Performance Analysis of Iterative Decoding Algorithms with Memory

by

Emil Janulewicz

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Applied Science

The Ottawa-Carleton Institute for
Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
April 2010

Copyright ©
2010 - Emil Janulewicz

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-68626-3
Our file Notre référence
ISBN: 978-0-494-68626-3

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

The undersigned recommend to

the Faculty of Graduate Studies and Research

acceptance of the Thesis

Performance Analysis of Iterative Decoding Algorithms with Memory

Submitted by Emil Janulewicz

in partial fulfilment of the requirements for the degree of

Master of Applied Science

Chair, Professor Howard Schwartz
Department of Systems and Computer Engineering

Thesis Supervisor, Professor Amir Banihashemi

Carleton University

April 2010

Abstract

Density evolution is often used to determine the performance of an ensemble of low-density parity-check (LDPC) codes under iterative message-passing algorithms. Conventional density evolution techniques over memoryless channels are based on the independence assumption amongst all the processed messages at variable and check nodes. This assumption is valid for many algorithms such as standard belief propagation (BP) and min-sum (MS) algorithms. However, there are other important iterative algorithms such as successive relaxation (SR) versions of BP and MS, and differential decoding with binary message passing (DD-BMP) algorithm of Mobini *et. al.*, for which this assumption is not valid. The dependence created among messages for these algorithms is due to the introduction of memory in the iterative algorithm. In this work, we propose a model for iterative decoding algorithms with memory which covers SR and DD-BMP algorithms as special cases. Based on this model, we derive a Bayesian network for iterative algorithms with memory over memoryless channels and use this representation to analyze the performance of the algorithms using density evolution. The density evolution technique is developed based on truncating the memory of the decoding process and approximating it with a finite order Markov process, and can be implemented efficiently. As an example, we apply our technique to analyze the performance of DD-BMP on regular LDPC code ensembles, and make a number of interesting observations with regard to the performance/complexity trade off of DD-BMP in comparison with BP and MS algorithms.

Acknowledgments

First I would like to give thanks to that which cannot be seen or touched. But it is always there and always will be. He is the alpha and the omega.

Throughout my academic experience I will always remember these two professors for inspiring my mind. Professor Amir Banihashemi, as my supervisor, you have been an amazing mentor for me in so many ways. Your dedication, hard word, and loyalty will always be remembered. Thank You. Secondly, Professor Yongyi Mao, you thrilled me during your lectures in Graphical Models and Information Theory with your intelligence and great sense of humour.

Anoosheh Heidarzadeh. I cannot begin to count all of the interesting and funny discussions we have had in topics such as math, engineering, philosophy, God, friendship, relationships, etc. There is no mind like yours and I am lucky to have met you during my time at Carleton. Thank You for always being there for me in bad times and in good times. Maybe life would be easier if we just considered the asymptotic case. But unfortunately it is finite for us, so let us always embrace the challenges.

Last but not least, Thank You to my wonderful loving parents, Janusz and Wanda. I do not even know where to begin so I will start at the end. I Love You. Thank You. And to the rest of my family; Felicia, Daniel, Helena, and Sunil. You have been my number one fans. I trust and love you all. Thank You for always being there throughout my journey.

Table of Contents

List of Tables

List of Figures

List of Symbols and Acronyms

List of Acronyms

Acronym	Explanation
i.i.d.	Independent identically distributed
p.d.f.	Probability density function
p.m.f.	Probability mass function
AWGN	Additive white Gaussian noise
BCJR	Bahl, Cocke, Jelinek and Raviv
BER	Bit Error Rate
BIAWGN	Binary input additive white Gaussian noise
BN	Bayesian network
BP	Belief propagation
DAG	Directed acyclic graph
dB	Decibel

DD-BMP	Differential decoding with binary message passing
DE	Density evolution
HMM	Hidden Markov model
LDPC	Low-density parity-check
LLR	Log-likelihood ratio
LR	Likelihood ratio
MAP	Maximum a posteriori
MS	Min-sum
SNR	Signal-to-noise ratio
SR	Successive relaxation

List of Symbols

Symbol	Explanation
$\perp\!\!\!\perp$	Independent (In the context of random variables)
β	Relaxation factor
Δ	Quantization interval length
$\delta[x]$	Dirac Delta function
$\lambda(x)$	Variable node degree profile
$\rho(x)$	Check node degree profile
σ^2	Noise variance
$\mathbf{a}$	A vector
$\mathbf{A}^T$	Transpose of matrix $\mathbf{A}$
$\mathbf{A}_{ij}$	Element at row i and column j of matrix $\mathbf{A}$
$B^{(\ell)}$	Memory content at iteration ℓ
C	Linear block code
$ch(V)$	Set of nodes which are the children of node V
c_j	j^{th} check node in a Tanner graph
c_{th}	Clipping threshold applied to the output of the channel
d	Depth of decoding tree

d_c	Check node degree
d_v	Variable node degree
$\mathcal{E}$	Set of directed edges in a graph G
$\mathbb{E}[X]$	Expected value of random variable X
E_b	Energy per information bit
G	Directed acyclic graph
H	Parity-check matrix
ℓ	iteration number
$m_{a\to b}^{(\ell)}$	Message passed from node a to node b at iteration ℓ
M	Number of check nodes
M_0	Message received from the channel
n	Memory truncation order
N	Number of variable nodes (block length)
$N_0/2$	Power spectral density of noise
$N(a)$	Neighbouring nodes of node a
$pa(V)$	Set of nodes which are the parents of node V
$\mathbb{P}(X)$	Probability distribution function of random variable X

$P(A)$	Probability of event A
$P(A\|B)$	Probability of event A given event B
q	Number of quantization bits
$Q(x)$	Q function
R	Rate of code
$\|S\|$	Cardinality of set S
$\mathcal{V}$	Set of nodes in a graph G
v_i	i^{th} variable node in a Tanner graph

Chapter 1

Introduction

1.1 LDPC Codes

Low-Density Parity-Check (LDPC) codes [1] were first introduced in 1962 by Robert G. Gallager. They are a class of linear block codes C in which the parity-check matrix H is sparse; meaning H contains a low density of 1 values. An LDPC code C can be represented by its corresponding Tanner Graph [2] which is a bipartite graph showing the connections between variable nodes and check nodes. For example, consider a parity check matrix H of size $M \times N$ where M and N represent the number of check nodes and variables nodes in the corresponding Tanner graph, respectively. The construction of the Tanner graph consists of adding an edge (link) between each variable node v_i, $i = 1, .., N$ and check node c_j, $j = 1, .., M$ such that $H_{ji} = 1$.[1]

An ensemble of LDPC codes can be fully described by an arbitrarily large block length N (analogous to the number of variable nodes), and the degrees of each variable and check node. The degree of a variable node v_i is the number of check equations that it participates in (or the number of 1's in column i of its corresponding

[1] H_{ji} refers to the element in the j^{th} row and i^{th} column of matrix H.

parity-check matrix). Similarly, the degree of a check node c_j is the number of variable nodes which take part in that particular check (or the number of 1's in the row j of its corresponding parity-check matrix).

