delta^2}{2(1-p_{\mathsf{x}}(a))^2}) + \delta(-\frac{\delta}{1-p_{\mathsf{x}}(a)} - \frac{\delta^2}{2(1-p_{\mathsf{x}}(a))^2}) + o(\delta^2)\}$$
$$\approx \frac{\delta^2}{2\ln 2}\frac{1}{p_{\mathsf{x}}(a)(1-p_{\mathsf{x}}(a))}$$

By symmetry,

$$D(p_{\mathsf{x}}(a)-\delta \| p_{\mathsf{x}}(a)) \approx \frac{\delta^2}{2\ln 2}\frac{1}{p_{\mathsf{x}}(a)(1-p_{\mathsf{x}}(a))}$$

$$\therefore \mathbb{P}(\exists a \in \mathcal{X}, \text{s.t. } |\frac{N(a|\tilde{\mathbf{x}})}{t} - p_{\mathsf{x}}(a)| > \delta) \le \sum_{a\in\mathcal{X}} 2\exp(-t\frac{\delta^2}{2\ln 2}\frac{1}{p_{\mathsf{x}}(a)(1-p_{\mathsf{x}}(a))})$$
$$\le 2|\mathcal{X}|\max_{p_{\mathsf{x}}(a)}\exp(-t\frac{\delta^2}{2\ln 2}\frac{1}{p_{\mathsf{x}}(a)(1-p_{\mathsf{x}}(a))})$$

Clearly, a sufficient condition for $\mathbb{P}(T_\delta^{(t)}) \ge 1-\epsilon$ is:

$$2|\mathcal{X}|\max_{a\in\mathcal{X}}\exp(-t\frac{\delta^2}{2\ln 2}\frac{1}{p_{\mathsf{x}}(a)(1-p_{\mathsf{x}}(a))}) \le \epsilon$$

$$\Rightarrow t \ge \frac{2\ln 2}{\delta^2}\ln\frac{2|\mathcal{X}|}{\epsilon}\max_{a\in\mathcal{X}} p_{\mathsf{x}}(a)(1-p_{\mathsf{x}}(a))$$

□

Chapter 5

Approximate Simulation

This chapter discusses the practical implementation details of the sampling scheme proposed in Section 3.3. A description of how the LDPC code used in the simulations is constructed is included in Section 5.1. Since the computational cost of Maximum-likelihood simulation is generally intractable, approximate simulation is performed instead via Belief Propagation algorithm. Section 5.2 presents the message update equations of BP algorithm, specialized to the proposed sampling scheme. Section 5.3 and 5.4 provide further details on how to start and terminate the BP algorithm. Finally, Section 5.5 focuses on how to dynamically determine the right code rate for the LDPC code.

5.1 Generation of LDPC Codes

The parity check matrix $\mathbb{H}$ used in the simulations is randomly generated, with almost constant left degree 3 (i.e. each variable participates in roughly three different checks). The procedure of producing such an $\mathbb{H}$ is described below:

1. Given the specified number of rows and columns, generate an all-zero matrix of appropriate size.

2. For each column, independently place three 1's at random rows.

3. For each row, if the number of 1's in the row is fewer than 2, place extra 1's in the row at random places.

4. Get rid of size-4 cycles on the factor graph corresponding to $\mathbb{H}$. In other words, make sure $\nexists a, b, i, j$ such that

$$\mathbb{H}_{a,i} = \mathbb{H}_{a,j} = \mathbb{H}_{b,i} = \mathbb{H}_{b,j} = 1$$

Step 4 is necessary due to the fact that BP algorithm is known to have poor performance on graphs with very small cycles. Step 4 may introduce new degree-1 checks or redundant rows (i.e. all-zero rows). One way to deal with this problem is to iterate step 3 and 4. In practice, when $\mathbb{H}$ is reasonably large, the probability of size-4 cycle appearing becomes pretty low[1] and the probability of introducing degree-1 check or redundant row in step 4 is even lower.

5.2 Belief Propagation Algorithm

Belief Propagation (BP), also known as Sum-Product algorithm, is a popular message passing algorithm. The general form of BP and its derivation can be found in many standard textbooks (e.g. [6], [27]) and thus are omitted here. Below is a detailed description of the messages specialized to the proposed sampling scheme.

For the clarity of presentation, the subscripts on variable/check/factor nodes will be omitted; but since messages are only defined for variable and check/factor nodes that are connected, the subscript should be easy to infer from context.

Let $\mathcal{N}(v)$ denote the neighbourhood of variable node v, i.e. the set of all the check

[1]For a matrix with K rows and N columns, the expected number of size 4 cycles is roughly

$$\binom{N}{2}\frac{3\binom{K-2}{1}}{\binom{K}{3}}$$

which is a constant if the ratio between N and K is fixed.

nodes and factor nodes that are connected to v. $\mathcal{N}(f)$ and $\mathcal{N}(c)$ are defined in a similar manner. Let x be the value taken by a variable v. Given a set $\mathcal{S}$ of variables, $\mathbf{x}_{\mathcal{S}}$ denotes the vector of values taken by all variable nodes in $\mathcal{S}$.

1. Messages from factor nodes to variable nodes:

$$m_{f\to v}^{(t+1)}(x) = \sum_{\mathbf{x}_{\mathcal{N}(f)\setminus\{v\}}} f(\mathbf{x}_{\mathcal{N}(f)\setminus\{v\}}, x) \prod_{v_* \in \mathcal{N}(f)\setminus\{v\}} m_{v_*\to f}^{(t)}(x_*)$$

2. Messages from check nodes to variables nodes:

$$m_{c\to v}^{(t+1)}(x) = \sum_{\mathbf{x}_{\mathcal{N}(c)\setminus\{v\}}} \mathbb{1}\{ \sum_{v_* \in N(c)\setminus\{v\}} v_* = z + x\} \prod_{v_* \in N(c)\setminus\{v\}} m_{v_*\to c}^{(t)}(v_*)$$

   where z is the parity check value corresponding to check c.

3. Messages from variable nodes to factor nodes:

$$m_{v\to f}^{(t+1)}(x) \propto \{ \prod_{c\in N(v)} m_{c\to v}^{(t)}(x)\}\{ \prod_{f_* \in N(V)\setminus\{f\}} m_{f_*\to v}^{(t)}(x)\}$$

   There is no fundamental difference between check nodes and factor nodes in this equation. Separating them into two terms is mainly for clarity.

4. Messages from variable nodes to check nodes:

$$m_{v\to c}^{(t+1)}(x) \propto \{ \prod_{f\in N(v)} m_{f\to v}^{(t)}(x)\}\{ \prod_{c_* \in N(v)\setminus\{c\}} m_{c_*\to v}^{(t)}(x)\}$$

5. At each iteration, the marginal distribution of each variable is computed from all the incoming messages

$$\mathbb{P}_v^{(t)}(x) \propto \{ \prod_{f\in N(v)} m_{f\to v}^{(t)}(x)\}\{ \prod_{c\in N(v)} m_{c\to v}^{(t)}(x)\}$$

   Symbol MAP decoding is used to produce an estimate of the values of the variables at each iteration. :

$$\hat{x}^{(t)} = \begin{cases} 1 & \text{if } \mathbb{P}_v^{(t)}(x=0) < \frac{1}{2} \\ 0 & \text{if } \mathbb{P}_v^{(t)}(x=0) \geq \frac{1}{2} \end{cases}$$

It should be noted that when the code rate is high enough, the probability that a variable have [0.5,0.5] marginals after the BP has converged should be very low. Thus if $\mathbb{P}_v(0) = \frac{1}{2}$, the tie can actually be broken arbitrarily. We will consistently assign such variables to 0 in the simulations so that in order to diagnose when the rate is too low (i.e. too many zeros come up.)

5.3 Initialization of Messages and Doping

In the simulations, a uniform initial value is used for all the variables to factor/check messages (i.e. all messages from variables to checks/factors are $[0.5, 0.5]^T$). It should be noted that for each check node, if any of its incoming messages from a connected variable is uniform, the outgoing messages from this check node to other variables is also uniform. If the source is symmetric about 0 and 1[2], all messages in subsequent iterations will be uniform if initial messages are, which is not a very interesting case.

One alternative choice would be a random initialization, but in practice such an initialization often results in failure of convergence of BP algorithm. Instead, a few randomly chosen variables, known as 'doping bits', are set to 0 or 1 uniformly randomly and then viewed as fixed. Doping bits can effectively start the algorithm yet keep the perturbation 'local', and they can be thought of as degree-1 check nodes. The number of doping bits is a parameter to tune, but should be a small fraction of the total number of variables in the system. In the simulations, doping rate = 0.05 is used. That is, 5% of the nodes are doped.

[2]All three distributions we considered here are symmetric about 0 and 1.

5.4 Termination Criteria

BP algorithm terminates either when all messages have stabilized or a specified maximum number of iterations is reached. The former case is regarded as 'converged' and the latter 'not converged'. To be more concrete, one keeps track of changes in messages between iterations. If the maximum change in all the messages has been smaller than a certain constant[3] for several consecutive iterations, the algorithm is said to have converged. Section 6.2.1 is devoted to the discussion of various aspects of convergence behavior in the simulations.

It is worth pointing out that the convergence behavior is somewhat different from that in [29] [30], where the same framework is used for compression. In the compression scenario, a sequence $\mathbf{x}$ is generated from a given source distribution and $\mathbb{H}$ is the parity check matrix of a randomly chosen LDPC code. $\mathbf{z} = \mathbb{H}\mathbf{x}$ is the output of the compressor and decompression is achieved by running BP algorithm on the combined graph. As $\mathbf{z}$ and the doping bits are both computed from the same $\mathbf{x}$ and thus consistent with each other, a higher code rate will never do any harm to the decoding process. And with a high enough rate, the correct $\mathbf{x}$ can be recovered. In the sampling application discussed in this thesis, however, the doping bits and the parity check values $\mathbf{z}$ are both generated randomly since 'ground truth' is no longer available. With increasing code rate, it is more likely that information from doping bits and the parity checks contains conflicts and thus BP algorithm may not converge.

5.5 Finding the Correct Code Rate

As discussed in Chapter 4.1, the required rate of LDPC code roughly equals to the entropy rate of the target distribution. Intuitively, that is the average number of bits of information contained in each bit of a sample. When the rate is too high, code graph and source graph provide contradicting information and BP algorithm fail to converge. When the rate is too low, there will be variables that do not receive enough

[3]0.01 was used in the simulations if not stated otherwise.

information from the checks/factors and their estimated marginals remain close to $[0.5, 0.5]^T$. The goal is to find the 'correct' rate, i.e. it is desirable to introduce to the system as many checks as possible, provided the BP algorithm can still converge[4].

However, the exact rate varies across different $\mathbb{H}$'s. For each LDPC code chosen, its 'correct' rate is individually found using the following procedure described:

1. Generate an LDPC code of rate r, where r is guaranteed to be an overestimate. In other words, if BP algorithm is run on the combined graph constructed using this LDPC code, it does not converge.

2. Decrease the number of checks in the LDPC code by a prescribed amount each time and run BP algorithm on the resulting combined graph to identify the approximate region of 'correct' rate. The prescribed amount should be reasonably large so that the approximate region can be found fast enough, but small enough to avoid a very time-consuming step 3.

3. In the approximate region, decrease the number of checks by a smaller prescribed amount each time and run BP algorithm on the resulting combined graph. In other words, a finer scan of the approximate region (found in step 2) is performed. If the number of unsettled variables (i.e. variables whose marginal is still changing by non-negligible amount after some fixed number of iterations) has increased considerably from last time, the checks just removed are regarded as 'important'[5]. The ordering of the checks is rearranged so that the 'important' checks are back in the combined graph. Instead a different set of checks is removed and step 3 is iterated until the 'correct' rate is found.

Section 6.2.1 will look in closer details at how different aspects of the BP algorithm on the combined graph change as the number of checks is decreased, which will justify the procedure described here.

[4]Empirically this provides good samples of the target distributions.
[5]for example, a degree-2 check is usually important.

Chapter 6

Simulation Results and Analysis

In this chapter, the LDPC-based sampling scheme developed in previous chapters is used to produce samples from three different types of target distributions: Markov Chain source, Single Loop source, and 2-dimensional Ising Model source. As discussed before, Maximum-likelihood simulation is theoretically correct but computationally intractable. Thus the results in this chapter are obtained by approximate simulations performed via Belief Propagation algorithm.

In Section 6.1, simulation results are reported for the three target distributions. Various properties of the obtained samples are examined to measure their quality. This is followed by discussions based on the simulation results presented in Section 6.2, where we attempt to answer the following questions: In the proposed sampling scheme, how does the convergence behaviour of Belief Propagation algorithm depend on the rate of the LDPC code? Can we make sense of the convergence behaviour intuitively? Apart from code rate, what other factors affect the quality of the obtained samples?

6.1 Simulation Results for Different Sources

This section contains the simulation results of the proposed LDPC-based sampling technique for different types of target distributions: Markov Chain source, Single Loop source, and Ising Model source. All the target distributions are assumed to be homogeneous. It is possible to sample exactly from the first two sources and thus the obtained samples are compared against those from exact sampling. For Ising model, exact sampling is computationally intractable. Therefore the results are compared with samples obtained using Markov Chain Monte Carlo methods, which are widely used in practice.

6.1.1 Markov Chain

The schematic of a Markov Chain source is shown in Figure 6-1:

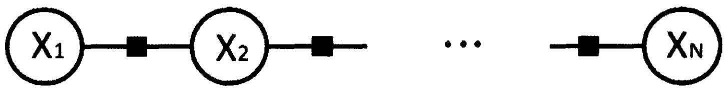


Figure 6-1: Schematic of Markov Chain source

The factors are homogeneous and symmetric about 0 and 1, i.e. $\forall i \in \{1, 2, ..., N-1\}$

$$\psi_{x_i, x_{i+1}} = \begin{cases} p & \text{if } x_i \neq x_{i+1} \\ 1-p & \text{if } x_i = x_{i+1} \end{cases}$$

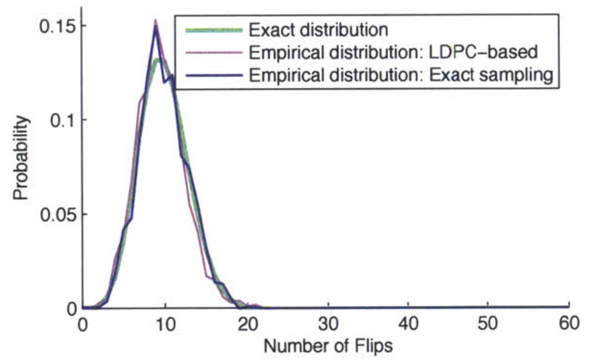
where $x_i \in \{0, 1\}$. The plots and figures in this section are produced by setting the parameters as in Table 6.1:

Number of Variables in the Source N	100
Number of Repetitions of the Source t	10
Probability of Flipping p	0.1/0.2/0.3/0.4
Number of Maximum Iterations in Belief Propagation algorithm	200
Doping Rate	0.05
Total Number of Samples	~1000

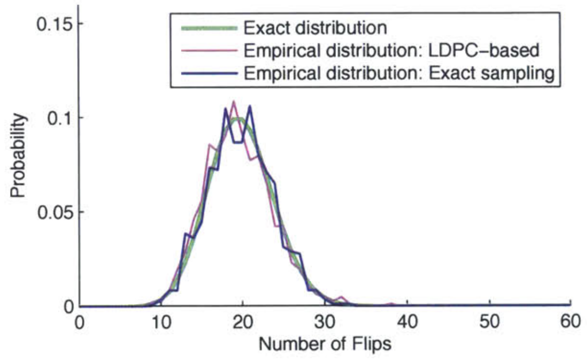
Table 6.1: Parameters for Markov Chain source.

Exact sampling from Markov Chain source is straightforward: one can first sample x_1 from $p_{x_1}(\cdot)$, then sample x_2 from $p_{x_2|x_1}(\cdot|x_1)$, and so on. The distribution to sample from in each step is a Bernoulli one and there are N steps in total.