When characterizing the degrees of variable and check nodes, LDPC codes fall under two categories; *regular* and *irregular*. A (d_v, d_c) regular LDPC code has variable node degree d_v for each variable node, and check node degree d_c for each check node. An irregular LDPC code can be described by two polynomials called degree profiles or distributions. This is shown by:

$$\lambda(x) = \sum_{i=1}^{d_v^{\max}} \lambda_i x^{i-1}, \;\; \rho(x) = \sum_{j=2}^{d_c^{\max}} \rho_j x^{j-1}, \tag{1}$$

where $0 \leq \lambda_i, \rho_j \leq 1$, $1 \leq i \leq d_v^{\max}$, $2 \leq j \leq d_c^{\max}$, $\sum_{i=1}^{d_v^{\max}} \lambda_i = 1$, and $\sum_{j=2}^{d_c^{\max}} \rho_j = 1$. λ_i represents the fraction of variable nodes that have degree i. Similarily, ρ_j represents the fraction of check nodes which have check degree j.

Remark: A (d_v, d_c) regular LDPC code can be thought of as a special case of an irregular code where $\lambda(x) = x^{d_v - 1}$ and $\rho(x) = x^{d_c - 1}$.

1.2 Decoding LDPC Codes

Consider we send a codeword $\mathbf{w} = \{w_1, w_2, ..., w_N\} \in C$ over some channel where each w_i can take on a finite number of values from a set denoted by S_C. At the output of the channel (input of the decoder), we receive the vector $\mathbf{y} = \{y_1, y_2, ..., y_N\}$ which is a noisy version of $\mathbf{w}$. The role of the decoder is to correctly deduce that $\mathbf{w}$ was sent given $\mathbf{y}$. Therefore, the decoder would choose a codeword $\mathbf{w}' \in C$ which is most probable to have been sent given the channel output $\mathbf{y}$:

$$\mathbf{w}' = \arg\max_{\mathbf{w} \in C} P(\text{codeword sent is } \mathbf{w} \mid \mathbf{y}). \tag{2}$$

This type of decoder is called a MAP (maximum a posteriori) decoder for a codeword. When dealing with LDPC codes, we may also be interested in the bit (symbol) error rate. In this case we wish to find the most probable symbol for each w_i, $1 \leq i \leq N$ given $\mathbf{y}$:

$$w_i = \arg\max_{w_s \in S_C} P\big(i^{\text{th}} \text{ symbol sent is } w_s \mid \mathbf{y}\big). \tag{3}$$

LDPC codes as a linear code can be represented by a trellis [3,4], and for codes with trellis representations, the BCJR algorithm [3] is capable of computing the a posteriori probabilities in (3). However, the complexity of the trellis increases exponentially with the dimension[2] of the code. The probabilities in (3) can also be computed exactly using an iterative algorithm if the code is cycle free [5]. However, even when the code is not cycle free, and iterative decoding algorithms become suboptimal, good performance can still be reached whilst significantly reducing the decoding complexity [6]. More formally, the decoding algorithms used for LDPC codes are known as iterative message passing algorithms, most notably the belief propagation (BP) [7] and min-sum (MS) [8] algorithms and their variants [9–12]. Iterative message passing algorithms can be described by their variable and check node operations which may be time (iteration) variant or invariant.

1.3 Iterative decoding algorithms with memory

There is also a novel class of iterative message passing algorithms which contain memory. Examples of these in recent literature are successive relaxation (SR) variants of BP and MS [13–15] and differential decoding with binary message-passing (DD-BMP) [16, 17].

[2]For a code C of block length N with parity check matrix H, the dimension k is represented as $k = N - \text{rank}(H)$.

DD-BMP is considered to be a hard-decision algorithm since the messages passed are binary. Therefore, the complexity is significantly reduced when compared to BP and MS which are soft-decision algorithms. However, due to the soft information stored in memory nodes, the DD-BMP algorithm still has good performance, performing as close as 1 dB and 0.5 dB to BP for particular random and finite-geometry LDPC codes. Due to its good performance and low complexity, this algorithm is specifically attractive for high-speed low-power applications.

SR algorithms were inspired by the dynamics of analog decoders and introduced in [13]. These algorithms were shown to have very good performance, even outperforming BP by up to about 0.5 dB. In general, as evident by these two algorithms (SR and DD-BMP), the presence of memory improves the performance; i.e. in comparison to BP and MS algorithms, the decoding complexity is significantly reduced whilst experiencing a low (if any) penalty in transmitted power.

1.4 Density Evolution

Density evolution is an analytical tool which can be used to find the *threshold* of a particular code ensemble under a given iterative decoding algorithm [18]. The *threshold* is an asymptotic measure of performance and is defined as the worst channel parameter (e.g., largest noise variance) for which the probability of error still converges to zero as the number of iterations in the decoding algorithm tends to infinity. Conversely, with fixed channel parameters, density evolution can find the highest code rate[3] r such that the probability of error still converges to zero as the number of iterations tends to infinity. Density evolution is also a powerful

[3]Assuming the rows of the parity-check matrix are linearly independent, $r = 1 - \frac{d_v}{d_c}$. More generally (irregular codes), $r = 1 - \frac{\sum_{i=1}^{d_v^{\max}} \lambda_i i}{\sum_{j=2}^{d_c^{\max}} \rho_j j}$.

technique for constructing irregular LDPC codes through the optimization of the degree distributions [19, 20]. In general, there is a large body of literature devoted to the derivation and applications of density evolution for a variety of iterative decoding algorithms, most notably for belief propagation (BP) and min-sum (MS) algorithms and their variants [21–23]. To the best of our knowledge however, all the message-passing algorithms analyzed by density evolution in the literature are *memoryless*, i.e., the output messages of a variable node (check node) at iteration ℓ are only a function of the input messages to that node at iteration ℓ ($\ell - 1$) and also the initial message of the channel in the case of variable nodes. There exist however a number of iterative decoding algorithms, such as successive relaxation (SR) variants of BP and MS [13,14], and DD-BMP (differential decoding with binary message-passing) [16, 17], that have memory. In fact, the presence of memory in these algorithms improves the performance but makes the density evolution analysis much more complex. So far, the analysis of such algorithms has been limited to Monte Carlo simulations.

At the heart of *density evolution* lies the following theorems which are introduced in [18].

Theorem 1. *[Concentration]: Let $P_e^N(\ell)$ be the expected fraction of incorrect messages which are passed in the ℓ^{th} iteration, where the expectation is over all instances of the code, the choice of the sent codeword, and the realization of the noise. For any $\delta > 0$, the probability that the actual fraction of incorrect messages which are passed in the ℓ^{th} iteration for any particular such instance lies outside the range $\left(P_e^N(\ell) - \delta, P_e^N(\ell) + \delta\right)$ converges to zero exponentially fast in N.*

Theorem 2. *[Convergence to Cycle-Free Case]: $P_e^N(\ell)$ converges to $P_e^\infty(\ell)$ as N tends to infinity, where $P_e^\infty(\ell)$ is the expected fraction of incorrect messages passed in*

the ℓ^{th} decoding round assuming that the graph does not contain cycles of length 2ℓ or less.

Using Theorem 2, in performing density evolution and in the calculation of thresholds, one can assume a cycle-free (tree-like) graphical representation of the ensemble [18], and track the probability distribution of the messages under this assumption. Concentration results (Theorem 1) then guarantee that, as the block length N of the code tends to infinity, almost all instances of the ensemble perform nearly the same and their performance tends to that of the cycle-free case.

In the density evolution analysis of memoryless algorithms over memoryless channels, the cycle-free assumption leads to the independence between all messages being processed at variable and check nodes. This significantly simplifies the analysis, and makes the complexity of density evolution in each iteration independent of the iteration number ℓ, and thus the total complexity of density evolution scales only linearly with ℓ. However, these independence assumptions do not hold for iterative algorithms with memory. The introduction of memory into the decoding algorithm will introduce dependency amongst messages processed at variable nodes. This significantly increases the difficulty of the analysis as well as the computational complexity.

There have also been studies on LDPC codes over channels with memory [24–27]. In particular, [24] performs density evolution of LDPC codes over the Gilbert-Elliot channel which is in essence a Hidden Markov Model (HMM). This would lead to dependence between symbols in the codeword through the introduction of dependence in the noise samples. This can be more clearly seen in Fig. 1. The variables S_i, $1 \leq i \leq N$ represent the state of the channel. The sequence $S_1, S_2, ..., S_N$ forms the underlying Markov chain which is *hidden*. The outputs Y_i of each state S_i is the

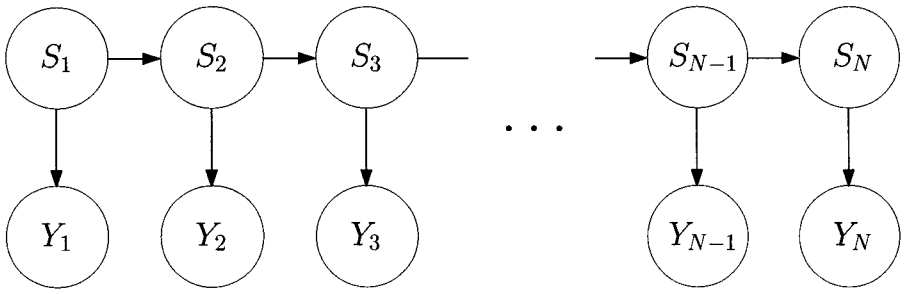


Figure 1: HMM representing the Gilbert-Elliot channel

corresponding noise sample which can be observed. It is well known that for a first-order, time-homogeneous, two-state[4] Markov chain, the mutual information between two states, i.e. S_i and S_j, goes to zero as the distance between states gets larger since the Markov chain always reaches equilibrium[5]. This implies that the mutual information[6] between the corresponding outputs (noise samples) goes to zero since $I(Y_i\,;Y_j) \leq I(S_i\,;S_j)$, $1 \leq i,j \leq N$, through the *Data Processing Inequality* [28]. Since the distance between any two received symbols can be arbitrarily large in a particular decoding neighbourhood in the asymptotic case ($N \to \infty$), the mutual information between their respective states goes to zero. Hence the independence assumption still holds.

1.5 Thesis Content and Organization

In this thesis, we develop the framework for the density evolution analysis of iterative message passing algorithms with memory over memoryless channels. We first present a general model for iterative decoding algorithms with memory which includes

[4]In the Gillbert-Elliot channel, each S_i has two states; either 'Bad' or 'Good'.

[5]Equilibrium implies that the probability of a particular state approaches a constant independent of time and of the initial state probabilities.

[6]The mutual information between two random variables, say X and Y, is defined as $I(X\,;Y) = \sum_{x,y} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$ where $p(x,y)$, $p(x)$, and $p(y)$ represent the joint and marginal probability mass functions of X and Y respectively.

DD-BMP and SR algorithms as special cases. Based on this model, we employ the directed acyclic graph (DAG) representation, also known as a *Bayesian network*, to capture the dependencies among different messages and memory contents in a space with two dimensions: iteration number ℓ and the depth of the decoding tree d. Using this, we show that for algorithms with memory, the incoming messages to a node are no longer independent, and that the existing dependencies cause the complexity of density evolution to grow at least exponentially with ℓ. We also derive the density evolution equations and use techniques to make them tractable. In particular, we introduce an approximation by truncating the memory of the decoding process and estimating it with a Markov process of finite order n (not an increasing function of ℓ). As an example, we apply the proposed density evolution technique to DD-BMP for a number of regular LDPC code ensembles and obtain the thresholds. The threshold results converge to the true threshold values with arbitrary precision as the memory truncation length n is increased. Our study shows that for DD-BMP, $n = 4$ is often enough to obtain the threshold with a precision of 0.01 dB. A careful inspection of the threshold values of DD-BMP for regular LDPC codes in comparison with BP and MS thresholds reveals a number of interesting phenomena: (a) By increasing the rate, the performance gap between BP and DD-BMP reduces so that for high rate codes the difference is only a fraction of a dB; (b) For a given rate, by increasing the degrees, the performance advantage of MS over DD-BMP decreases at first and then eventually reverses so that for large degrees, e.g., (7,14) for rate-1/2 codes, DD-BMP outperforms MS by a few tenths of a dB. These performance results for DD-BMP are impressive since both the message-passing and the check node operations in DD-BMP are significantly less complex than those in BP and MS algorithms. In addition to the above results, one of the main contributions of this work is to introduce the general framework for presentation and analysis of iterative decoding algorithms

with memory. This can be used to design irregular codes for the existing algorithms or to devise new algorithms with improved performance/complexity trade offs.

This thesis is organized as follows: In Chapter 2, we present the general model for iterative algorithms with memory and show that both DD-BMP and SR algorithms are special cases. Chapter 3 is devoted to the Bayesian network representation of iterative algorithms with memory and discussions on (conditional) dependencies among different random variables. In Chapter 4, the equations for density evolution are derived and the complexity of the analysis is discussed. In Chapter 5, memory truncation is explained and validated. Chapter 6 is used to report the numerical results of DD-BMP applied to regular LDPC codes over the binary input additive white Gaussian noise (BIAWGN) channel and their analysis.

1.6 List of Contributions

- A general model for iterative decoding algorithms with memory which includes DD-BMP and SR algorithms as special cases is developed
- The decoding process of iterative deocding algorithms with memory is modeled and analyzed through a *Bayesian network* which captures the dependencies among different messages and memory contents in a space with two dimensions: iteration number ℓ and depth of the decoding tree d
- A recursive algorithmic framework based on the *Bayesian network* is developed to compute the distribution of messages at each iteration ℓ
- Efficient algorithms are developed to compute probability distributions required for the *density evolution* analysis with low complexity

- A method for approximating the calculations performed in *density evolution* through *memory truncation* is introduced in order to make the computations tractable

- Approximations are validated by showing a convergence property towards the true *threshold* value under DD-BMP decoding

- Threshold values under DD-BMP decoding for many code ensembles are computed and interesting comparisons are made to well known iterative decoding algorithms such as BP and MS

Chapter 2

General Model for Iterative Decoding Algorithms with Memory

2.1 Model

Fig. 2(a) shows a snapshot of the Tanner graph of an LDPC code at iteration ℓ for a memoryless decoding algorithm, where variable and check nodes are represented by circles and squares, respectively. Under the cycle-free assumption over memoryless channels and based on the principle of *extrinsic* message passing[1], all incoming messages $M_1^{(\ell)}, \ldots, M_{d_v-1}^{(\ell)}$ to node V are independent of each other and of the channel message M_0 (d_v is the variable node degree). Moreover, all the incoming messages are identically distributed. The outgoing message $M_{V\to C}^{(\ell)}$ is thus a function of $d_v - 1$ independent identically distributed (i.i.d.) random variables and the channel message. Similarly, the outgoing message of a check node is a function of $d_c - 1$ i.i.d. random variables corresponding to the extrinsic incoming messages. The distribution of a function of independent random variables is relatively easy to find since the joint distribution of these random variables can be factored as the product of their respective marginal distributions. In this case, the complexity of finding the message

[1]An outgoing message from a node V on an edge e does not depend on any incoming messages to node V on edge e.

distributions in each iteration, does not change with increasing ℓ.