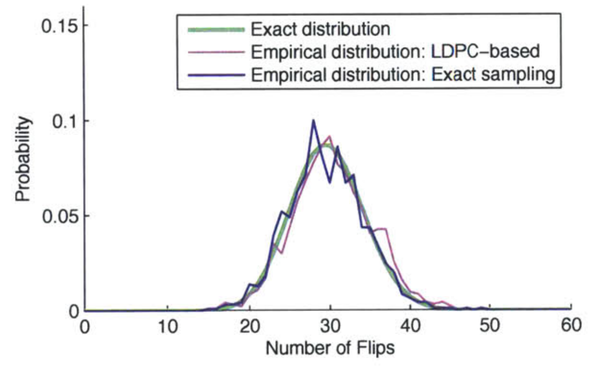
For samples from homogeneous Markov chains, one important property is the number of flips F (a 'flip' is a pair of neighbouring variables that take different values). Figure 6-2 examines the distribution of F, computed using samples from exact sampling and the proposed sampling scheme respectively.



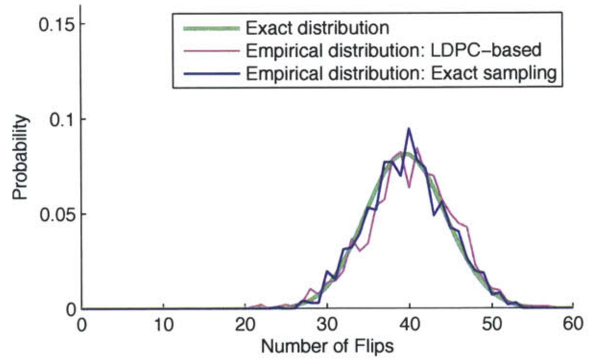
(a) $p = 0.1$



(b) $p = 0.2$



(c) $p = 0.3$



(d) $p = 0.4$

Figure 6-2: Markov Chain source: distribution of number of flips F.

The horizontal axis in the above plots shows the number of flips F. The vertical axis shows the probability of having a certain number of flips. The green curves indicate the true distribution of F, which is computed using 100000 samples from exact sampling. The purple curves are constructed using samples obtained from the proposed method. The blue curves are plotted using the same number of samples ($\sim$1000), but obtained via exact sampling method.

It can be seen from Figure 6-2 that the distribution of F produced by the proposed sampling scheme follows the true distribution fairly nicely, but the curve is less smooth. The 'noise' is expected, though, because only 1000 samples are used to produce these figures. However, as the plots suggest, the performance is comparable to the case where the same number of samples from exact sampling are used. To be more quantitative, the divergence between the empirical distributions of F and the true distribution are computed and listed in the Table 6.2[1]:

p	Proposed Scheme (1000 samples)	Exact Sampling (1000 samples)
0.1	0.0260	0.0215
0.2	0.0284	0.0326
0.3	0.0393	0.0295
0.4	0.0483	0.0354

Table 6.2: Divergence between the empirical distributions and true distribution of F.

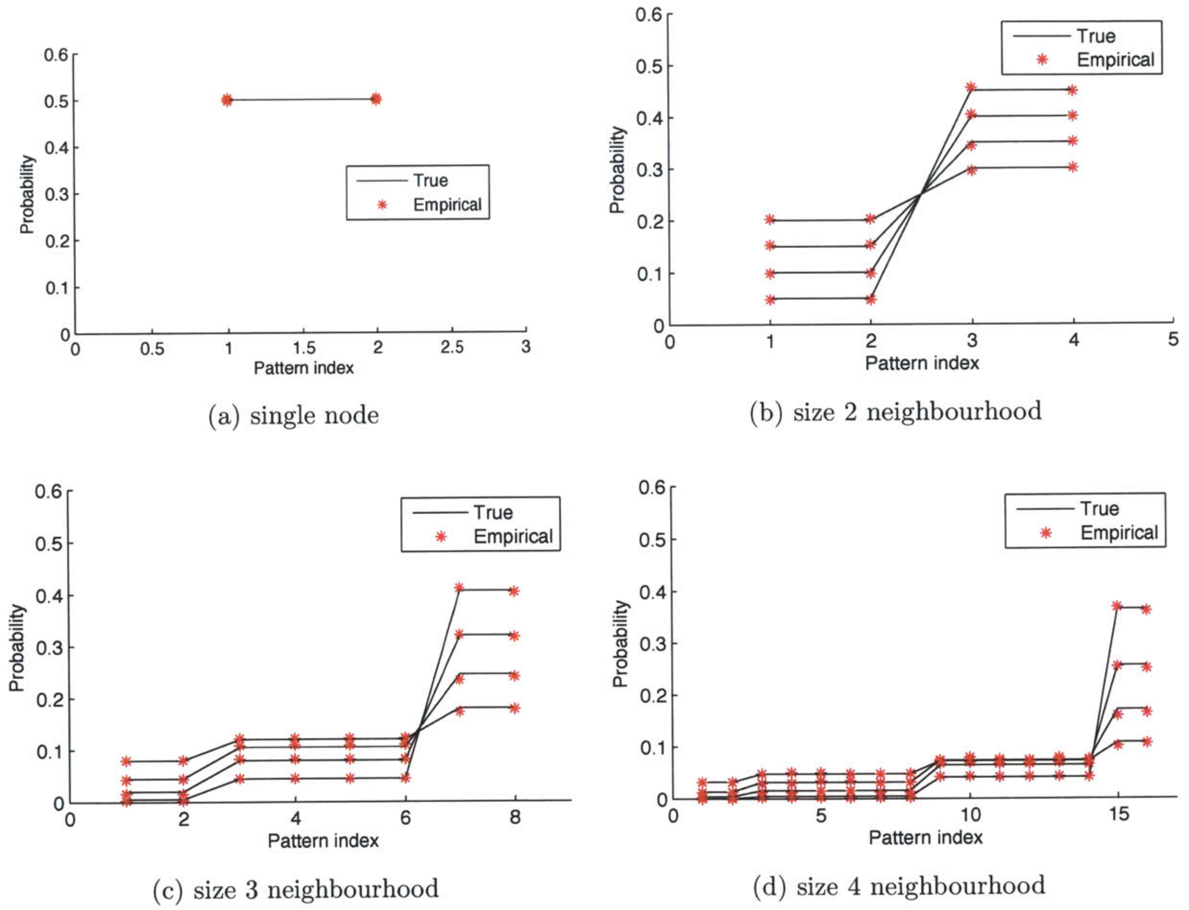
To further examine the quality of the obtained samples, it would be helpful to look at some local structures and test whether the frequencies of different patterns appeared in the obtained samples agree with analytical prediction, i.e. the marginal distributions of the local structures. These local tests will be particularly useful when exact sampling of the source is not possible.

Figure 6-3 looks at four local structures in Markov chain: single node and neighbourhood of size 2, 3 and 4 respectively. Note that in the Markov Chain case, these neighbourhoods are just sub-chains. A 'pattern' is a configuration of the local structure, e.g. a single node has two patterns: 0 and 1; for a size-2 neighbourhood, there

[1]Divergence between two distributions $p(\cdot)$ and $q(\cdot)$ is computed by $D(p\|q) = \sum_{a\in\mathcal{X}} p(a)\log\frac{p(a)}{q(a)}$, where $\mathcal{X}$ is the alphabet for $p(\cdot)$ and $q(\cdot)$ and $|\mathcal{X}| < \infty$.

are 4 possible patterns: 00, 01, 10, and 11. Similarly, size 3 and 4 neighbourhood have 8 and 16 patterns respectively. Since the target distribution is assumed to be homogeneous, averages can be taken with respect to different local structures of the same size.

Each plot in Figure 6-3 focuses on one local structure and each plot contains 4 sets of data (for p = 0.1, 0.2, 0.3 and 0.4 respectively). Horizontal axis shows the index of patterns, while vertical axis shows the probability. Black lines indicate the probabilities of different patterns computed analytically. Red stars are the empirical probabilities estimated using the samples from proposed scheme. The patterns are sorted in increasing probability for clarity of presentation.



(a) single node

(b) size 2 neighbourhood

(c) size 3 neighbourhood

(d) size 4 neighbourhood

Figure 6-3: Markov Chain source: the comparison of theoretical and empirical probabilities of different patterns in different local structures.

As can be seen from the plots, the marginals over local structures are nicely approximated using the samples from the proposed LDPC-sampling scheme.

In practice, a common usage of samples is to estimate some quantities about the variable. In the Markov Chain case, a natural quantity to estimate is the flipping probability p. Since the Markov chain is homogeneous, p can be estimated by

$$\frac{1}{(N-1)S}\sum_{i=1}^{S} F(\mathbf{x}^{(i)})$$

where S is the number of samples and thus $(N-1)S$ is the total number of neighbouring pairs in the samples. $F(\mathbf{x}^{(i)})$ is the number of flips in sample $\mathbf{x}^{(i)}$.

Table 6.3 shows the empirically estimated value of p versus the truth value. As can be seen from the table, samples from proposed distribution provide a reasonably good approximation of the true value of p. This should not be surprising because as shown in Figure 6-2, the samples from the proposed scheme can approximate the distribution of F nicely.

True Flipping Probability p	0.1000	0.2000	0.3000	0.4000
Estimated Flipping Probability	0.0962	0.1968	0.3064	0.4051

Table 6.3: Markov Chain source: estimation of flipping probability p.

6.1.2 Single Loop Source

The schematic of a homogeneous Single Loop source is shown in Figure 6-4. Just as the Markov Chain source considered in last section, the factors are homogeneous and symmetric about 0 and 1, i.e. for any $i \in \{1, 2, ..., N\}$

$$\psi_{x_i, x_{i+1}} = \begin{cases} p & \text{if } x_i \neq x_{i+1} \\ 1-p & \text{if } x_i = x_{i+1} \end{cases}$$

$x_i \in \{0, 1\}$, and assume $x_{N+1} = x_1$.

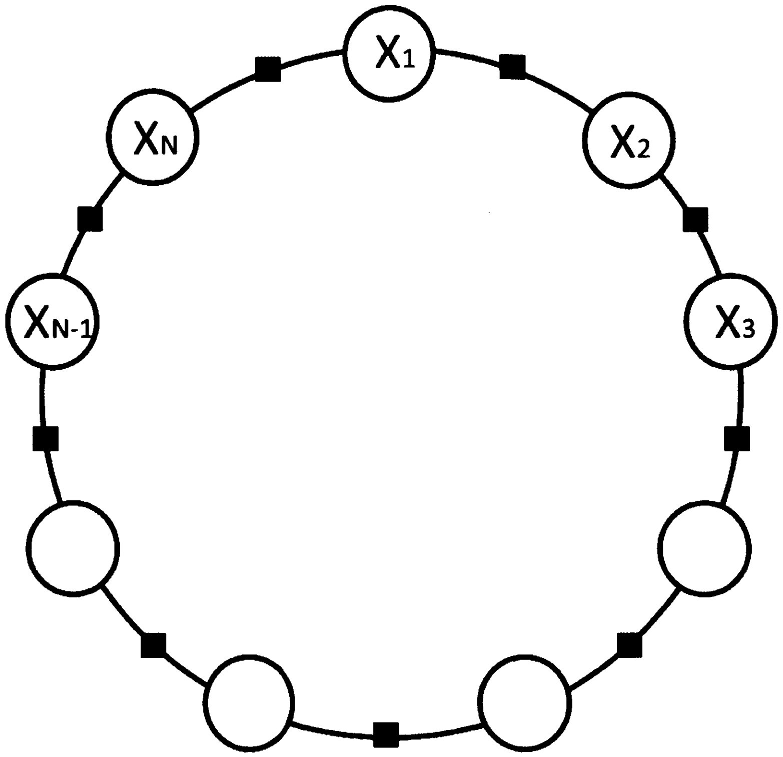


Figure 6-4: Schematic for single loop source.

The parameters are set as in Table 6.4:

Number of Variables in the Source N	100
Number of Repetitions of the Source t	10
Probability of Flipping p	0.1/0.2/0.3/0.4
Number of Maximum Iterations in Belief Propagation algorithm	200
Doping Rate	0.05
Total Number of Samples	$\sim$1000

Table 6.4: Parameters for single loop source.

Notice that due to the homogeneity and symmetry of the loop, the distribution can be sampled exactly using the following algorithm:

1. Sample the number of flips F. Notice

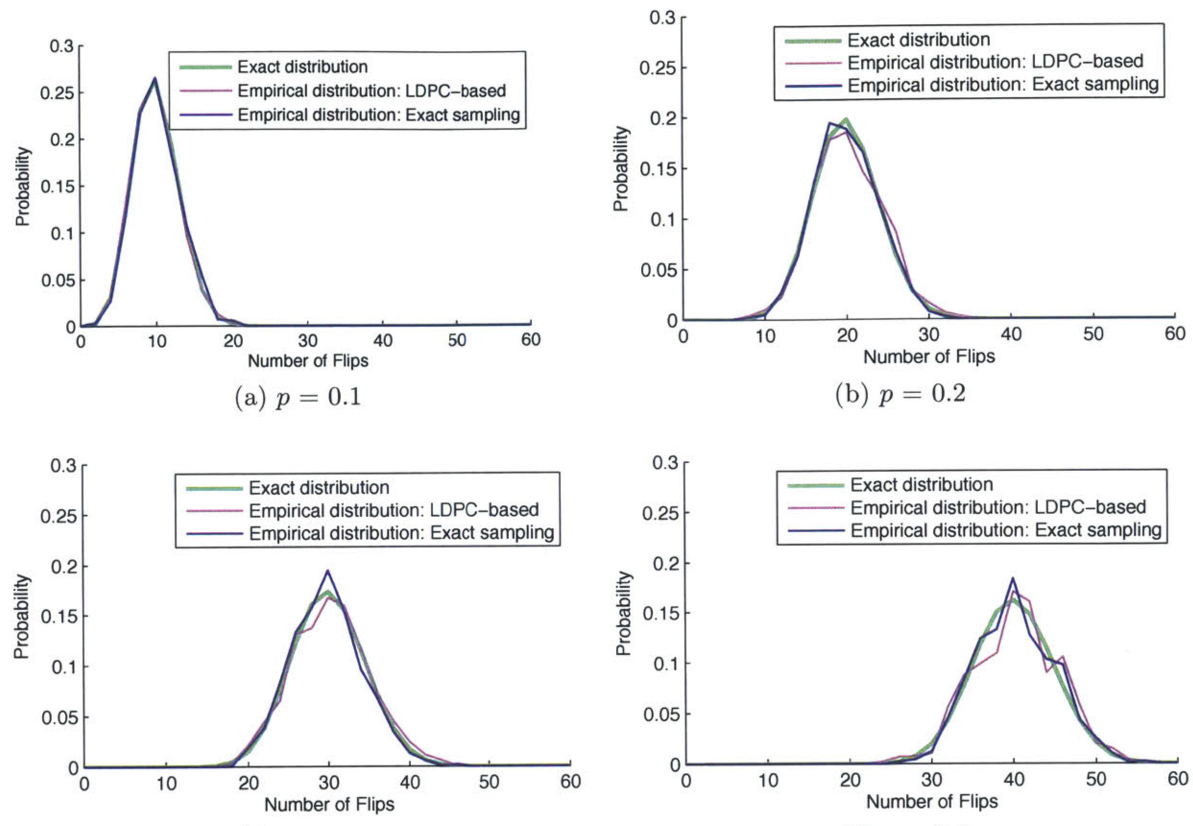
$$\mathbb{P}(F = k) \propto \binom{N}{k} p^k (1-p)^{N-k} \text{ if } k \text{ is even and}$$
$$\mathbb{P}(F = k) = 0 \text{ if } k \text{ is odd}$$

   Since F only takes $N + 1$ possible values, the partition function of distribution of F can be computed in time linear with respect to N. Once the partition function is available, F can be easily sampled.

2. Given F, all configurations with F flips are equi-probable. A configuration can be sampled by uniformly randomly picking F positions out of the total N positions, where the flips can happen. There will be two configurations corresponding to the same 'flip pattern' and the two configurations are complement of each other (all bits are different). Since the source is symmetric about 0 and 1, uniformly randomly pick one of them as the desired sample.

As with Markov Chain source, the distribution of the number of flips F, is examined first and the corresponding plots are shown in Figure 6-5. The horizontal axis shows F and vertical axis shows the probability. The green, purple and blue curve indicate the true distribution, empirical distribution constructed using samples from the proposed method and using the same number of samples from exact sampling method, respectively. As discussed before, $\mathbb{P}(F = k) = 0$ if k is odd. But these zero data points are not shown in the plots, mainly for clarity of illustration.

Similar to the Markov Chain case, the samples from LDPC-based sampling method follows the true distribution fairly well. Some noises do exist, mainly due to the relatively small number of samples used. The performance is comparable to exact sampling provided the number of samples is kept the same. The divergence between the empirical and theoretical distributions of F is shown in Table 6.5:



(a) $p = 0.1$ (b) $p = 0.2$ (c) $p = 0.3$ (d) $p = 0.4$

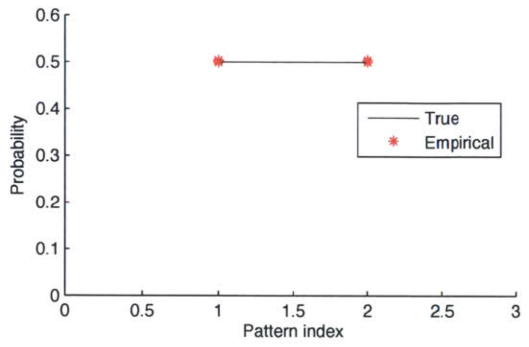
Figure 6-5: Single loop source: distribution of number of flips F.

p	Proposed Scheme (1000 samples)	Exact Sampling (1000 samples)
0.1	0.0021	0.0104
0.2	0.0184	0.0259
0.3	0.0183	0.0105
0.4	0.0444	0.0193

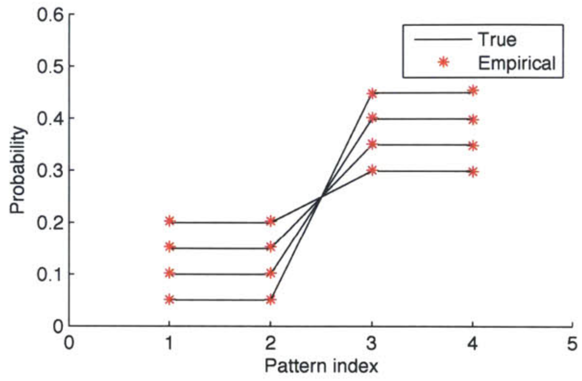
Table 6.5: Divergence between the empirical distributions and true distribution of F.

Next, Figure 6-6 compares the empirical and theoretical probabilities of different patterns of local structures. Again four local structures are considered: single node, and neighbourhood of size 2, 3, and 4. Horizontal axis represents the pattern index and vertical axis shows the probability. As can be seen from Figure 6-6, the probabilities of different patterns in local structures are well approximated by samples from the proposed sampling scheme.

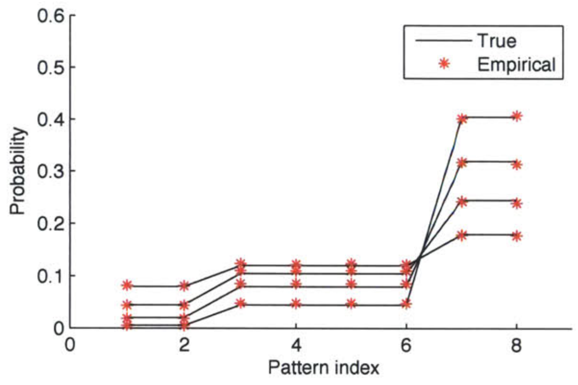
Table 6.6 compares the flipping probability estimated by the proposed method with the truth value, and the results indicate that the LDPC-based sampling scheme can estimate flipping probability of the single loop source relatively accurately.



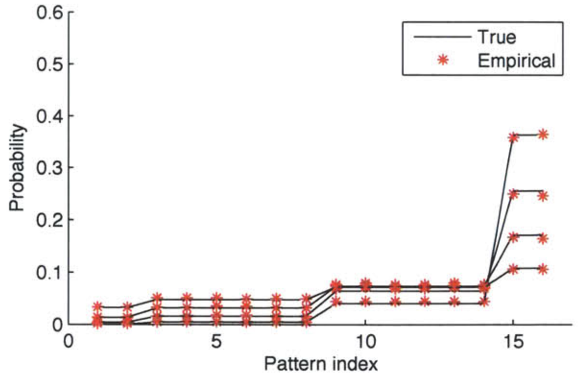
(a) single node



(b) size 2 neighbourhood



(c) size 3 neighbourhood



(d) size 4 neighbourhood

Figure 6-6: Single loop source: for different local structures, the comparison of theoretical and empirical probabilities of different patterns.

True Flipping Rate p	0.1000	0.2000	0.3000	0.4000
Estimated Flipping Rate	0.1004	0.1998	0.3083	0.4062

Table 6.6: Single loop source: estimation of flipping probability p.

6.1.3 Ising Model

Schematic of 2-dimensional Ising model is shown in Figure 6-7.

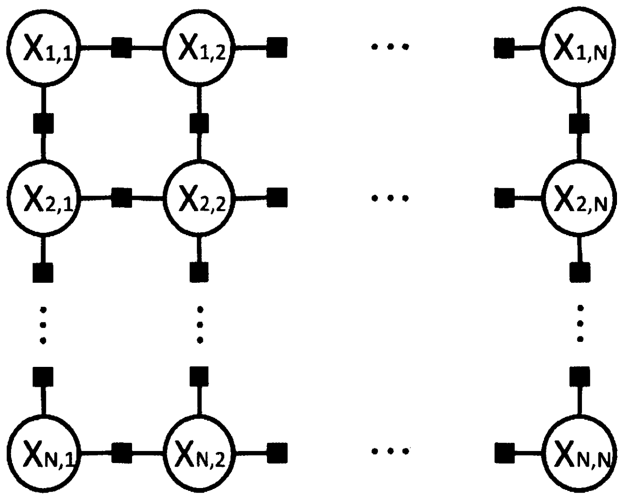


Figure 6-7: Schematic of Ising model.

The Ising model considered here is also homogeneous and symmetric about 0 and 1. $\forall i, j$ s.t. $(i, j) \in \mathcal{E}$,

$$\psi_{x_i,x_j} = \begin{cases} p & \text{if } x_i \neq x_j \\ 1-p & \text{if } x_i = x_j \end{cases}$$

where $\mathcal{E}$ is the set of all edges in the grid and $x_i \in \{0, 1\}$. Moreover, the Ising model is assumed to have periodic boundary conditions to ensure fair comparison with analytical results from Onsager's exact solution. The simulation parameters are listed in Table 6.7.

Unlike in the case of Markov Chain sources and Single Loop sources, 2D Ising model cannot be sampled exactly. The most popular sampling methods used in practice are Markov Chain Monte Carlo methods. Thus for Ising model, the samples from the proposed sampling scheme will be compared with those from Markov Chain Monte Carlo methods. In particular, a Gibbs sampler with a burn-in period of 100000 iterations is used. One sample is produced out of each 50000 iterations. To match the number of samples from proposed scheme, 1000 samples are obtained from Gibbs sampler by running 100 Markov chains in parallel, i.e. each chain produces 10 samples.

Number of Variables Each Column N_1	10
Number of Variables Each Row N_2	10
Number of Repetitions of the Source t	10
Probability of Flipping p	0.1/0.2/0.3/0.4
Number of Maximum Iterations in Belief Propagation algorithm	200
Doping Rate	0.05
Total Number of Samples	~1000

Table 6.7: Parameters for 2D Ising model source.

Three tests are carried out to test the quality of obtained samples:

1. Estimation of flipping probability p.

2. Estimation of patterns' probabilities in local structures.

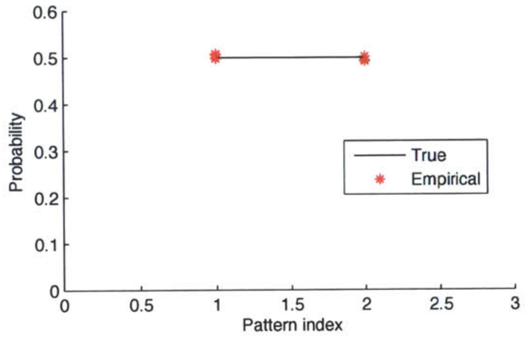
3. Estimation of partition function.

Notice that for all three tests, the true value is available: flipping rate of the target distribution is a given parameter; probabilities of patterns for small local structures can be easily computed; and partition function of 2D homogeneous Ising model can be calculated via Onsager's exact solution (as discussed in Section 2.5). These tests will be carried out for samples from both the proposed sampling scheme and MCMC methods, and then the results are compared with the true values obtained analytically.

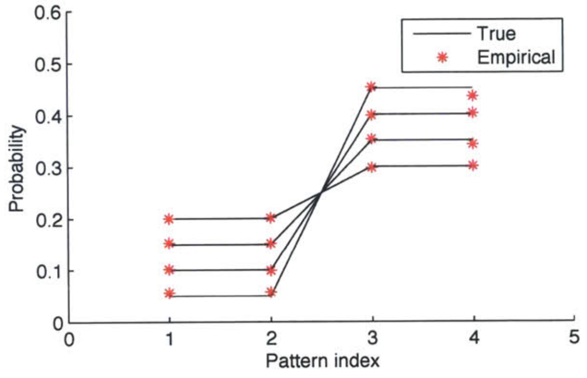
Firstly, estimations of flipping probability p are shown in Table 6.8.

True Flipping probability p	0.1000	0.2000	0.3000	0.4000
p estimated by LDPC-based Scheme	0.1074	0.1951	0.2998	0.3958
p estimated by Gibbs Sampler	0.0022	0.0284	0.2168	0.3917

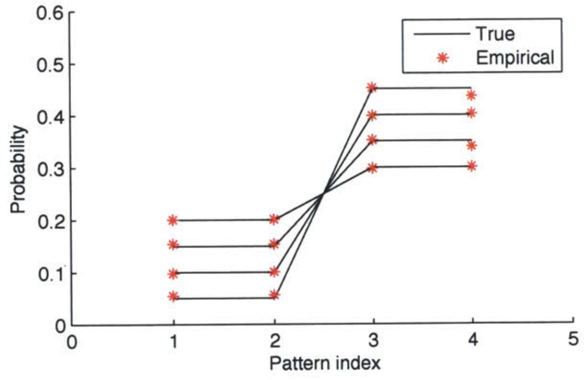
Table 6.8: 2D Ising model: estimation of flipping probability p.



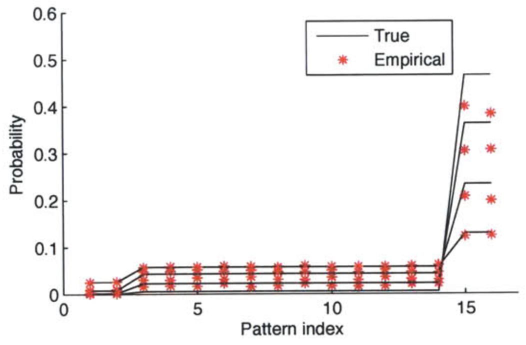
(a) single node



(b) size 1 × 2 subgraph



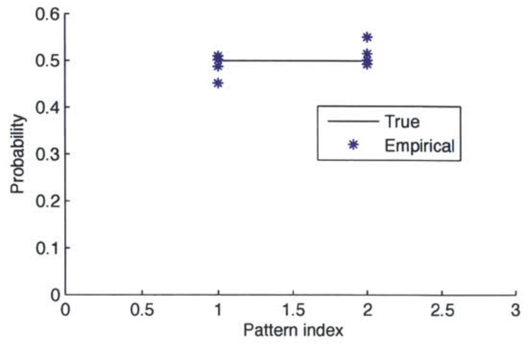
(c) size 2 × 1 subgraph



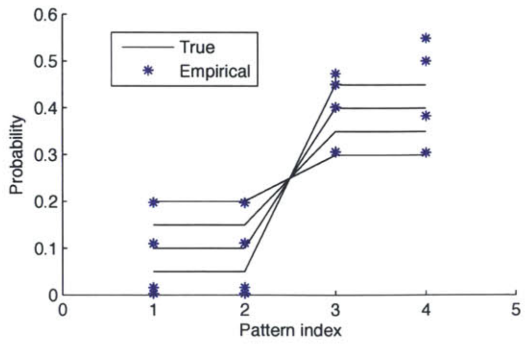
(d) size 2 × 2 subgraph

Figure 6-8: True and empirical marginals of local structures: Proposed Scheme.

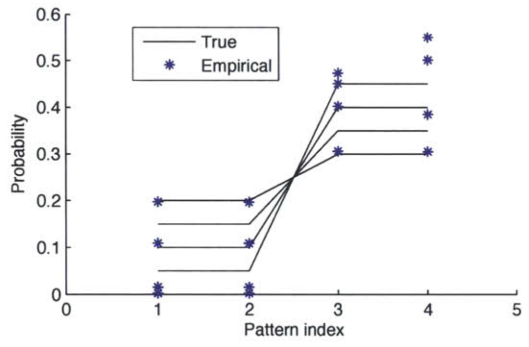
For the second test, estimation of patterns' probabilities in local structures are presented in Figure 6-8 and 6-9, for samples from LDPC-based scheme and Gibbs sampler respectively. The horizontal axis in the plots is the index of patterns and the vertical axis represents probability. Same as in Figure 6-3 and 6-6, the black curves are the true probabilities computed analytically, while the red and blue dots are the empirical probabilities, estimated from the proposed scheme (red ones) and Gibbs sampler (blue ones).



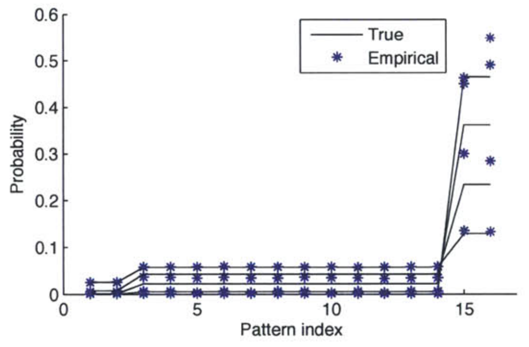
(a) single node



(b) size 1 × 2 subgraph



(c) size 2 × 1 subgraph



(d) size 2 × 2 subgraph

Figure 6-9: True and empirical marginals of local structures: Gibbs sampler.

It can be seen from Table 6.8 that samples produced using the proposed LDPC-sampling scheme correctly estimate the flipping rate for all p values, while samples from Gibbs sampler have very poor performance at low p values. Similar phenomena can be observed in Figure 6-8 and 6-9. Samples from Gibbs sampler produce poor estimations of pattern probabilities for local structures for $p = 0.1$, $p = 0.2$ and $p = 0.3$ (except in the single node case, which basically states that the model is symmetric about 0 and 1). Samples from LDPC-based scheme make very accurate estimation of pattern probabilities in the single node, 1×2 and 2×1 cases. The estimations in the 2×2 case with $p = 0.1$ and $p = 0.2$ are less satisfactory, but still outperform the Gibbs sampler samples.

It should be commented that although a very basic version of Gibbs sampler is used in our simulations, similar trends can be observed for more sophisticated variants of Gibbs sampling, or other Markov Chain Monte Carlo methods. This has to do with the existence of a phase transition of 2D Ising model. For the parameters used in our simulations, the phase transition[2] takes place at $p_c = 0.2929$. For $p < p_c$, the Ising model is in a so-called 'ferromagnetic' phase (in statistical physics, it is also interpreted as the system being at low temperature) and the Markov chains in Markov Chain Monte Carlo methods are likely to be trapped in a state that has relatively high probability and require impractically long time to mix.

The third test aims to estimate the partition function. To be more precise, assume the distribution defined by the 2D Ising model is

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{Z} p^{F(\mathbf{x})}(1-p)^{T(\mathbf{x})-F(\mathbf{x})}$$

where $F(\mathbf{x})$ is the number of flips in $(\mathbf{x})$, and $T(\mathbf{x})$ is the total number of neighbouring pairs. Thus $T(\mathbf{x}) - F(\mathbf{x})$ is the number of 'non-flips' (i.e. pairs of neighbouring variables that take the same value). The goal is to compute the partition function Z. Partition function has long been a quantity of interest in statistical physics[3] and is provably hard to calculate. It can be proved that performing inference on general graphs is polynomially reducible to computing the partition function of the corresponding graph. In other words, given an oracle that can compute the partition function for any given graph, one can come up with polynomial time algorithms to find marginals or most likely configuration.

In the case of homogeneous 2D Ising model, Onsager developed in [1] a solution to find out Z exactly, which is described in Section 2.5. It should be noted that in

[2]Computation of phase transition temperature can be found in [23].

[3]Many quantities of important physical meanings, such as free energy and canonical entropy, are defined as a function of Z. Details can be found in [23].

[1], the distribution is of the form

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{Z}\exp(-J \sum_{(i,j)\in\mathcal{E}} x_i x_j)$$

where $x_i \in \{-1, +1\}$ is the spin at location i and $\mathcal{E}$ is the set of all edges in the Ising model. If we set $J = \frac{1}{2}\ln(\frac{p}{1-p})$ and replace 0's with -1's in the grid, the two distributions are exactly the same, except that the partition functions differ by a constant factor. For convenience of presentation, the computations in this section stick to the distribution form $p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{Z}\exp(-J \sum_{(i,j)\in\mathcal{E}} x_i x_j)$.