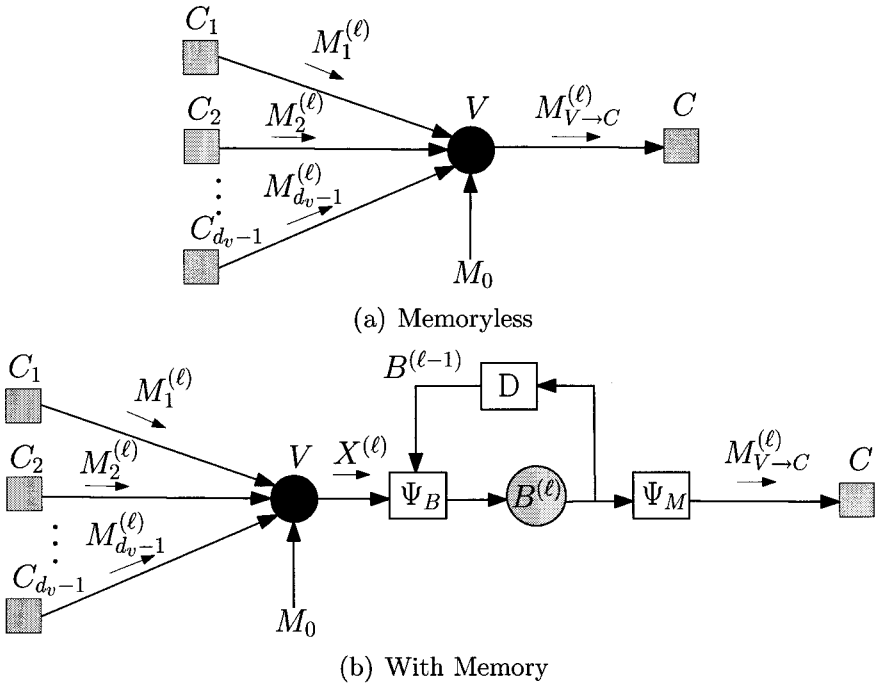


Figure 2: Message passing between variable and check nodes in iterative algorithms a)without and b)with memory.

In this work, we consider the existence of a memory node on the links from variable nodes to check nodes. This can be generalized to the case where memory exists in both directions. Our model is shown in Fig. 2(b) on a snapshot of the Tanner graph at iteration ℓ. Similar to Fig. 2(a), we have a set of i.i.d. extrinsic incoming messages to a variable node V. The outgoing message from V is denoted by $X^{(\ell)}$. The straight pass through line between V and C in Fig. 2(a) is now replaced by a memory unit whose content $B^{(\ell)}$ is updated by $\Psi_B(X^{(\ell)}, B^{(\ell-1)})$, where Ψ_B is a deterministic function of $X^{(\ell)}$ and the content of the memory at iteration $\ell - 1$. In a more general case, $B^{(\ell)}$ can be a function of all previous memory contents $B^{(0)}, \ldots, B^{(\ell-1)}$. For the sake of simplicity however we limit ourselves in this work to the case described in Fig. 2(b). The incoming message $M_{V\to C}^{(\ell)}$ to node C from V is obtained by $\Psi_M(B^{(\ell)})$ as a deterministic function of the memory content $B^{(\ell)}$. It is important to note that

while the message $X^{(\ell)}$ is a function of independent random variables, the outgoing message, $M_{V\to C}^{(\ell)}$, is a function of dependent random variables $X^{(\ell)}$ and $B^{(\ell-1)}$. For the check nodes however, since no memory exists at the output links to the variable nodes, we just need to find the distribution of a function of $d_c - 1$ i.i.d. extrinsic incoming messages. This is similar to the case for memoryless algorithms. In the following therefore, our focus will be on the link from variable nodes to check nodes and on finding the distribution of $B^{(\ell)}$ and $M_{V\to C}^{(\ell)}$.

2.2 Proposed Algorithm for Iterative Decoding with Memory

For this algorithm we consider the application of a binary LDPC code C over a binary input symmetric output memoryless channel. Coded bits 0 and 1 are mapped to channel input symbols $+1$ and -1, respectively. An LDPC code C of block length N can be described by a full row rank[2] $M \times N$ parity check matrix H where M and N represent the number of check nodes and variable nodes in the corresponding Tanner graph, respectively. This algorithm can be applied to regular and irregular degree distributions however in this work we only consider density evolution over regular codes.

Between each v_i and c_j we assign a memory node denoted as $B_{v_i,c_j}^{(\ell)}$ which is the memory content of the edge adjacent to v_i and c_j at iteration ℓ. Theoretically, a memory node may take on values from the set of real numbers, however, we choose a discrete set for implementation. Using q quantization bits, each memory element takes on a value from the set $S_B := \{-(2^{q-1}-1), ..., 2^{q-1}-1\}$. The message sent from node a to node b at iteration ℓ is denoted by $m_{a\to b}^{(\ell)}$. Messages passed along the edges of the

[2]A parity-check matrix having full row rank ensures the non-existence of redundant check nodes.

graph take on a value from the set $S_M := \{-D, -(D-1), ..., 0, ..., D-1, D\}$ where $D = (|S_M| - 1)/2$ for $|S_M|$ odd, and $S_M := \{-D, -(D-1), ..., -1, 1, ..., D-1, D\}$ where $D = |S_M|/2$ for $|S_M|$ even. Note that for any finite set S, $|S|$ represents the cardinality (number of elements) of S. The initial received values y_i at the decoder input at each variable node take on a possible value from the set S_O which is the set of possible outputs of the channel. These values are clipped symmetrically at a threshold c_{th}, and then uniformly quantized in the range $[-c_{th}, c_{th}]$. The quantized value is denoted as $K_B(y_i)$. In general, $K_B : S_O \rightarrow S_B$ such that $K_B(\pm y) = \pm b, y \in S_O, b \in S_B$. K_B is specifically chosen to be an odd function in order to preserve the symmetry of the channel output. The importance of channel (and decoder) symmetry is fully discussed in section 2.4. We now give a detailed description of the iterative decoding algorithm with memory as well as the two special cases: DD-BMP and SR algorithms.

2.2.1 Algorithm Description

Initialization, $\ell = 0$

$z_i^{(0)} = (1 - \mathrm{sgn_r}(y_i))/2, i = 1, ..., N$, where $\mathrm{sgn_r}(x) = 1$, for $x > 0$, and -1 for $x < 0$. (For $x = 0$, $\mathrm{sgn_r}(x)$ takes on $+1$ or -1 randomly with equal probability). Let $\mathbf{z^{(0)}} = \left(z_1^{(0)}, ..., z_N^{(0)}\right)$. If $\mathbf{z^{(0)}} H^T = \mathbf{0}$ then the decoding is stopped and the transmitted codeword is estimated by $\mathbf{z^{(0)}}$. Otherwise, the memory contents are initialized by $B_{v_i,c_j}^{(0)} = K_B(y_i), i = 1, ..., N, j = 1, ..., M$, such that $H_{ji} = 1$. The initial messages sent from variable nodes to check nodes are $m_{v_i \rightarrow c_j}^{(0)} = \Psi_M\left(B_{v_i,c_j}^{(0)}\right)$ where $\Psi_M : S_B \rightarrow S_M$ and Ψ_M is an odd function.

Check Node Operation, $\ell \geq 1$

Consider a particular check node to variable node message $m^{(\ell)}_{c_j \to v_i}$. This message is a function of $d_c - 1$ incoming messages as follows:

$$m^{(\ell)}_{c_j \to v_i} = \Psi_C \left(m^{(\ell-1)}_{v_k \to c_j} : v_k \in N(c_j) \backslash v_i \right) \tag{4}$$

where $\Psi_C : S_M^{d_c - 1} \to S_M$, $N(b)$ denotes the neighboring nodes of node b, and $N(b)\backslash a$ denotes $N(b)$ excluding a.