An empirical distribution can be produced from a set of samples $\mathcal{SP} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(S)}\}$ and the corresponding partition function can be computed via

$$\tilde{Z} = \sum_{\mathbf{x}\in\mathcal{SP}} \exp(-J \sum_{(i,j)\in\mathcal{E}} x_i x_j)$$

If samples in $\mathcal{SP}$ are from exact sampling of 2D Ising model and the size of $\mathcal{SP}$ is large enough, the expected number of appearances in $\mathcal{SP}$ of a certain configuration $\mathbf{x}$ is roughly

$$\frac{|\mathcal{SP}|}{\#\text{ of possible configurations}} \times p_{\mathbf{x}}(\mathbf{x})$$

In such an ideal case,

$$\tilde{Z} \sim \frac{|\mathcal{SP}|}{\#\text{ of possible configurations}} \times Z$$

should hold. Notice the number of all possible configurations of the Ising model grows exponentially with the size of the model and for any practical simulation

$$|\mathcal{SP}| \ll \#\text{ of possible configurations}$$

Yet

$$\frac{\#\text{ of all possible configurations}}{|\mathcal{SP}|} \times \tilde{Z}$$

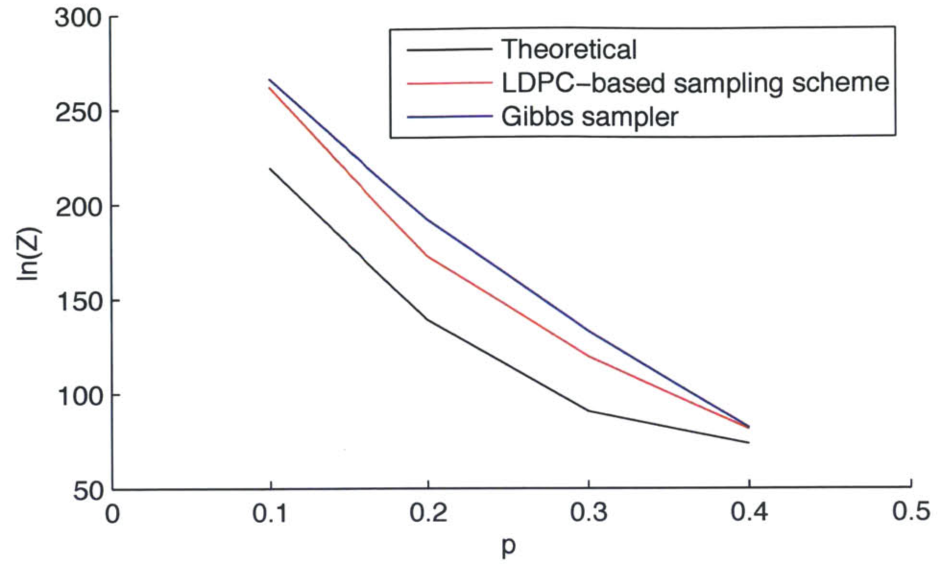


Figure 6-10: 2D Ising model: estimation of partition function Z.

is still used to approximate Z. The results are plotted in Figure 6-10. The horizontal axis represents the flipping probability p, and the vertical axis shows the value of the partition function Z. For clarity of presentation, $\ln(Z)$ is used in the plot. It can be seen that for all values of flipping probability p, the estimation of partition function using samples from LDPC-based scheme consistently outperform that using samples from Gibbs sampler. However, both estimations are higher than the true value computed using Onsager's exact solution by a significant amount.

One major reason for the over-estimation is that the sample set size ($\sim$1000) is only a tiny fraction of the number of all possible configurations (2^{100}). For many values of F, the expected number of samples in $\mathcal{SP}$ that have F flips is much less than 1, which means it is likely that $\mathcal{SP}$ does not contain such a sample at all. Notice these values of F are exactly the ones that result in low values of $\exp(-J \sum\limits_{(i,j)\in\mathcal{E}} x_i x_j)$. Approximating Z using $\frac{2^{100}}{1000} \times \tilde{Z}$ can be interpreted as estimating Z using a new set of samples, where each sample in $\mathcal{SP}$ is replicated $\frac{2^{100}}{1000}$ times. Thus essentially the higher values of $\exp(-J \sum\limits_{(i,j)\in\mathcal{E}} x_i x_j)$ are counted more than they should be, and the lower values are almost never counted. As a result, the estimated partition function is an over-estimate of the true value of Z.

6.2 Analysis of Simulation Results

Section 6.1 presented simulation results for the three target distributions considered and showed that the samples from the proposed sampling scheme are close representations of the target distributions. This section moves on to report and explain some interesting phenomena observed in the simulations. In particular, Section 6.2.1 focuses on describing various aspects of convergence behavior of the Belief Propagation algorithm, while Section 6.2.2 explores how check nodes' average degree impacts the performance of the sampling scheme.

6.2.1 Convergence Behavior of Belief Propagation algorithm

Recall that approximate simulation of the proposed scheme is achieved by running Belief Propagation algorithm on the combined graph. The combined graph is clearly a loopy one, thus the convergence behaviour is in general complicated. Inspired by EXIT function analysis of channel coding, [30] has developed a method for summarizing the evolution of messages for the same framework but with different setup. The theoretical analysis of Belief Propagation algorithm is beyond the scope of this thesis, but we aim to provide a detailed description of the convergence behaviour, along with some intuitive arguments.

The figures in this section are produced using simulation data from Markov Chain model with flipping probability $p = 0.1$. The corresponding entropy rate is ~ 0.45. When referring to 'low code rate', 'correct code rate' and 'high code rate', LDPC codes with rate around 0.2, 0.4, and 0.6 are used respectively. Data from other models and/or different values of flipping probabilities p exhibits similar trends in terms of convergence behaviors.

Below, convergence behavior, entropy of messages, fraction of unsatisfied checks, probabilities of the resulting sequence, and empirical code rate are discussed in turn.

Convergence Behavior

Briefly speaking, when the rate of the LDPC code is too low, Belief Propagation algorithm will converge, but the resulting sequence (obtained using symbol MAP decoding as described in Section 5.2) will not be a typical one of the target distribution. On the other hand, if the code rate is too high, the algorithm will fail to converge. There is some 'correct' code rate around which the algorithm will converge and produce a typical sequence.

Notice this is different from what was described in [29], where the same framework is used for compression. In [29], since the doping bits and parity check values are all computed using the same sequence (i.e the sequence to be compressed), the information provided is always consistent and a higher code rate will never do harm to the convergence of Bellief Propagation algorithm. In the proposed scheme, however, the doping bits and parity check values are generated independently, thus too much (randomly generated) information will likely become inconsistent and lead to confusion of Belief Propagation algorithm.

One way to look at the combined graph is that the variables form a boundary between the source graph and the code graph. The goal is to figure out what values the variables should take using information from two directions: source graph and code graph[4]. Consider the extreme case where there are no checks, then running Belief Propagation on the source graph will result in a 'most likely' sequence[5] for the given source. On the other extreme, if all variables are independent and symmetric about 0 and 1 (i.e. source graph does not contain any factors), Belief Propagation algorithm will converge to a sequence that satisfies all the checks defined in the code graph (i.e. a codeword). In general cases where both source graph and code graph exist, there

[4]The doping bits are viewed as degree-1 checks and grouped into the code graph.

[5]To be more precise, bit-wise most likely, since symbol MAP decoding is used.

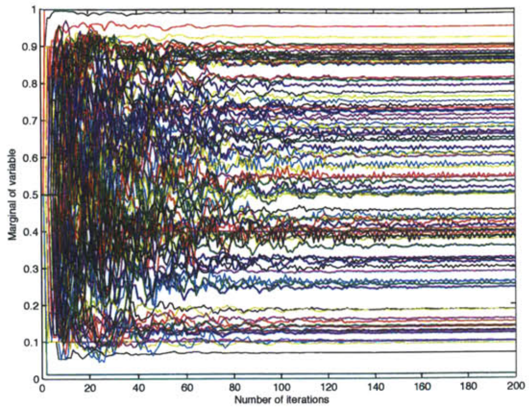
is a competition between the two.

When the code rate is too low, codewords are abundant and the algorithm will converge to a 'likely' (in bit error sense) sequence, which may not be typical. This case can also be regarded as information from the code graph being too weak.

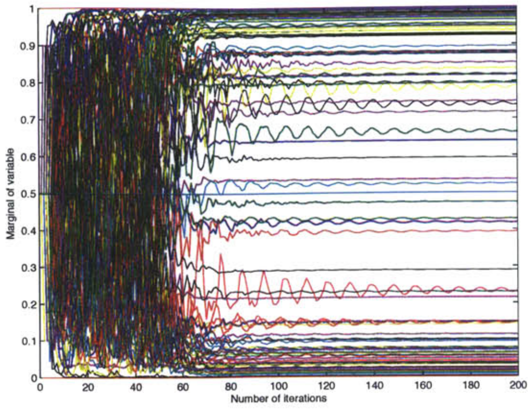
Conversely, when the code rate is too high, there are very few codewords. The information from source graph and code graph try to pull the variables in different directions and an agreement cannot be reached. As a result, Belief Propagation algorithm fails to converge.

The interesting case is when the code rate is roughly matched to the entropy rate of the source. In this case, discussion in Section 4.1 states that with high probability, there is a sequence that is both a codeword and a typical sequence of the target distribution. With well balanced information from source graph and code graph, Belief Propagation algorithm should be able to find that sequence.

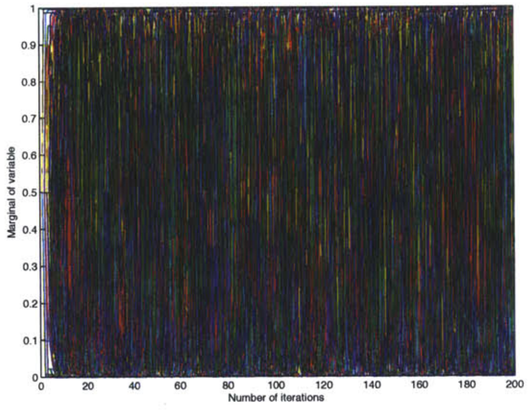
Figure 6-11 shows how the marginals of variables evolve with number of iterations of Belief Propagation algorithm, for different code rate. The horizontal axis represents the number of iterations performed, while the vertical axis is the marginal probabilities of variables (in particular, the probability that variable takes value 1 is plotted). For clarity of presentation, only a randomly chosen subset of the 1000 variables are displayed. As can be seen from the plots, when code rate is too low, the marginal estimates stop changing after some number of iterations but the beliefs of most variables are close to 0.5. In other words, the algorithm is still fairly unsure about most variables' values. When the correct code rate is used, the algorithm converges, and most beliefs are close to 0 or 1. When the code rate is too high, the algorithm fails to converge and the estimated marginals vary dramatically between iterations.



(a) low code rate



(b) 'correct' code rate



(c) high code rate

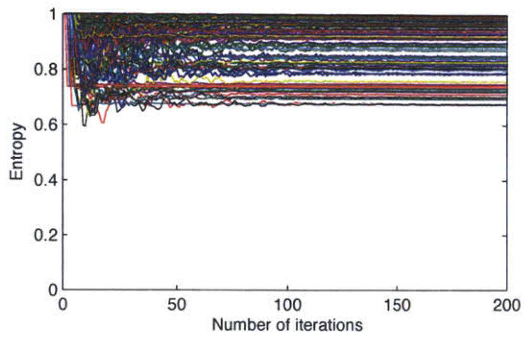
Figure 6-11: The variable marginals estimated from the Belief Propagation algorithm, as a function of number of iterations.

Entropy of Messages

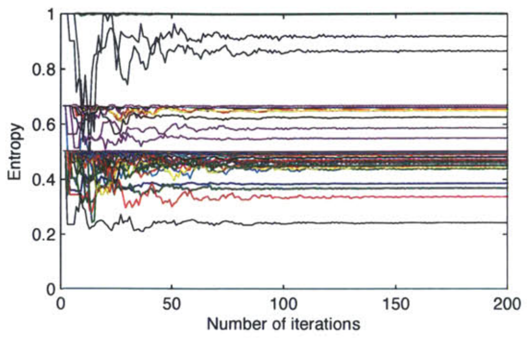
To further distinguish the impact from source graph and code graph, the entropy of messages coming into variables, both the factor-to-variable ones and the check-to-variable ones are computed for each node. Define $\mathcal{N}_{\mathcal{F}}(i)$ and $\mathcal{N}_{\mathcal{C}}(i)$ to be the set of all factor nodes and all check nodes connected to variable i respectively, then

$$H_{\mathcal{F}}^{(t)}(i) = \frac{1}{|\mathcal{N}_{\mathcal{F}}(i)|} \sum_{f \in \mathcal{N}_{\mathcal{F}}(i)} H(\mathbf{m}_{f \to i}^{(t)})$$
$$H_{\mathcal{C}}^{(t)}(i) = \frac{1}{|\mathcal{N}_{\mathcal{C}}(i)|} \sum_{c \in \mathcal{N}_{\mathcal{C}}(i)} H(\mathbf{m}_{c \to i}^{(t)})$$

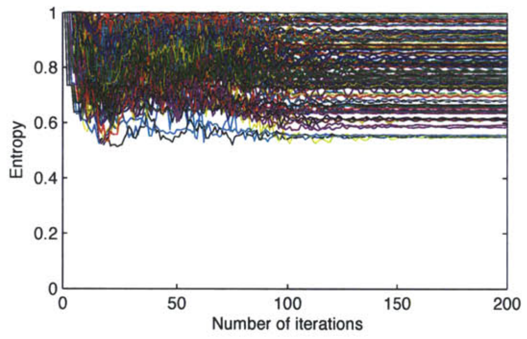
where $H(\mathbf{m}) = -m(0)\log_2(m(0)) - m(1)\log_2(m(1))$ is the binary entropy of normalized message $\mathbf{m} = [m(0), m(1)]^T$, $m(0) + m(1) = 1$.



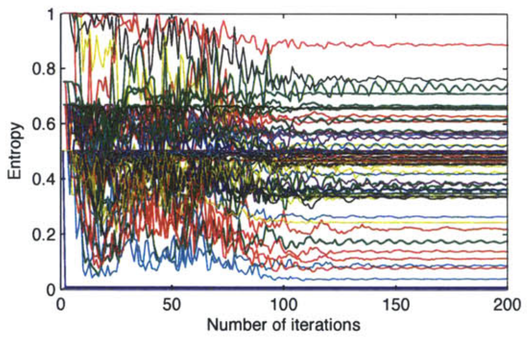
(a) low code rate: source graph



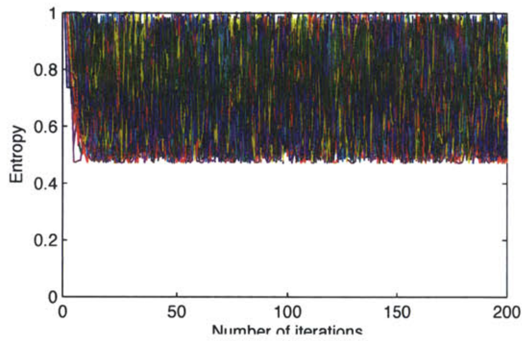
(b) low code rate: code graph



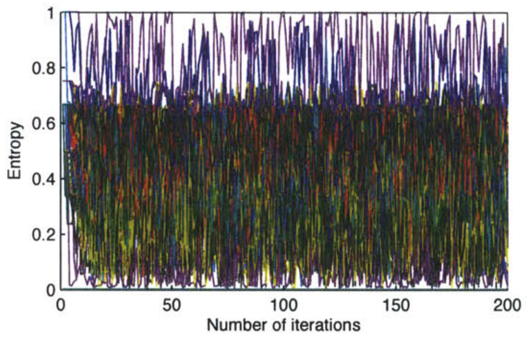
(c) 'correct' code rate: source graph



(d) 'correct' code rate: code graph



(e) high code rate: source graph



(f) high code rate: code graph

Figure 6-12: Entropy of messages from source graph and code graph.

Figure 6-12 shows how the message entropies look like for different values of code rate. In all the plots in Figure 6-12, the horizontal axis shows the number of iterations and the vertical axis shows entropy. A message with entropy close to 1 indicates 'ignorance', i.e. the message is close to $[0.5, 0.5]^T$ and does not have much preference about what value the variable should take, while a message with entropy close to 0 is an 'information-rich' one and relatively sure about the corresponding value.

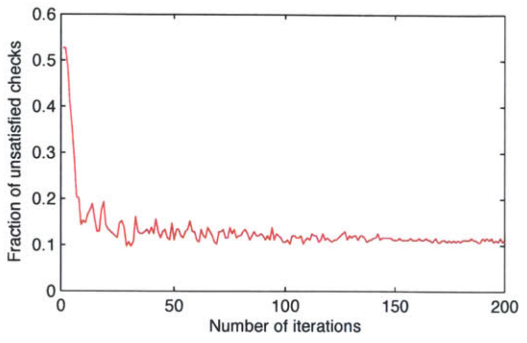
It should be commented that given the way the target distributions are defined, they are symmetric about 0 and 1. Thus for any given variable, without any external information, the source graph does not have preferences its value. However, if the value of a variable is known (either given explicitly or via a non-uniform marginal), the source graph will have opinions about the values of its neighbouring variables.

It can be seen from Figure 6-12 that with low code rate, although the code graph side messages are reasonably 'information rich', there are too few of them. As a result, the source graph is still very uncertain about many variables' values. In other words, the entropy of messages from source graph are close to 1. With the correct code rate, the entropy of messages from both code graph and source graph are reasonably away from 1. With high code rate, Belief Propagation algorithm fails to converge and the entropy of messages varies dramatically from iteration to iteration, as expected.

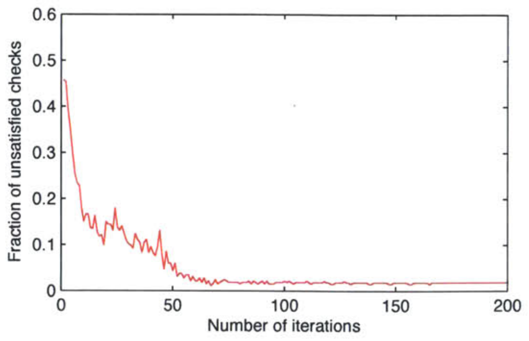
Proportion of Unsatisfied Checks

Recall that the goal of code graph is to find a codeword, i.e. make all checks satisfied. Figure 6-13 examines how close the resulting sequence is to a codeword, measured by fraction of checks that are unsatisfied (out of all checks).

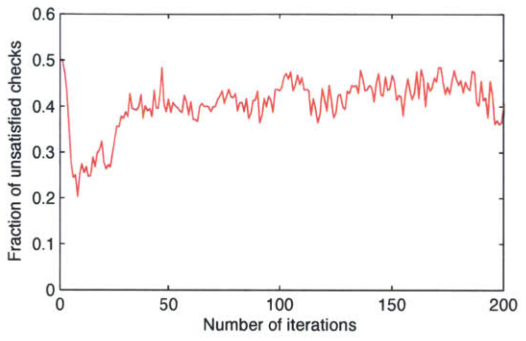
With low code rate, the impact from code graph is dominated by that from source graph and not all checks can be satisfied. With correct code rate, almost all checks are satisfied. With high code rate, impact from code graph is strong, but likely to be conflicting, either among themselves and/or with information from source graph, leaving the Belief Propagation algorithm confused and not converging. Thus a big portion of the checks are unsatisfied. Notice in the plot for high code rate, there is an initial drop in fraction of unsatisfied checks. This is mainly due to the fact that information comes in from code graph first (through doping bits). It takes some



(a) low code rate



(b) 'correct' code rate



(c) high code rate

Figure 6-13: The fraction of unsatisfied checks as a function of number of iterations.

time for the information to propagate through the source graph. In this period code graph will have stronger influence, and thus the fraction of unsatisfied checks is lower.

It is worth mentioning here that in practice, since it is impossible to have very fine scale control of the competition between source graph and code graph, even with correct code rate, it often so happens that a small portion ($< 5\%$, usually $1{\sim}2\%$) of the checks are still unsatisfied at convergence. Recall that the fixed points of Belief Propagation algorithm on loopy graph are local extrema of the corresponding Bethe Approximation problem [13]. Since the distribution defined by the combined graph is expected to be complicated, it should not be surprising when the algorithm gets stuck in a local extrema.

Figure 6-14 shows a typical situation where a variable gets disagreeing information from the source graph and the code graph when the Belief Propagation algorithm has converged. Note that the code graph wants this bit to be 0 while the source graph wants it to be 1, In this particular case, source code wins and results in an unsatisfied check. However, further investigation shows that all checks can be satisfied by flipping only a few bits of the resulting sequence. In other words, the resulting sequence is a 'near-codeword' and the effect on the obtained samples should be negligible.

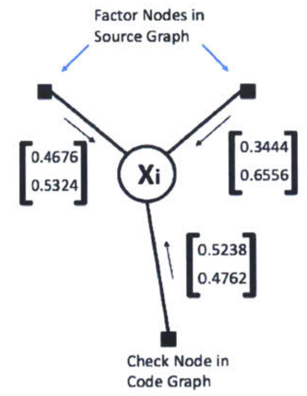


Figure 6-14: Source and code graph provide conflicting information to the variable.

Probability of the Resulting Sequence

Figure 6-15 shows how the probability of the sequence under the target distribution evolves with number of iterations of Belief Propagation algorithm. Here, log of the probability is used for clarity of illustration. Horizontal axis represents number of iterations and vertical axis shows log of the probability. The blue curve indicates the log probability of a typical sequence of the given distribution.

When the rate is too low, there are many more codewords than typical sequences so the algorithm finds a sequence more probable than a typical sequence (purple curve in Figure 6-15). Conversely, when the rate is too high, there are very few codewords compared to typical sequences and it is simply too difficult to find one. Moreover, in search of a codeword, the influence from code graph makes the resulting sequence have probability much lower than a typical sequence (red curve). At a suitable rate, there is one sequence that is both a 'near codeword' and typical to the source with high probability. Belief Propagation algorithm will converge to it (green curve).

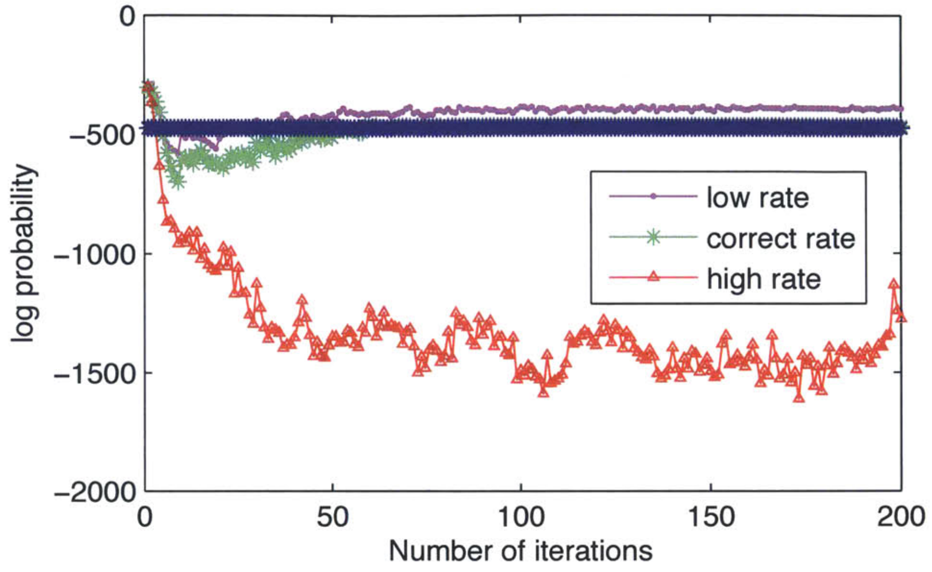


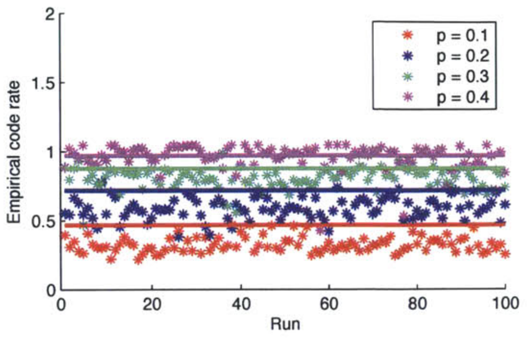
Figure 6-15: log probability of obtained sequence as a function of iteration number.

Empirical Rate of LDPC Codes

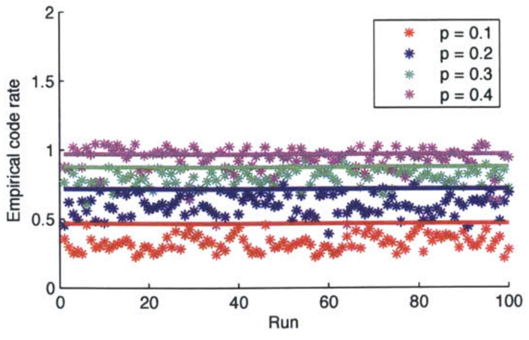
Figure 6-16 shows the comparison between the actual code rate used by the simulations and the theoretical entropy rate computed analytically. The proposed sampling is run 100 times to produce the 1000 samples required, and the empirical code rate varies among different runs. In Figure 6-16, the horizontal axis indexes which run it is and the vertical axis shows the rate. The red, blue, green and purple dots represent $p = 0.1$, 0.2, 0.3, and 0.4 respectively, and the lines with matching colors show the corresponding entropy rates.

It can be seen that for Markov chain source and single loop source, the empirical values track the entropy rate reasonably well, but are slightly lower in a consistent manner. Recall from Section 4.1 that the 'correct' rate is

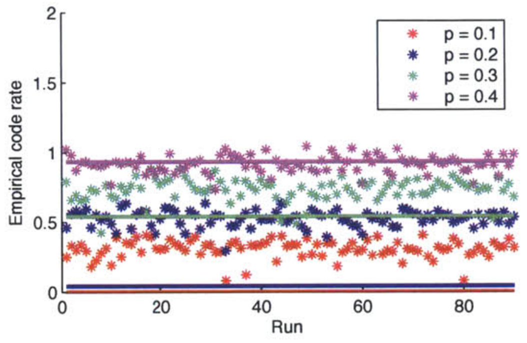
$$\frac{t}{Nt}(H_{p_{\mathbf{x}}} - \eta)$$



(a) low code rate



(b) 'correct' code rate



(c) high code rate

Figure 6-16: Empirical rate of LDPC code used in the proposed scheme.

where $\eta = -\delta \sum_{a \in \mathcal{V}} \log(p_{\mathsf{v}}(a)) > 0$. Therefore the gap has its theoretical justification.

In the case of Ising model and relatively small flipping probability p, the actual code rate used in simulations is much higher than the corresponding entropy rate. This can be partially explained by the fact that the factor graph describing Ising model contains a large number of closely tied loops and thus very sensitive to external information. In fact, any external information, weak it may be, will be significantly amplified in Ising model by running Belief Propagation algorithm. The source graph can be regarded as the strong side in the competition, which in some sense makes convergence easier. To put it another way, Ising model can tolerate more randomness when running Belief Propagation algorithm. More details on this can be found in Section 6.2.2.

6.2.2 Check Degree and Sample Quality

Section 6.2.1 detailed how different code rate can lead to different behaviors of the proposed sampling scheme. It is observed from the simulations that in the competition between source graph and code graph, rate is not the only factor that has impact. Empirically it is observed that average check node degree also influences the effectiveness of information from code graph, and thus affects the quality of samples.

This section will discuss how check nodes' average degree can affect performance of the LDPC-based sampling scheme, as well as present an empirical method to find the correct degree.

Sample Quality As a Function of Degree Distribution

Consider the messages update equations at a check node, as shown in Figure 6-17. Without loss of generality, assume the parity value associated with the check is 0.

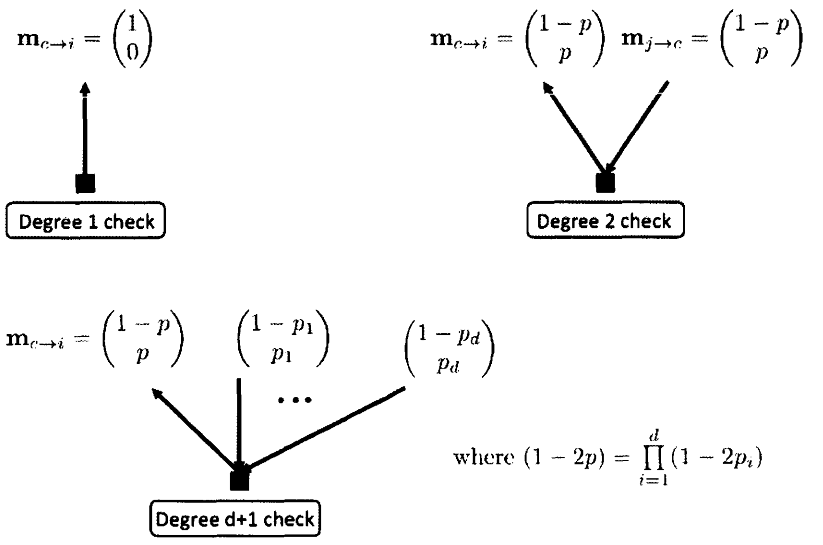

$\mathbf{m}_{c\to i} = \begin{pmatrix}1\\0\end{pmatrix}$
Degree 1 check
$\mathbf{m}_{c\to i} = \begin{pmatrix}1-p\\p\end{pmatrix}$ $\mathbf{m}_{j\to c} = \begin{pmatrix}1-p\\p\end{pmatrix}$
Degree 2 check
$\mathbf{m}_{c\to i} = \begin{pmatrix}1-p\\p\end{pmatrix}$ $\begin{pmatrix}1-p_1\\p_1\end{pmatrix}$ ... $\begin{pmatrix}1-p_d\\p_d\end{pmatrix}$
where $(1-2p) = \prod_{i=1}^{d}(1-2p_i)$
Degree d+1 check

Figure 6-17: Message update equations at a check node

Notice $(1-2p) = \prod_{i=1}^{d}(1-2p_i) \Rightarrow |1-2p| = \prod_{i=1}^{d}|1-2p_i|$ and $|1-2p_i| \leq 1, \forall i \in \{1,2,...,d\}$

$$\therefore |1-2p| \leq |1-2p_i|, \forall i \in \{1, 2, ..., d\}$$

That is, if one of the incoming messages to the check is uncertain (i.e. the message is close to $[0.5, 0.5]^T$), the outgoing messages are even less certain. When the degree of a check is higher, it is more likely to have an uncertain incoming message, which in turn leads to weak outgoing messages. As a result, the source graph can dominate the competition easily. The intuition is clear when one considers extreme cases: given the same number of checks, i.e. the same amount of randomness, if all of them are degree 1 checks, the values of the corresponding variables are directly determined regardless of what the source graph says; on the other hand, if all checks have very high degrees such that each check has uninformative incoming message ($[0.5, 0.5]^T$), then the messages from checks to variables will be uninformative. The actual situation is something in between the extreme cases, yet the general trend is true: the smaller a check node's degree is, the stronger/more informative its outgoing messages will be.

Assuming a regular LDPC code, how does the behaviour of the proposed sampling technique vary as a function of the check nodes degree? The quantities of interest include:

1. The empirical distribution of number of flips in each sample, F, and the corresponding average.
2. After convergence, how certain Belief Propagation algorithm is about the variables' values. This certainty is measured using 'confidence', defined as the gap between estimated marginal and 0.5, averaged over all variables.[6]
3. Proportion of unsatisfied checks when at convergence of Belief Propagation algorithm.

These quantities will be examined in turn. It should be noted that figures in this

[6]confidence = 0.5 when the algorithm is entirely certain about the value of each variable and all marginals take value $[0, 1]^T$ or $[1, 0]^T$. confidence = 0 when the algorithm is totally ignorant of the variables' values, i.e. all marginals take the value $[0.5, 0.5]^T$.

section are produced using experiments with the Markov chain model, as defined in Section 6.1.1, with flipping rate $p = 0.1$. But the trends described hold for other sources and other parameters as well.

To begin with, Figure 6-18 examines the empirical distribution of number of flips F and Table 6.9 shows the corresponding average. Recall that the actual flipping rate here is $p = 0.1$. The horizontal axis in Figure 6-18 shows number of flips in the Markov chain and vertical axis shows probability. Curves of different colors represent empirical distribution estimated using different check node degree, as shown by the legends.