Variable Node Operation, $\ell \geq 1$

We define the output of a variable node as $X^{(\ell)}_{v_i,c_j}$ which is used in updating the memory element $B^{(\ell)}_{v_i,c_j}$. This is not to be confused with the variable node to check node message since with the addition of memory, $m^{(\ell)}_{v_i \to c_j}$ is a deterministic function of $B^{(\ell)}_{v_i,c_j}$ and not of the variable node output.

$$X^{(\ell)}_{v_i,c_j} = \Psi_V \left(m^{(\ell)}_{c_k \to v_i} : c_k \in N(v_i) \backslash c_j, y_i \right) \tag{5}$$

where $\Psi_V : S_M^{d_v - 1} \times S_O \to S_X$. The alphabet space S_X of the output of variable nodes will vary depending on Ψ_V, S_M, S_O, and d_v.

Memory Element Update, $\ell \geq 1$

$$B^{(\ell)}_{v_i,c_j} = \Psi_B \left(B^{(\ell-1)}_{v_i,c_j}, X^{(\ell)}_{v_i,c_j} \right), \tag{6}$$

where $\Psi_B : S_B \times S_X \to S_B$. As mentioned before, $m^{(\ell)}_{v_i \to c_j}$ is a deterministic function of $B^{(\ell)}_{v_i,c_j}$:

$$m^{(\ell)}_{v_i \to c_j} = \Psi_M \left(B^{(\ell)}_{v_i,c_j} \right), \tag{7}$$

where $\Psi_M : S_B \to S_M$.

Hard-Decision, $\ell \geq 1$

$$D_i^{(\ell)} = \sum_{c_j \in N(v_i)} \text{sgn}(B_{v_i,c_j}^{(\ell)}) + \text{sgn}(y_i), \tag{8}$$

$$z_i^{(\ell)} = \begin{cases} 1 & \text{if } D_i^{(\ell)} < 0 \\ 0 & \text{if } D_i^{(\ell)} > 0 \\ z_i^{(\ell-1)} & \text{if } D_i^{(\ell)} = 0 \end{cases}, \tag{9}$$

where $\text{sgn}(x) = 1$ for $x > 0$; $= -1$ for $x < 0$; and $= 0$ for $x = 0$. The algorithm is stopped if $\mathbf{z}^{(\ell)} H^T = \mathbf{0}$, or if the number of iterations has reached its maximum. In the former case, $\mathbf{z}^{(\ell)}$ is used as an estimate for the transmitted codeword, while in the later case a decoding failure is declared. Otherwise, $\ell = \ell + 1$, and the algorithm is continued from the check node operation. Note that we may substitute Equation (8) for $D_i^{(\ell)} = \sum_{c_j \in N(v_i)} B_{v_i,c_j}^{(\ell)} + K_B(y_i)$. It is possible that this new update operation may give better results, however, this has not been studied in this work.

2.3 Special cases of iterative decoding algorithms with memory

We now present two existing algorithms, DD-BMP and SR-MS-LLR[3], and give their specific check node, variable node, and memory update operations. We then show how these two algorithms can be presented in a more general context.

[3]SR-MS-LLR is chosen to be discussed since in [14], out of the four SR algorithms studied, SR-MS-LLR was shown to perform the best and has the lowest complexity. As well, the decoding algorithm is symmetric, making the density evolution analysis practical.

2.3.1 Differential Decoding with Binary Message Passing

Differential decoding with binary message passing (DD-BMP) was introduced in [16, 17], as an attractive alternative to purely hard-decision algorithms. This algorithm combines the simplicity of binary message-passing with the good performance of soft-decision algorithms, where the soft information is stored in edge- or node-based memories. We now give the specific operations pertaining to this algorithm.

The memory to message map $\Psi_M : S_B \rightarrow S_M$ is defined as $\Psi_M(x) = \text{sgn}_r(x)$. The *check node operation* is given by

$$\begin{aligned} m^{(\ell)}_{c_j \rightarrow v_i} &= \Psi_C \left(m^{(\ell-1)}_{v_k \rightarrow c_j} : v_k \in N(c_j) \backslash v_i \right) \\ &= \prod_{v_k \in N(c_j) \backslash v_i} m^{(\ell-1)}_{v_k \rightarrow c_j}. \end{aligned} \tag{10}$$

The *variable node operation* is given by

$$\begin{aligned} X^{(\ell)}_{v_i,c_j} &= \Psi_V \left(m^{(\ell)}_{c_k \rightarrow v_i} : c_k \in N(v_i) \backslash c_j \right) \\ &= \sum_{c_k \in N(v_i) \backslash c_j} m^{(\ell)}_{c_k \rightarrow v_i}. \end{aligned} \tag{11}$$

In this case, the set of output messages from a variable node is defined as $S_X := \{-(d_v - 1), -(d_v - 3), ..., 0, ..., d_v - 3, d_v - 1\}$ for odd $d_v \geq 3$ and $S_X := \{-(d_v - 1), -(d_v - 3), ..., -1, 1, ..., d_v - 3, d_v - 1\}$ for even $d_v \geq 4$. And finally, the *memory update* is given by

$$\begin{aligned} B^{(\ell)}_{v_i,c_j} &= \Psi_B \left(B^{(\ell-1)}_{v_i,c_j}, X^{(\ell)}_{v_i,c_j} \right) \\ &= K_c \left(B^{(\ell-1)}_{v_i,c_j} + X^{(\ell)}_{v_i,c_j} \right). \end{aligned} \tag{12}$$

$K_c : \mathbb{Z} \rightarrow S_B$ is a clipping function defined as $K_c(x) = \underset{b \in S_B}{\arg\min} |x - b|$.

2.3.2 Successive Relaxation

Any standard memoryless iterative algorithm, such as BP or MS, can be turned into an SR algorithm by proper introduction of memory [13, 14]. SR algorithms can be performed in different message domains. In this work, we assume log-likelihood ratio (LLR) domain for messages (SR-LLR). Under this assumption, the SR algorithms are symmetric. We now give the specific operations pertaining to the SR-MS-LLR decoding algorithm[4].

There is no need for a memory to message map Ψ_M since the memory space S_B is the same as the message space S_M. Hence, Ψ_M as shown in Fig. 2(b) is equivalent to a pass-through. The *check node operation* is given by

$$\begin{aligned} m_{c_j \to v_i}^{(\ell)} &= \Psi_C \left(m_{v_k \to c_j}^{(\ell-1)} : v_k \in N(c_j) \backslash v_i \right) \\ &= \min \left(\left| m_{v_k \to c_j}^{(\ell-1)} \right| : v_k \in N(c_j) \backslash v_i \right) \prod_{v_k \in N(c_j) \backslash v_i} \operatorname{sgn}_{\mathrm{r}} \left(m_{v_k \to c_j}^{(\ell-1)} \right). \end{aligned} \tag{13}$$

The *variable node operation* is given by

$$\begin{aligned} X_{v_i,c_j}^{(\ell)} &= \Psi_V \left(m_{c_k \to v_i}^{(\ell)} : c_k \in N(v_i) \backslash c_j, y_i \right) \\ &= \left(\sum_{c_k \in N(v_i) \backslash c_j} m_{c_k \to v_i}^{(\ell)} \right) + K_B(y_i). \end{aligned} \tag{14}$$