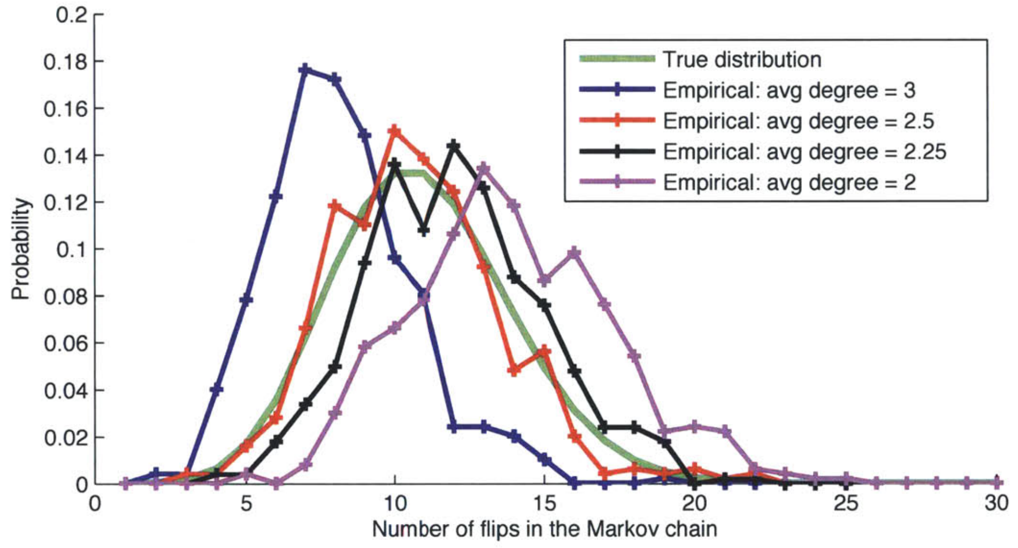


Figure 6-18: distribution plot for different values of average check degree

Average Check Degree	3	2.5	2.25	2
Estimation of Flipping Rate p	0.0722	0.0981	0.1106	0.1302

Table 6.9: Estimation of flipping probability for varying average check degree.

It can be seen that the higher the average check degree, the lower the number of flips in each sample. This is in agreement with the intuitive arguments discussed before. As the average check degree increases, the messages from code graph becomes weaker/less informative, thus the source graph gains more advantage in the competition. Given the way the source graph factors are defined ($p < 0.5$), the source graph prefers neighbouring variables to have the same value. Thus with higher check degree, samples obtained contain fewer flips.

Next we consider the confidence level. In Figure 6-19, horizontal axis shows the average degree of check nodes, while vertical axis shows the confidence. Different dots represent different runs of the proposed scheme. As shown in Figure 6-19, the confidence level decreases slightly with increasing average check degree. It should be noted that if Belief Propagation algorithm does converge, the confidence level can be viewed as a measure of how much randomness is introduced into the combined graph from the code graph side: the higher the confidence, the more randomness is effectively introduced. Thus the trend in Figure 6-19 is in accordance with intuitive arguments: with higher average check degree, the messages from the code graph is weaker and less effective in introducing randomness.

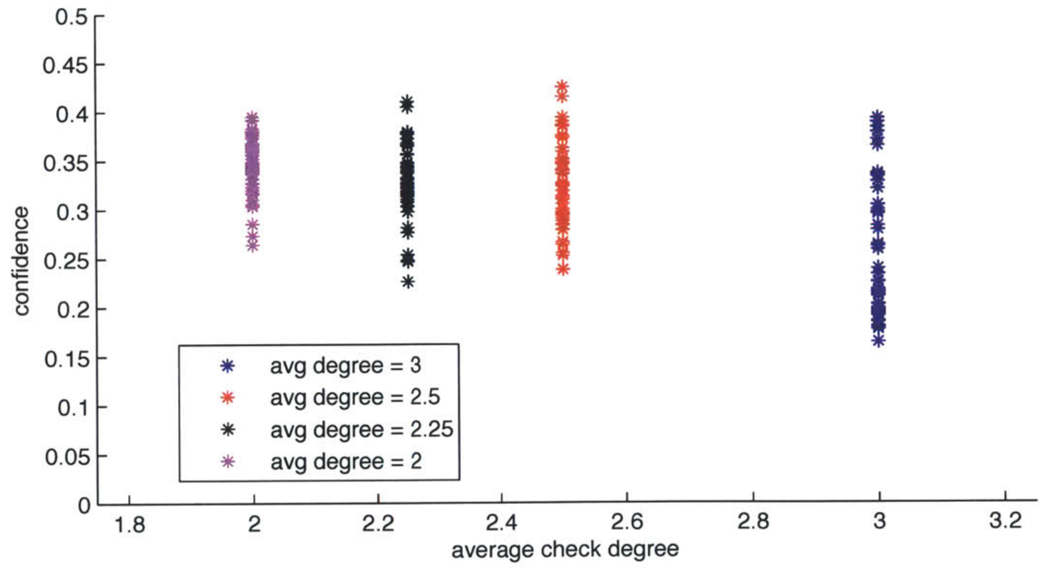


Figure 6-19: Confidence level for different values of average check degree

Finally, Figure 6-20 depicts how the fraction of unsatisfied checks at convergence changes with average check degree. Horizontal axis again shows the average check node degree and vertical axis shows the fraction of unsatisfied checks. Following a similar argument, Figure 6-20 makes intuitive sense: with higher average check degree, the code graph is the weaker side in the competition and thus a bigger fraction of checks is unsatisfied at convergence.

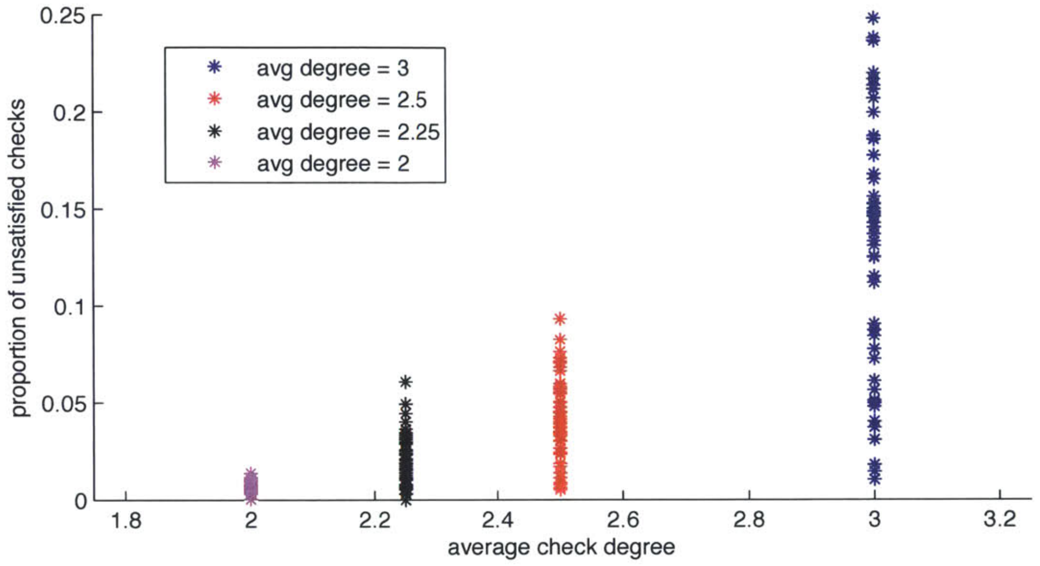


Figure 6-20: Fraction of unsatisfied checks for different values of average check degree

As is evident from Figure 6-18, for the purpose of sampling, there is a range of good average check degrees that will result in samples that can approximate the source distribution nicely. The problem now becomes how to find this range. An empirical method that works well in practice is discussed as follows.

Sensitivity to External Information

The basic idea is to match the strength of messages from code graph to the sensitivity of the source graph. Depending on the structure, different factor graphs can respond to the same external information in very different manners. When the source graph contains loops, Belief Propagation algorithm will cause amplification of the message, i.e. a message will be counted more than once through different paths but the algorithm still treat them as information from independent sources. The amplification effect will be more severe when smaller loops exist, and even worse when the small loops are closely tied.

To demonstrate this, several toy examples are studied below: size 3 loop, size 6 loop, and size 3×3 Ising model. In particular, the sensitivity to external information of the toy examples are examined by associating a biased node potential to a single chosen variable and running Belief Propagation algorithm on the corresponding factor graph. It can be proved that Belief Propagation algorithm will converge and the quantities of interest are the variables' marginals estimated at convergence.

<u>**Example 1: size 3 loop**</u>

The schematic of a size 3 loop is shown in Figure 6-21. The loop is homogeneous and symmetric about 0 and 1. There are two parameters involved: flipping probability p controls the coupling of neighbouring variables and q indicates how strong the external information is.

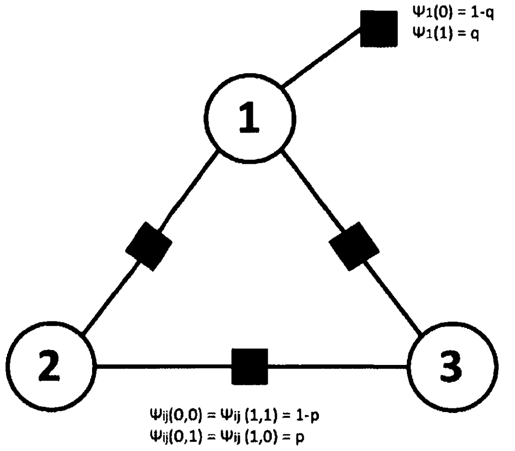
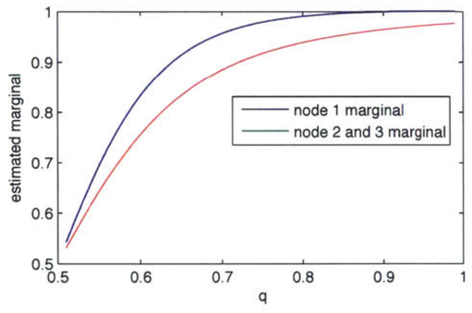
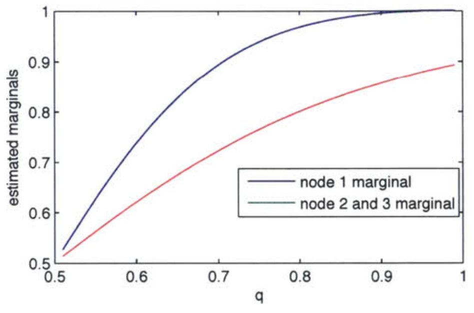


Figure 6-21: Schematic of a size-3 loop with external message on one variable.

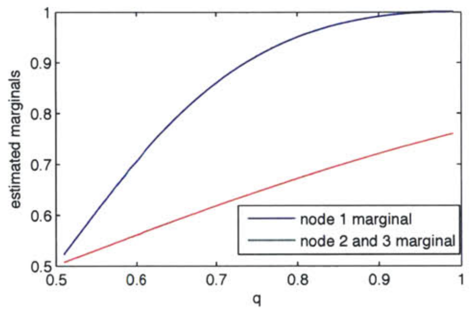
Plots in Figure 6-22 are produced by running Belief Propagation algorithm until convergence for each given pair of p and q. External message q varies from 0.51 to 0.99, in steps of 0.01. Horizontal axis shows the value of q and vertical axis shows the estimated marginal. As expected, with increasing q (i.e. the external information becomes stronger), the final belief produced by Belief Propagation algorithm is higher. With increasing p, the variables are less coupled to each other (notice if $p = 0.5$, the 3 variables are independent), and thus the influence caused by external information on the other variables becomes weaker. Finally, it should be noted that there are only two lines for three variables due to the symmetry of the source (two lines are on top of each other).



(a) $p = 0.1$



(b) $p = 0.2$



(c) $p = 0.3$

Figure 6-22: The sensitivity to external information for a size 3 loop.

Example 2: size 6 loop

The schematic of a size 6 loop is shown in Figure 6-23. Again, the model is homogeneous and symmetric about 0 and 1.

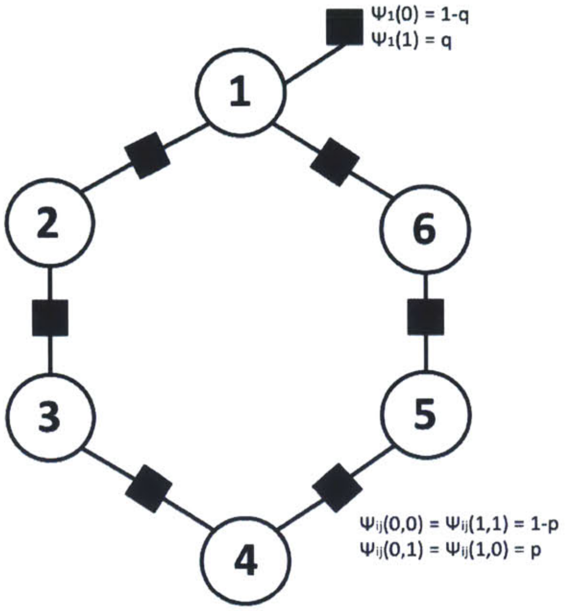
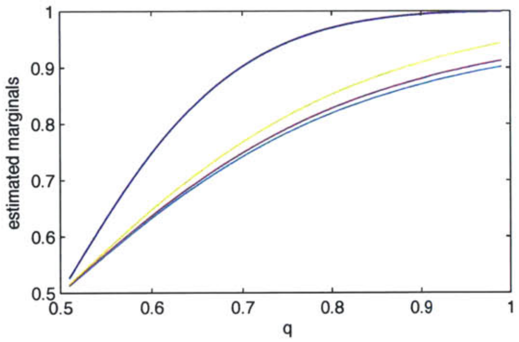
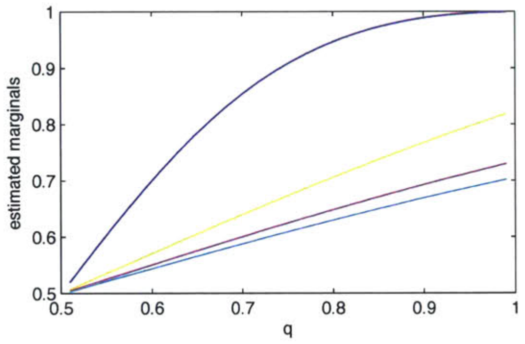


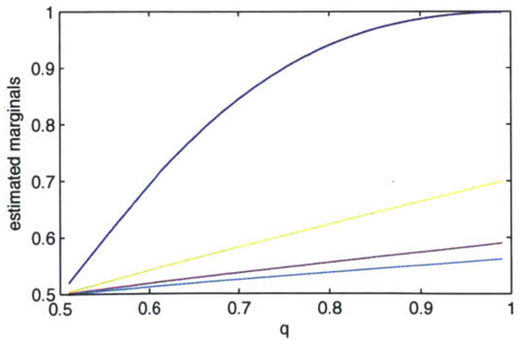
Figure 6-23: Schematic of a size-6 loop with external message on one variable.



(a) $p = 0.1$



(b) $p = 0.2$



(c) $p = 0.3$

Figure 6-24: The sensitivity to external information for a size 6 loop.

With varying p and q, Figure 6-24 shows that the sensitivity to external information exhibits similar trends as in size-3 loop. However, since the loop is larger and the graph more locally tree-like, the response is weaker than in a size-3 loop given external messages with the same strength.

When there are closely coupled small loops, the behavior will be more dramatic: the source becomes very sensitive to weak external messages. This is well illustrated by the third example below:

Example 3: size 3×3 Ising model

The schematic of the 3×3 Ising model is shown in Figure 6-25.