Let $\chi = 2^{q-1} - 1$. In this case, the set of output messages from a variable node is defined as $S_X := \{-\chi d_v, -(\chi d_v - 1), ..., \chi d_v - 1, \chi d_v\}$. And finally, the memory update is given by

$$\begin{aligned} B_{v_i,c_j}^{(\ell)} &= \Psi_B \left(B_{v_i,c_j}^{(\ell-1)}, X_{v_i,c_j}^{(\ell)} \right) \\ &= K_c \left((1-\beta) B_{v_i,c_j}^{(\ell-1)} + \beta X_{v_i,c_j}^{(\ell)} \right), \quad \beta \in \mathbb{R}, \end{aligned} \tag{15}$$

where K_c is the same clipping function as used in (12) except that the domain is

[4]In this work we study the quantized version of SR-MS-LLR.

now the set of reals, i.e. $K_c : \mathbb{R} \rightarrow S_B$. In (15), β is called the *relaxation factor*, and can be optimized for the best performance. The optimal value of β is usually within $(0, 1)$. We note that Equation (13) reduces to Equation (10) for DD-BMP because $S_M := \{-1, 1\}$, hence the *min* function will always return 1. The variable node operation differs from the fact that for SR-MS-LLR we are always including a quantized version of the originally received message. This will play a crucial role in the density evolution analysis since it introduces more dependency between messages as will be seen in Chapters 3 and 4. The memory update for SR-MS-LLR differs slightly from DD-BMP because of the relaxation factor β.

2.3.3 General Instance of Iterative Decoding Algorithms with Memory

We can describe the above two algorithms in a more general case for any particular S_B and S_M. The initialization step as well as the check node operation remains identical, however, we define a more general variable node operation and memory update rule:

- Variable Node Operation, $\ell \geq 1$:
  $X_{v_i,c_j}^{(\ell)} = \left(\sum_{c_k \in N(v_i) \setminus c_j} m_{c_k \rightarrow v_i}^{(\ell)} \right) + \gamma K_M(y_i), \gamma \in \mathbb{R}$, where $K_M : S_O \rightarrow \mathbb{Z}$ is function which quantizes the originally received value at variable node i. If $K_M = K_B$ as in SR-MS-LLR, then $S_X := \{-T, -(T-1), ..., T-1, T\}$ where $T = D(d_v - 1)$+round$(\gamma(2^{q-1} - 1))$.[5]

- Memory Element Update, $\ell \geq 1$:
  $B_{v_i,c_j}^{(\ell)} = K_c \left(\alpha B_{v_i,c_j}^{(\ell-1)} + \beta X_{v_i,c_j}^{(\ell)} \right), \quad \alpha, \beta \in \mathbb{R}.$

This general framework is presented to stress the fact that for a particular ensemble of LDPC codes, there should exist an optimal set of parameters, α,

[5]The *round* function simply rounds the number to the nearest integer.

β, and γ in order to provide the best performance. Although iterative decoding algorithms with memory may be described in an even more general sense, this is a good starting point to convince the existence of an optimal[6] decoder with memory.

Using the algorithm notation described above, DD-BMP can be described with $S_M = \{-1, 1\}$, Ψ_M is the sgn_r function, $\alpha = \beta = 1$, and $\gamma = 0$. Since $\gamma = 0$, we do not need to define the K_M function for DD-BMP. Similarly, SR-MS-LLR can be described by having $S_B = S_M$, $\alpha = 1 - \beta$, $K_M = K_B$ and $\gamma = 1$. The function Ψ_M is omitted since it is just a pass-through, i.e. $m_{v\to c}^{(\ell)} = B_{v,c}^{(\ell)}$.

2.4 Symmetry of the Decoder and Error Probability

The analysis of iterative decoders is greatly simplified assuming that both the channel and the decoder are symmetric. The decoder symmetry conditions for variable and check nodes are defined in [18]. All the decoders considered in this work are assumed to be symmetric. Channel symmetry is met since we assume the application of binary LDPC codes over symmetric output memoryless channels. Check node symmetry states that the signs of the messages factor out in the check node operation:

$$\Psi_C(m_1, m_2, ..., m_{d_c-1}) = \Psi_C(|m_1|, |m_2|, ..., |m_{d_c-1}|) \prod_{i=1}^{d_c-1} \text{sgn}_\text{r}(m_i), \qquad (16)$$

where $m_1, m_2, ..., m_{d_c-1}$ are the messages used in the computation at some check node. This symmetry as given in Equation (16) is clearly satisfied in Equation (13)[7] since the signs naturally factor out from the definition of the function. The variable node

[6] An optimal decoder is defined as follows: for a particular LDPC code ensemble, fixed channel and fixed decoding complexity (in terms of the computational and memory requirements), an optimal decoder with memory would have the lowest probability of decoding error.

[7] Satisfaction of check node symmetry for Equation (13) implies satisfaction of check node symmetry for Equation (10).

symmetry condition has some implications on the choices of the mappings Ψ_V, Ψ_B, Ψ_M and the clipping/quantizing functions K_M and K_C:

$$\begin{aligned}
&\Psi_V(-m_0, -m_1, ..., -m_{d_v-1}) = -\Psi_V(m_0, m_1, ..., m_{d_v-1}), \\
&K_M(-y) = -K_M(y), \\
&\Psi_B(-x, -b) = -\Psi_B(x, b), \\
&\Psi_M(-x) = -\Psi_M(x),
\end{aligned} \tag{17}$$

where $m_0, m_1, ..., n_{d_v-1}$ are the messages used in the computation at a variable node, b denotes the current memory content, x is the result of the function Ψ_V, and y is the originally received message from the channel. All the equations presented in (17) satisfy what is known as sign inversion invariance [18]. Therefore we can group these equations together to form the following variable node symmetry rule:

$$\Psi_M(\Psi_B(\Psi_V(-m_0, -m_1, ..., -m_{d_v-1}), -b)) = -\Psi_M(\Psi_B(\Psi_V(m_0, m_1, ..., m_{d_v-1}), b)). \tag{18}$$

This variable node symmetry is satisfied for both DD-BMP and SR-MS-LLR since Ψ_V and Ψ_B both consist of addition (which is naturally sign inversion invariant), and all clipping and quantizing functions are odd and symmetric with respect to the origin.

With both the channel and the decoder being symmetric, we can assume, without loss of generality, that the all-zero codeword[8] is transmitted. We now formulate the equation for finding the average probability of bit error at a particular iteration ℓ. Usually for conventional density evolution, one focuses on the sign of the messages passed from variable nodes to check nodes, and the probability that the sign of the message is negative (in error). Based on this, it would be natural to use the sign of $m_{V\to C}^{(\ell)}$ as the indication of error. Since $m_{V\to C}^{(\ell)}$ is either equal to the memory content $B^{(\ell)}$ or is a deterministic function (which preserves sign inversion invariance) of $B^{(\ell)}$,

[8] A reminder to the reader that a 0 bit is converted to +1 when sent over the channel.

we can simply track the sign of the memory content. Hence, we can formulate the average probability of bit error in iteration ℓ as follows:

$$P_e^{(\ell)} = P(B^{(\ell)} < 0) + \frac{1}{2}P(B^{(\ell)} = 0)\,, \tag{19}$$

where $P(\mathcal{A})$ is used to denote the probability of an event $\mathcal{A}$.