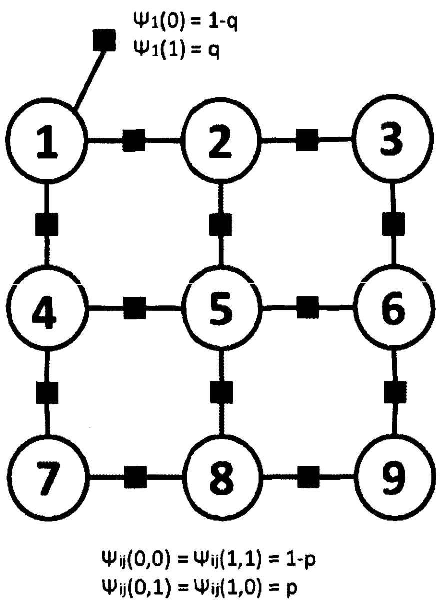
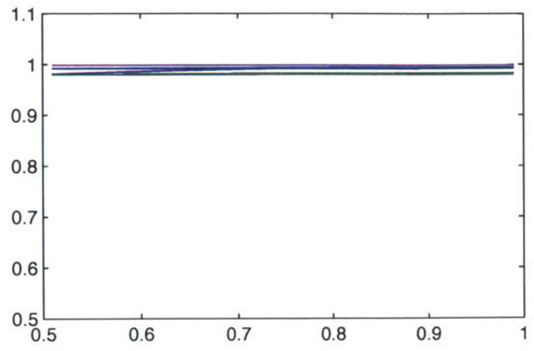
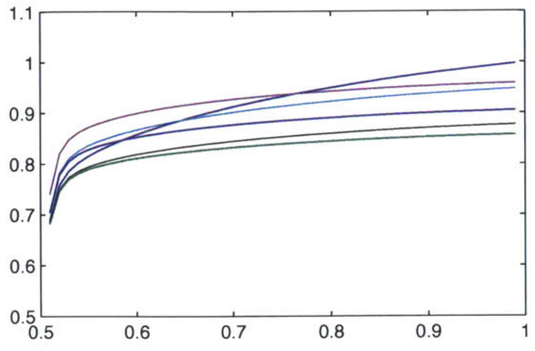


Figure 6-25: Schematic of a 3×3 Ising model with external message on one variable.

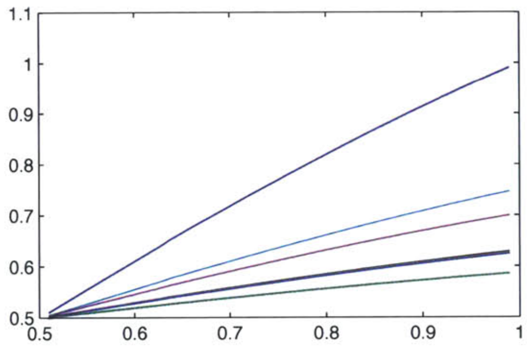
As can be seen from Figure 6-26, for small p (i.e. strong coupling between neighbouring variables), the estimated marginals of variables in Ising model respond dramatically even to weak external information. This is mainly due to the severe amplification of messages introduced by closely coupled size-4 loops.



(a) $p = 0.1$



(b) $p = 0.2$



(c) $p = 0.3$

Figure 6-26: The sensitivity to external information for a 3×3 Ising model.

Intuitively, if the source is sensitive to weak external messages, it would be ok to have weak messages from the code graph, which translates to the fact that the average check degree should be relatively large in the code graph. Conversely, if the source only responds to stronger external messages, the code graph messages should be strong and direct, thus the average check degree should be smaller.

In practice, what can be done is to carefully examine a few typical examples to calibrate the correct range of average check degrees that will produce good samples: ones that approximate the corresponding target distribution nicely. Then these examples can be used as references: given a new target distribution, its sensitivity to external information will be compared with the references to determine a range that is expected to work well for the new distribution.

Chapter 7

Further Discussion

The LDPC-based sampling scheme has been the focus of the previous chapters: Chapter 3 provides its formal description; Chapter 4 proves its theoretical guarantee of correctness; Chapters 5 and 6 discuss the practical implementation and report the simulation results. This chapter takes a step back and considers how the proposed sampling scheme fits into the big picture.

One natural question to ask is, how does the proposed sampling scheme compare to other existing sampling methods? What are the relative advantages and disadvantages? Section 7.1 attempts to compare the various aspects of different sampling methods and provide advices on how to choose the right sampling method depending on the specific application.

In the simulations, the LDPC codes are assumed to have (almost) constant check degree and generated in the way described in Section 5.1. This is easy to implement, but it is clearly not the only way to generate LDPC codes. Nor do we claim it to be the optimal way. In Section 7.2, a few other possibilities for LDPC code generation are briefly touched upon.

7.1 Choose the Right Sampling Method

Estimating marginals of a general distribution $p_{\mathbf{x}}(\cdot)$ is NP-hard. However, if some sampling method can produce independent samples from $p_{\mathbf{x}}(\cdot)$, the marginals can be approximated arbitrarily closely in polynomial time[1]. Thus one should not expect the existence of such a sampling method that runs in polynomial time.

Different sampling schemes have different goals as priority while making compromises in other aspects. Depending on the actual application, one or another sampling method may be the most suitable. This section attempts to discuss how to choose the right sampling technique for various applications.

When one is looking for samples from a certain distribution, the following properties are usually on the wish list:

1. *Cost of the sampling method.* Computational cost, both in space and time, should be tractable. Clearly, the more efficiently the samples can be produced, the better.

2. *Quality of the samples.* The empirical distribution of the samples should be a good approximation of the target distribution. Correlation between successive samples should be small because it determines the number of samples needed for Law of Large Numbers. Ideally, independent samples are desired.

3. *Theoretical guarantee of the quality of samples.* It is desirable to be able to prove either the correctness of the samples, or bounds on the number of samples needed to approximate the distribution to some certain accuracy.

4. *Randomness utilization.* On average, how many random bits are necessary to generate one sample? How does it compare to the minimum amount of

[1]To be more precise, let us denote the estimation of $p(x_i = a)$ as $\bar{p}_{x_i=a}$, then

$$\mathbb{P}(|p(x_i = a) - \bar{p}_{x_i=a}| > \epsilon) < \delta$$

when we have $O(\frac{1}{\delta\epsilon^2})$ iid samples.

randomness required in theory? Generating random bits may be expensive, and in such cases, economic use of randomness would be desirable.

As discussed above, it is unrealistic to hope all the listed requirements to be satisfied by one single sampling method. The trade-off will be made depending on the characteristics of the specific application in mind:

1. Is the target distribution over discrete variables or continuous variables?
2. Is the target distribution over high-dimensional space or low-dimensional space?
3. How are the samples going to be used? Do the actual samples matter or one only cares about estimating the expected value of a certain function?
4. What does the distribution look like? Distribution with certain features will be particularly undesirable for some sampling methods. For example, Markov Chain Monte Carlo methods will fail when there are islands of high-probability states connected by low-probability states in between.

Below, the sampling methods discussions in Section 2.1 as well as the proposed LDPC-based sampling scheme are examined in turn to understand under what circumstances each of them is useful.

Rejection Sampling

The samples from Rejection Sampling method are provably correct and successive samples are independent. However, computational cost of the technique depends on the rejection rate, which in turn depends on how close the proposal distribution $q_{\mathbf{x}}(\cdot)$ is to the target distribution $p_{\mathbf{x}}(\cdot)$. The situation where $q_{\mathbf{x}}(\cdot)$ has low probability at places where $p_{\mathbf{x}}(\cdot)$ has high probability is particularly undesirable.

Rejection Sampling can deal with continuous or discrete variables. Technically, Rejection Sampling can be applied to higher dimensional $\mathbf{x}$, but the acceptance rate[2]

[2]Acceptance rate = 1 - Rejection rate

will decrease exponentially with the number of dimensions.

Random bits are necessary each time a sample is generated from the proposal distribution, regardless of whether it is accepted as a sample of the target distribution. Moreover, the randomness cannot be re-used, otherwise the independence among samples can no longer be guaranteed. Thus when the rejection rate is high, the utilization of randomness in Rejection Sampling can be very inefficient. Many random bits are necessary to produce one single sample of the target distribution.

Importance Sampling

Importance Sampling estimates the value of a certain expectation (with respect to the target distribution) without producing samples from the target distribution. The estimation is provably correct in the limit, but similar to Rejection Sampling, the actual computational cost of obtaining a reasonable estimate depends heavily on the quality of the proposal distribution $q_{\mathsf{x}}(\cdot)$. If for some x, $p_{\mathsf{x}}(\cdot)$ is large while $q_{\mathsf{x}}(\cdot)$ is small, x is unlikely to come up as we are sampling from $q_{\mathsf{x}}(\cdot)$. Yet x is needed to accurately estimate the function value because it is 'important' for $p_{\mathsf{x}}(\cdot)$. Even worse, for Importance Sampling, it is difficult to diagnose how good the estimate is at any certain point and thus difficult to determine when to stop[3].

Importance Sampling can be applied to discrete or continuous variables. It is also worth noting that Sequential Monte Carlo method/Particle Filters can in some sense be viewed as a special case of Importance Sampling applied to HMM: the target distribution is $p_{\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_N|\mathsf{y}_1,\mathsf{y}_2,\ldots,\mathsf{y}_N}(\cdot|y_1,y_2,\ldots,y_N)$ while the proposal distribution is $p_{\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_N}(\cdot)$. The curse of dimensionality applies to Importance Sampling as well, due to the difficulty of finding a good proposal distribution in high-dimensional space.

Following a similar argument as in Rejection Sampling, independent random bits are

[3]In contrast, in Rejection Sampling, it is easy to know when to stop by counting the number of samples generated.

needed each time a sample is generated from proposal distribution $q_{\mathbf{x}}(\cdot)$. If $q_{\mathbf{x}}(\cdot)$ looks very different from target distribution $p_{\mathbf{x}}(\cdot)$, the weights will vary dramatically, and the estimation is dominated by a few terms. Thus a large number of samples (from proposal distribution) are needed to produce a good estimate. In those situations, utilization of randomness is low.

Markov Chain Monte Carlo Methods

In theory, Markov Chain Monte Carlo methods will produce samples from the target distribution as the number of iterations goes to infinity. They can deal with continuous or discrete variables, and scales well with dimensionality. However, several difficulties exist in practice:

1. It is non-trivial to determine how long the burn-in period is.

2. Since the successive samples from Markov Chain Monte Carlo methods are correlated[4], the task of extracting roughly independent samples from the sequence of samples is non-trivial. The effective number of samples is much smaller than the number of iterations run.

3. Depending on the target distribution, the Markov Chain Monte Carlo methods may easily get stuck and cannot explore the distribution very well.

In practice, heuristics are usually used to deal with the issues mentioned above. There are also variants of Markov Chain Monte Carlo methods that are specifically designed to deal with one or more of them. Examples include Block Gibbs Sampling which is designed to reduce correlation between consecutive samples, or using Simulated Annealing at early stage of Markov Chain Monte Carlo to reduce random walk behavior, etc. But these variants are still in the heuristic regime and do not necessarily work for all target distributions.

[4]The correlation is particularly high under Gibbs sampling since only one coordinate is changed at each iteration.

In terms of randomness utilization, the amount of randomness needed is closely coupled with number of iterations performed by the Markov Chain Monte Carlo method. The latter in turn depends on factors such as the length of burn-in period and correlation between successive samples, and thus is hard to quantify.

LDPC-based Sampling Scheme

For the sampling technique proposed in this thesis, the samples produced are provably from the target distribution assuming Maximum-likelihood simulation. In practice, however, Belief Propagation algorithm is implemented to perform approximate simulation and thus the correctness is only approximated. Samples from different runs of the scheme are independent. The samples in the same run are correlated (correlation introduced through the LDPC code), but form a sequence that is typical of the target distribution and thus provide good approximation to the target distribution .

The proposed scheme scales well with dimensionality. The computational cost is tractable[5] because the code graph is sparse and the number of iterations needed in the decoder does not grow significantly with the size of the combined graph. However, for moderate-size sources (e.g. $N = 100$, $t = 20$), the LDPC-based method takes considerably longer to produce the same number of samples than Markov Chain Monte Carlo methods[6].

The LDPC-based sampling method only applies to discrete variables, but can deal with arbitrary distributions over discrete variables without requiring a proposal distribution that approximates the target distribution. Moreover, it can deal with high dimensional $\mathbf{x}$.

The proposed scheme makes particularly efficient use of randomness. In theory, the

[5]roughly linear in t, the number of repetitions of source graph, and at most quadratic in N, the number of variables in the source.

[6]It is a bit tricky to provide quantitative comparison of computation times for the two methods because the samples produced by Markov Chain Monte Carlo methods are correlated. Finding the effective number of independent samples, as just discussed, is non-trivial task.

minimum amount of randomness per bit needed to produce samples from a target distribution $p_{\mathsf{x}}(\cdot)$ equals to the entropy rate $H(p_{\mathsf{x}})$. Each time the proposed scheme is run, the number of required random bits equals the number of parity checks plus the number of doping bits. Thus the average randomness per bit, as discussed before, roughly equals the entropy rate of $p_{\mathsf{x}}(\cdot)$. To be precise, extra randomness is needed

1. during the process of dynamically determining the 'correct' rate because an overestimate of the rate is initially picked;

2. to determine the positions of doping bits.

3. to generate the LDPC code;

The amount of randomness needed for item 1 and 2 above are linear in total number of variables in combined graph with small constants, while that for item 3 will be amortized. Thus the randomness needed by the proposed LDPC-based sampling scheme is very close to the theoretical lower bound.

It is worth mentioning here that for all the sampling techniques discussed so far, the goal is to approximate the target distribution and emphasis is on sampling the more likely events. For example, the LDPC-based scheme only look at typical sequences of the target distribution and explicitly ignores rare events. Probably none of them would do a good job in sampling rare events. There are sampling methods that specialize in sampling rare events, examples include Forward Flux Sampling[26], Transition Path Sampling[14] etc.

A Few Real-life Applications

Let us conclude Section 7.1 by a few real-life applications where sampling is needed and discuss what sampling method is suitable for each of them:

Example 1: Molecular Dynamics Simulation

In statistical physics, a common task is to compute some thermodynamic potentials such as free entropy or internal energy, which can be expressed as expectation of some function over the Boltzmann distribution[7]

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{Z}\exp(-\beta E(\mathbf{x}))$$

Exact computation is intractable except for a few special cases, thus some sort of sampling techniques is often used. Here the data is high-dimensional, usually discrete (particles usually take one of a finite number of possible states, e.g. spins take value +1 or -1), and people care mostly about estimating some functions instead of obtaining the actual samples.

In practice, some variant of Markov Chain Monte Carlo method is commonly used for such applications, mainly due to its ability to scale with dimensionality. The LDPC-based sampling technique proposed in this thesis is also well suited for approximating thermodynamic potentials. In particular, the Ising model discussed in Section 6.1.3 serves as a nice example.

Example 2: Max-Weight Independent Set

Given an undirected graph on which each node is associated with a positive weight, the Max-weight Independent Set (MWIS) problem aims to find the set of mutually non-adjacent nodes that has the largest total weight. The problem arises naturally in many scenarios. Examples include scheduling channel access and transmissions in wireless networks [24].

The problem is known to be NP-hard and can be hard to approximate [18]. One way to solve the problem is to put a distribution over all the independent sets: $p(\mathcal{I}) \propto \prod_{i\in\mathcal{I}} \lambda^{w_i}$, where $\lambda \geq 1$ is a constant. If this distribution $p(\mathcal{I})$ can be efficiently sampled from

[7]For example, free entropy is defined as $\log(Z(\beta))$ and internal energy is defined as $\mathbb{E}_{p_{\mathbf{x}}(\cdot)}[E(\mathbf{x})]$

for large enough λ, the MWIS problem can be solved with high probability because independent sets with higher total weights are more likely to appear.

Again, the data involved is high-dimensional and discrete, thus Markov Chain Monte Carlo is a common choice. However, [25] proves that there exists threshold on λ above which it will provably take exponential time for the Markov Chain to mix. Due to the discrete nature of the problem, the LDPC-based sampling method proposed in this thesis can also be a good candidate. It would be interesting to explore what happens if the proposed method is applied to sampling independent sets.

7.2 Alternative Ways to Generate Code Graph

In the proposed sampling scheme, a random LDPC code is introduced in order to distribute the randomness from fair coin flips into the source graph to generate samples. The LDPC code is generated in the way described in Section 5.1 mainly because it is both easy to implement and is known to work well with Belief Propagation algorithm. Yet this is clearly not the only way to construct the code graph. In this section, a few possible alternatives for code graph construction[8] are briefly talked about.

Irregular LDPC Codes

The LDPC codes considered so far are almost regular: both variable node degrees and check degrees are made as even as possible. Allowing irregular LDPC codes provides more flexibility in designing the code graph. When used as a channel code, carefully designed irregular LDPC codes are known to have better decoding performance than regular ones[11], and decoding will happen at check nodes with small degrees (where the information is most direct) first and information gradually propagates to other parts of the code graph. Although the setup in the proposed sampling scheme is slightly different (Belief Propagation runs on the combined graph instead of just the

[8]Due to the limitation of time, this thesis will not explore these possibilities in full depth. Yet it is worthwhile to mention them here.

code graph), a similar improvement may be expected when switching to irregular LDPC codes.