Chapter 3

Bayesian Network Representation of Iterative Decoding Algorithms with Memory

3.1 Bayesian Networks and Conditional Independence

A Bayesian network is a graphical model that represents a set of random variables and their statistical (in)dependence via a directed acyclic graph (DAG). In this work, we use a Bayesian network to represent the dependencies among different messages and memory contents of an iterative decoding algorithm with memory. These dependencies are studied as a function of both the iteration number ℓ and the depth of the decoding tree d based on the model of Fig. 2(b). In fact, as we will see later, it is the conditional independence among different random variables that plays a crucial role in simplifying the representation and derivation of the desired distributions.

We start by some basic definitions and notations. A *discrete random variable*

X (denoted by upper case) is defined by its sample space denoted by S_X and its probability mass function denoted by $\mathbb{P}(X)$. The probability of an outcome $x \in S_X$ (denoted by lower case) will be denoted by $P(X = x)$ or simply by $P(x)$ if in the context it is obvious that $x \in S_X$. We consider a DAG G as an ordered pair $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of directed edges in G. Each node represents a random variable, and directed edges imply conditional (in)dependence among the random variables as explained in the following. Throughout this work, we use the terms "nodes" and "random variables" interchangeably. Nodes V and U are referred to as the *head* and the *tail* of the directed edge (U, V). We also call U a *parent* of V, and V a *child* of U. In general, all the parents of a node V denoted by $pa(V)$ is the set of nodes $\{V_1, V_2, ..., V_n\} \subset \mathcal{V}$ such that $(V_i, V) \in \mathcal{E}$ for $i = 1, ..., n$. Similarly, all the children of a node V denoted by $ch(V)$ is the set of nodes $\{U_1, U_2, ..., U_k\} \subset \mathcal{V}$ such that $(V, U_i) \in \mathcal{E}$ for $i = 1, ..., k$. We refer to the union of the sets $ch(V), ch(ch(V)), \ldots$ as the *descendants* of V and denote them by $de(V)$. For a DAG, it is clear that the sets $pa(V)$ and $de(V)$ are disjoint. In a DAG, we define an *undirected path* between a node U and a node V as a sequence of nodes starting with U and ending with V such that successive nodes are connected by an edge.

The *conditional independence* between two sets of random variables $\mathcal{X}$ and $\mathcal{Y}$ given a third set $\mathcal{Z}$ is defined by

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z} \iff \mathbb{P}(\mathcal{X}, \mathcal{Y} | \mathcal{Z}) = \mathbb{P}(\mathcal{X} | \mathcal{Z}) \mathbb{P}(\mathcal{Y} | \mathcal{Z}), \tag{20}$$

where $\mathbb{P}(\mathcal{X})$ and $\mathbb{P}(\mathcal{X}|\mathcal{Z})$ are the marginal distribution of $\mathcal{X}$ and the conditional distribution of $\mathcal{X}$ given $\mathcal{Z}$, respectively. *Marginal independence* between $\mathcal{X}$ and $\mathcal{Y}$ is denoted by $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}$ and is defined as in (20) where $\mathcal{Z} = \emptyset$ and $\mathbb{P}(\mathcal{X}|\emptyset) := \mathbb{P}(\mathcal{X})$.

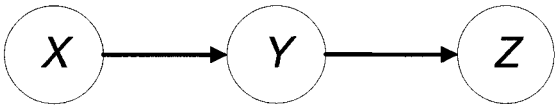


Figure 3: Markov Chain with Random Variables X, Y, and Z

We say that a DAG $G(\mathcal{V}, \mathcal{E})$ is a Bayesian network with respect to the set of random variables $\mathcal{V} = \{V_1, \ldots, V_N\}$, if the joint probability distribution of $\mathcal{V}$ can be factored as follows:

$$\mathbb{P}(\mathcal{V}) := \mathbb{P}(V_1, \ldots, V_N) = \prod_{i=1}^{N} \mathbb{P}(V_i | pa(V_i)), \tag{21}$$

or in other words, if for any $V \in \mathcal{V}$, we have

$$V \perp\!\!\!\perp \mathcal{V} \backslash \{V \cup de(V) \cup pa(V)\} \mid pa(V). \tag{22}$$

Note that in Equation (21), $\mathbb{P}(V_i | pa(V_i)) = \mathbb{P}(V_i)$ if $pa(V_i) := \emptyset$ for $i = 1, ..., N$.

As a simple example, consider a Markov Chain between random variables X, Y and Z as shown in Fig. 3. Here $\mathcal{V} := \{\mathcal{X}, \mathcal{Y}, \mathcal{Z}\}$ and $\mathcal{E} := \{(\mathcal{X}, \mathcal{Y}), (\mathcal{Y}, \mathcal{Z})\}$. Using Equation (21), we can factor the joint probability of X, Y, Z as follows:

$$\mathbb{P}(X, Y, Z) = \mathbb{P}(Z|Y)\mathbb{P}(Y|X)\mathbb{P}(X), \tag{23}$$

since the parent of Z is Y, the parent of Y is X, and X has no parents.

To go beyond the basic conditional independence relations given by the local constraints (22), and to be able to ascertain whether a general conditional independence statement $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z}$ is implied by a given DAG, where $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ are disjoint sets of random variables, we need the following definitions:

Definition 1 (Undirected path blocked). *An undirected path between a node V and a node W is* blocked *by a set of nodes $\mathcal{X}$ if the path includes a node U such that either: (a) U has head-to-head arrows along the path ($\longrightarrow U \longleftarrow$), and neither U nor any of its descendants is in $\mathcal{X}$, or (b) U does not have head-to-head arrows along the path and U is in $\mathcal{X}$.*

Definition 2 (D-Separation). *Given a set of nodes $\mathcal{Z}$, if every undirected path from any node in $\mathcal{X}$ to any node in $\mathcal{Y}$ is blocked by $\mathcal{Z}$, then $\mathcal{X}$ is said to be* d-separated *from $\mathcal{Y}$ by $\mathcal{Z}$.*

Fact 1 [29]: In a DAG with disjoint sets of nodes $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, if $\mathcal{Z}$ d-separates $\mathcal{X}$ from $\mathcal{Y}$, we have $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}|\mathcal{Z}$.

In fact, it can be seen that characterization of (21) for a joint distribution is equivalent to the satisfaction of all the conditional independence properties implied by the DAG through d-separation [29].

The notion of *d-separation* can be better understood visually through what is known as *Bayes' Ball algorithm* [30]. For this we introduce a set of rules for Bayes' Ball algorithm shown in Fig. 4. Shaded nodes indicate that the particular random variable is observed, i.e. conditioned on. The models in Fig. 4 show all the possible configurations of a segment of a path between two nodes. If you were to pass a "Bayes' Ball" through one of the models, you either have a blockage, denoted by $\rightarrow |$, or a pass-through, denoted by $\rightarrow$. For some DAG $G = (\mathcal{V}, \mathcal{E})$, let $U, V \in \mathcal{V}$ be two particular nodes, and $O \subseteq \mathcal{V} \backslash \{U, V\}$ be a set of nodes which are observed. Then we can say that U is conditionally independent of V given O if it is not possible for a Bayes' Ball to travel through any path connecting U and V. In the case that $O = \{\emptyset\}$, none of the nodes are observed (shaded). Hence, Bayes' Ball algorithm will