Moreover, as discussed in Section 6.2.2, with a regular LDPC code, the performance of the proposed sampling technique depends on the average check degree and it is necessary to match the average check degree to the source empirically in a rather ad-hoc way. With irregular LDPC codes, this phenomenon may be alleviated because check nodes with different degrees will send messages of different strengths to the source graph. It definitely would be interesting to see how the proposed sampling scheme performs using an irregular LDPC code as its code graph. However, we should comment that finding a good degree distribution is not an easy task.

Progressive Edge-Growth (PEG) Algorithms

Since the combined graph is a loopy one, Belief Propagation algorithm is expected to work better if the combined graph is more locally tree-like, or in other words, if it has a larger girth. This section presents an algorithm that provably generates a combined graph whose girth grows as $O(\log(Nt))$. The algorithm is inspired by [19], but modifications are made to account for the source graph.

Basically, the algorithm progressively add new edges between variable nodes and check nodes such that the placement of a new edge has as small an impact on the girth as possible. The main advantages of the PEG algorithm in [19] include its simplicity, in that it has low complexity and can be used for constructing codes of very large block length, and its flexibility, in that it can generate LDPC codes of any block length and any rate. Both advantages carry over to the following algorithm.

A few notations before stating the algorithm:

1. $\mathcal{V} \triangleq \{v_j^{(i)} : 1 \leq i \leq t, 1 \leq j \leq N\}$, $\mathcal{F} \triangleq \{f_j^{(i)} : 1 \leq i \leq t, 1 \leq j \leq M\}$ and $\mathcal{C} \triangleq \{c_i : 1 \leq i \leq K\}$ denote the set of variable nodes, factor nodes and check

nodes respectively[9].

2. The number of factors $v_j^{(i)}$ is connected to is called factor degree of $v_j^{(i)}$ and denoted as $fd_{v_j^{(i)}}$. Similarly the check degree of $v_j^{(i)}$ is denoted as $cd_{v_j^{(i)}}$. The degree of $v_j^{(i)}$ is the sum of factor degree and check degree: $d_{v_j^{(i)}} = fd_{v_j^{(i)}} + cd_{v_j^{(i)}}$. The variable degree of check node c_i is the number of variables connected to c_i and denoted as vd_{c_i}. Similarly the variable node of $f_j^{(i)}$ is denoted as $vd_{f_j^{(i)}}$.

3. The subgraph rooted at $v_j^{(i)}$ and expanded to the l^{th} level of variables (root as the 0^{th} level) is denoted as $\mathcal{N}^l_{v_j^{(i)}}$. Notice that factor nodes, as well as check nodes are included in the subgraph.

The factor degree and check degree of each variable node are given. The girth of source graph, g_s, equals to the girth of the factor graph describing the target distribution (because the source graph consists of t independent copies of the target distribution facor graph) and is also an input to the PEG-style algorithm. Clearly the girth of the combined graph is upper bounded by g_s. In the following discussion, g_s is assumed to be large enough so that at least one of the shortest cycles of the combined graph contains check nodes in it.

Algorithm 1 PEG-Style Algorithm for Generating the Combined Graph

```
for $i = 1 \to t$, $j = 1 \to N$ do
    for $r = 1 \to cd_{v_j^{(i)}}$ do
        • $\mathcal{N}^l_{v_j^{(i)}}$ for increasingly large $l$, until one of the following cases happens:
        1. $\exists l$, s.t. $\mathcal{N}^l_{v_j^{(i)}} = \mathcal{N}^{l+1}_{v_j^{(i)}}$, and $\exists c \in C$ s.t. $c \notin \mathcal{N}^l_{v_j^{(i)}}$.
        2. $\exists l$, s.t. all check nodes in $C$ appear in $\mathcal{N}^{l+1}_{v_j^{(i)}}$ but not all of them appear
in $\mathcal{N}^l_{v_j^{(i)}}$. Then choose $c \in C$ s.t. $c \notin \mathcal{N}^l_{v_j^{(i)}}$.
        • Add an edge between $v_j^{(i)}$ and $c$. [10]
    end for
end for
```

[9]For variable nodes and factor nodes, the subscript indicates the variable/factor index in the target distribution, while the superscript indicates the index of the copy (of the factor graph of target distribution).

The following theorem guarantees that if the combined graph is generated in the way described in Algorithm 1, its girth will grow as $O(Nt)$.

Theorem 3. *The girth of the combined graph generated from the above algorithm g satisfies:*

$$g \geq 2(\log_a(\frac{K(a-1)}{d_{\max}}+1)+1)$$

where $a = (vd_{\max}-1)(d_{\max}-1)$, $d_{\max} = \max_{i,j} d_{v_j^{(i)}}$, $vd_{\max} = \max\{\max_i vd_{c_i}, \max_{i,j} vd_{f_j^{(i)}}\}$

Proof of Theorem 3. Suppose during the construction, the first time a length g cycle appears after connecting variable node v to check node c. Notice only case 2 will introduce a new cycle, thus by the way the algorithm is designed, we know that before adding this new edge, $c \notin \mathcal{N}_v^l$ but $c \in \mathcal{N}_v^{l+1}$.
$\therefore g = 2(l+2)$
Since we are in case 2, $\forall c \in C$, $c \in \mathcal{N}_v^{l+1}$

$$\begin{aligned}
\therefore K &\leq \sum_{i=1}^{l+1} \text{number of check nodes at level i} \\
&\leq d_{\max} + d_{\max}(vd_{\max}-1)(d_{\max}-1) + \cdots + d_{\max}(vd_{\max}-1)^l(d_{\max}-1)^l \\
&= d_{\max}\frac{(vd_{\max}-1)^{l+1}(d_{\max}-1)^{l+1}-1}{(vd_{\max}-1)(d_{\max}-1)-1} \\
&= d_{\max}\frac{a^{l+1}-1}{a-1} \\
\Rightarrow & \frac{K(a-1)}{d_{\max}}+1 \leq a^{l+1} \\
\Rightarrow & l \geq \log_a(\frac{K(a-1)}{d_{\max}}+1)-1 \\
\Rightarrow & g \geq 2(\log_a(\frac{K(a-1)}{d_{\max}}+1)+1)
\end{aligned}$$

Since a and $d_{\max}$ are constants, girth g grows as log(K), where K (the number of check nodes) is in turn proportional to Nt (total number of variable nodes) and Nm (total number of factor nodes). □

Ramanujan Graphs

Ramanujan graphs are regular graphs that have the largest possible spectral gap.

These graphs are interesting in the coding context because they can be constructed to have both large girth and good expander properties[11]. There have been attempts to construct codes based on Ramanujan graphs, such as [12]. However, the code constructed in [12] is of a fixed rate 1/2. The construction is specific to that rate and seems difficult to generalize.

Construction of codes would be much more straight-forward if the underlying graph was a bipartite Ramanujan graph. In fact, [28] has proved that bipartite Ramanujan graph of every degree exists. Unfortunately, the proof is existential rather than constructional. If someday a computationally feasible construction can be found for bipartite Ramanujan graph, it might be useful to produce good codes with large girth.

[11][10] proves that sparse graphs with good expander properties result in good LDPC codes.

Chapter 8

Conclusions and Future Work

This thesis has proposed a new sampling technique that generates samples from an arbitrary target distribution over a finite alphabet. Such a distribution can always be converted into one over a sequence of binary variables, thus without loss of generality, we consider target distribution $p_{\mathbf{x}}(\cdot)$ over $\mathbf{x} \in \{0,1\}^N$. The distribution is described by a factor graph.

The main intuition of the proposed sampling scheme comes from the concept of typical sequences. If the concatenation of k independent copies of $\mathbf{x}$: $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(k)})$ is in the $\delta-$strong typical set defined with respect to $p_{\mathbf{x}}(\cdot)$, then by definition, the empirical distribution computed from $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ..., $\mathbf{x}^{(k)}$ approximates $p_{\mathbf{x}}(\cdot)$ with arbitrary accuracy for large enough k. Thus $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ..., $\mathbf{x}^{(k)}$ can be seen as samples from the target distribution. On the other hand, it is well known that the typical set dominates the probability asymptotically, and all typical sequences are roughly equi-probable. Thus each typical sequence can be thought of as associated with an index. Sampling the target distribution can then be approximated asymptotically by uniformly picking an index and then finding the typical sequence that corresponds to the chosen index.

Conceptually, the proposed scheme does exactly that: it applies a randomly chosen LDPC code with appropriate rate to the variables, and uses the parity check values as the index. To uniformly randomly generate the index, K fair coins (K is the number of checks in the LDPC code) are flipped to determine whether each parity check value is 0 or 1. The space of all possible sequences is partitioned into 2^K bins, each corresponding to a particular value of the K parity checks.

To find the typical sequence associated with this index, we look for the sequence that both satisfies all the parity checks and has high probability under the target distribution. It is proved in Chapter 4 that a randomly generated LDPC code with appropriate rate serves as a good hash function for typical sequences, i.e. almost all bins contain one and only one typical sequence. Moreover, the typical sequence is also the most likely one in the bin with high probability. As a result, if the most probably codeword can be found (i.e. Maximum-likelihood is possible), the scheme can find a typical sequence with high probability and the obtained samples can approximate the target distribution arbitrarily closely. In practice, Maximum-likelihood simulation is computationally intractable and thus approximate simulation is performed via Belief Propagation algorithm. As can be seen from Section 6.1, the sampling scheme still provides good quality samples in this sub-optimal case.

Some desirable features of the proposed sampling scheme are highlighted as follows:

1. Unlike Rejection sampling, Importance sampling, or Metropolis-Hastings algorithm, the proposed scheme does not require a proposal distribution or proposal Markov chain. Recall that although the choice of proposal distribution/proposal Markov chain does not affect the asymptotic correctness, it does have vital impact on how fast good quality samples can be obtained. Choosing a proposal distribution/proposal Markov Chain that is easy to sample from and produce good samples efficiently is generally hard.

2. The proposed scheme can deal with high-dimensional $\mathbf{x}$. The computational cost of the scheme grows linearly with N (size of the source, measured in number of binary variables) and t (number of repetitions of the source) since the combined

graph is sparse and the number of iterations in Belief Propagation does not increase significantly with the size of the graph. In contrast, the acceptance rate usually decrease exponentially with dimension and thus Rejection sampling does not work well for high-dimensional $\mathbf{x}$. A similar argument can be made for Importance sampling as well.

3. In the proposed scheme, samples from different runs of the algorithm are guaranteed to be independent. Samples from the same run, although not independent (coupling introduced by the code graph), form a strong typical sequence of the target distribution. Notice with Markov Chain Monte Carlo methods, one cannot make similar independence claims due to the correlation between successive samples.

4. The proposed scheme makes relatively efficient use of randomness. As discussed in Section 7.1, when normalized by total number of bits, the amount of randomness needed in the proposed scheme is close to the entropy rate of the target distribution, which is a theoretical lower bound on randomness requirement. In contrast, all other sampling techniques discussed in this thesis may demand exponentially many random bits.

However, as already discussed, one should not expect the existence of a perfect sampling method due to the inherent hardness of general inference problem and Monte Carlo methods' ability to solve inference problems with arbitrary precision given good quality samples. The major weaknesses of the proposed sampling scheme are listed below:

1. While Rejection sampling, Importance sampling and MCMC methods work for both continuous and discrete variables, the proposed scheme only works for discrete variables with finite size alphabets.

2. Although the running time can be proved to be polynomial, for moderate N (e.g. $N = 100$) and the sources considered in this thesis, the proposed scheme takes considerably longer than Markov Chain Monte Carlo methods to produce

the same number of samples. (But this is not a fair comparison since the consecutive samples from Markov Chain Monte Carlo methods can be highly correlated.)

3. For Rejection sampling, although it can take very long to produce a sample, when a sample is produced, it is always a correct one. For the proposed scheme, since approximate simulation is used (via Belief Propagation algorithm), there is no theoretical guarantee on the correctness of the obtained samples. The quality of the samples can only be examined empirically. Importance sampling and Markov Chain Monte Carlo methods have the same problem.

In short, as with any other existing sampling technique, the proposed LDPC-based sampling scheme has its merits and problems. But it definitely provides a different trade-off between quality, computational cost and randomness utilization.

Finally, a few possible directions for future work are listed as follows:

1. Section 7.2 proposes a few alternative ways to generate the code graph. These ideas certainly deserve a more thorough exploration. In particular, using irregular LDPC codes will provide more flexibility and may alleviate the sampling performance's dependence on check degrees. Thus understanding how to find a good degree distribution for a given target distribution $p_{\mathbf{x}}(\cdot)$ is a topic worth spending some time on.

2. The discussion on Belief Propagation algorithm and approximate simulation in this thesis is mainly qualitative and descriptive. It would be nice to have a more quantitative analysis on the subject and be able to make sharper statements.

3. The target distributions used in the simulations so far (Markov Chain model, Single Loop model, Ising model) are relatively artificial. It would be interesting to test the proposed scheme in applications of real importance, for example, the Weighted Maximum Independent Set Problem mentioned in Section 7.1.

Bibliography

[1] L. Onsager *Crystal Statistics. I. A Two-Dimensional Model With An Order-Disorder Transition* Phys. Rev. 65, 117149 1944

[2] B. Kaufman Crystal Statistics. II. Partition Function Evaluated By Spinor Analysis Phys. Rev. 76, 12321243 1949

[3] G.F.Newell, E.W.Montroll *On the Theory of the Ising Model of Ferromagnetism* 1953

[4] R.G.Gallager *Low Density Parity Check Codes* 1963

[5] E.R.Berlekamp, R.J.McElieve, H.C.A.van Tilborg *On the Intractability of Certain Coding Problems* 1978

[6] J.Pearl *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* 1988

[7] R.Arratia, L.Gordon *Tutorial on Large Deviations for the Binomial Distribution* 1989

[8] D.J.C.Mackay *Introduction to Monte Carlo Methods* 1996.

[9] D.J.C.Mackay, R.M.Neal *Near Shannon Limit Performance of Low Density Parity Check Codes* 1996

[10] M.Sipser, D.A.Spielman *Expander Codes* 1996

[11] M.G.Luby, M.Mitzenmacher, M.A.Shokrollahi, D.A.Spielman *Analysis of Low Density Codes and Improved Designs Using Irregular Graphs* 1998

[12] J.Rosenthal, P.O.Vontobel *Construction of Codes Using Ramanujan Graphs and Ideas from Margulis* 2000

[13] J.S.Yedidia, W.T.Freeman, Y.Weiss *Understanding Belief Propagation and Its Generalizations* 2001

[14] Transition P.G.Bolhuis, D.Chandler, C.Dellago, P.L.Geissler *Path Sampling: Throwing Ropes Over Rough Mountain Passes, in the Dark* 2002

[15] Y.Matsunaga, H. Yamamoto *A Coding Theorem for Lossy Data Compression by LDPC Codes* 2003

[16] M.I.Jordan *An Introduction to Probabilistic Graphical Models* Book Draft 2003

[17] G.Caire, S.Shamai, S.Verdu *Noiseless Data Compression with Low-Density Parity-Check Codes* 2004

[18] L.Trevisan *Inapproximability of Combinatorial Optimization Problems* 2004

[19] X.Y.Hu, E.Eleftheriou, D.M.Arnold *Regular and Irregular Progressive Edge-Growth Tanner Graphs* 2005

[20] D.M.Bishop *Pattern Recognition and Machine Learning* Springer. pp. 359418 2006

[21] S.Miyake, J.Muramatsu *Construction of A Lossy Code Using LDPC Matrices* 2007.

[22] D.J.Costello, Jr., G.D.Forney, Jr. *Channel Coding: The Road to Channel Capacity* 2007

[23] M.Mézard, A.Montanari *Information, Physics and Computation* 2008

[24] S.Sanghavi, D.Shah, A.Willsky *Message-passing for Maximum Weight Independent Set* 2008

[25] E.Mossel, D.Weitz, N.Wormald *On the Hardness of Sampling Independent Sets Beyond the Tree Threshold* 2008

[26] R.J.Allen, C.Valeriani, P.R.ten Wolde *Forward Flux Sampling for Rare Event Simulations* 2009

[27] D.Koller, N.Friedman, *Probabilistic Graphical Models: Principles and Techniques* MIT Press 2009

[28] A.Marcus, D.A.Spielman, N.Srivastava *Interlacing Families I: Bipartite Ramanujan Graphs of All Degrees* 2013

[29] Ying-zong Huang *A Class of Compression Systems with Model-Free Encoding* 2014.

[30] Ying-zong Huang *Model-Code Separation Architectures for Compression Based on Message-Passing* Doctoral Thesis 2015