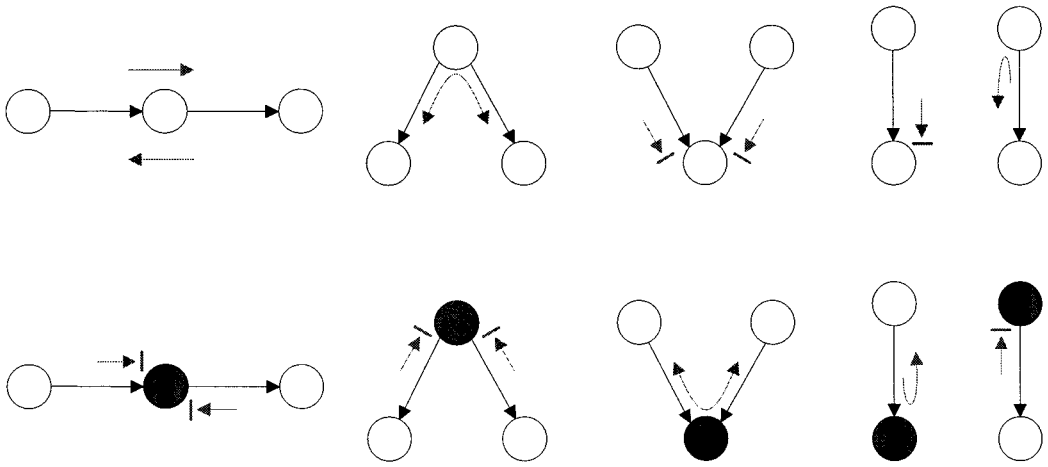


Figure 4: Rules for Bayes' Ball Algorithm

give a statement concerning the marginal independence of U and V.

Example 1. *Consider the graphical model shown in Fig. 5 defined by* $\mathcal{V} := \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$ *and* $\mathcal{E} := \{(\mathcal{A}, \mathcal{B}), (\mathcal{A}, \mathcal{C}), (\mathcal{D}, \mathcal{B}), (\mathcal{D}, \mathcal{C}), (\mathcal{C}, \mathcal{E})\}$. *Again, following Equation (21) we can show the factorization of the joint probability distribution:*

$$\mathbb{P}(A, B, C, D, E) = \mathbb{P}(E|C)\mathbb{P}(C|A, D)\mathbb{P}(B|A, D)\mathbb{P}(A)\mathbb{P}(D) \tag{24}$$

Consider nodes A and D. There are three distinct paths from A to D: $A \rightarrow B \leftarrow D$, $A \rightarrow C \leftarrow D$, and $A \rightarrow C \rightarrow E \leftarrow C \leftarrow D$. *There are infinitely many more paths but they do not need to be considered since they would contain at least one of the distinct paths previously defined. Using Bayes's Ball algorithm on each of the three paths given no observed nodes, we would have a blockage at B, C, and E, respectively. Hence, A is independent of D. Similarly, we can say that B is conditionally independent of C given A and D.*

Being able to recognize conditional independence from these models is extremely helpful when faced with the task of finding the probability mass function of a particular random variable as is the goal of *density evolution.*

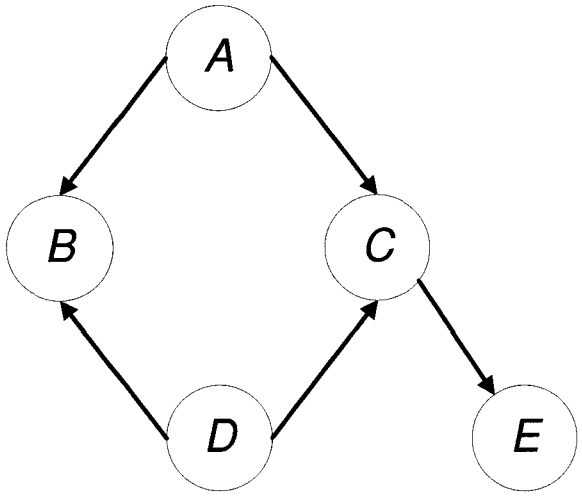


Figure 5: *Sample Graphical Model with Random Variables A, B, C, D, and E*

In the following section, we construct the Bayesian network corresponding to the model of Fig. 2(b). This Bayesian network will serve as a guide to deriving the distributions required for the error performance analysis of the decoder. More specifically, we will use Fact 1 for efficient representation and calculation of the distributions.

3.2 Bayesian Networks of Iterative Decoders with Memory

Consider the directed neighborhood $\mathcal{N}_d(\overrightarrow{e})$ of depth d of the directed edge $\overrightarrow{e}$ [18] for a (d_v, d_c) Tanner graph. Assuming a cycle-free graph, the neighborhood is tree-like. Fig. 6 shows an example of $\mathcal{N}_2(\overrightarrow{e})$, where $\overrightarrow{e} = (V, C)$, for a $(3, 4)$ Tanner graph. Memory elements only appear at even depths on the tree, and they are shown by shaded circles. We use the notation $B_{d/2}$ to denote the memory elements that appear in $\mathcal{N}_d((V, C)) \backslash \mathcal{N}_{d-2}((V, C))$. It is easy to see $|B_i| = \{(d_v - 1)(d_c - 1)\}^i$, for $i = 1, 2, \ldots$. In Fig. 6, $|B_1| = 6$.

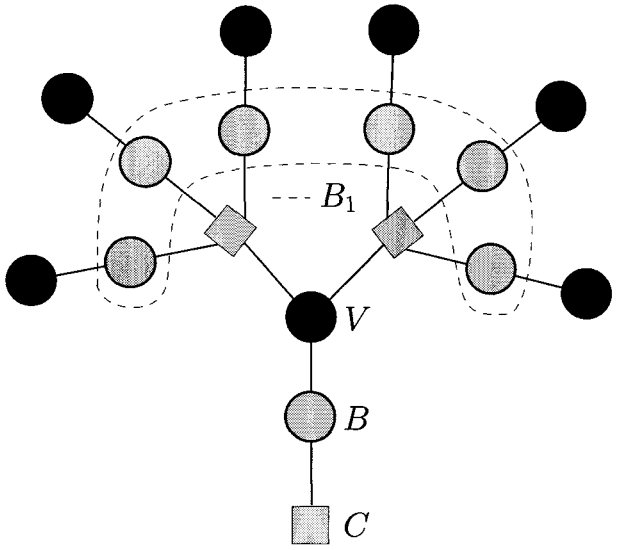


Figure 6: Tree-like neighborhood of the edge (V, C) in a regular (3,4) LDPC code

Based on the principle of extrinsic message-passing, one can see that $X^{(\ell)}$ in Fig. 2(b) is a deterministic function of $B_1^{(\ell-1)}$ and M_0. Moreover, as we discussed before, $B^{(\ell)}$ is a function of $B^{(\ell-1)}$ and $X^{(\ell)}$. In addition, $B_i^{(\ell)}$ depends on $B_i^{(\ell-1)}$ and $B_{i+1}^{(\ell-1)}$. Fig. 6 does not give a clear picture of how the dependency relationships occur between messages. This is because it is only a snapshot of the random variables at a particular iteration. It is more useful to describe the dependencies of the described random variables above through a Bayesian network which shows the evolution through iteration ℓ. The Bayesian network corresponding to these relationships is given in Fig. 7. In this figure, $M_{0,i}$ is defined as the initial messages corresponding to the variable nodes in $\mathcal{N}_{2i}((V,C))\backslash\mathcal{N}_{2i-2}((V,C))$ (see Fig. 6). The dashed edges between M_0 and nodes $X^{(\ell)}$ would disappear if $X^{(\ell)}$ is not a function of M_0. This is for example the case for DD-BMP.

In Fig. 7 we have shown all the random variables which have an effect on the memory content distribution at iterations 1, 2, and 3. It is also important to note that each node is a function of its parents. Consider a section of Fig. 7 as shown in

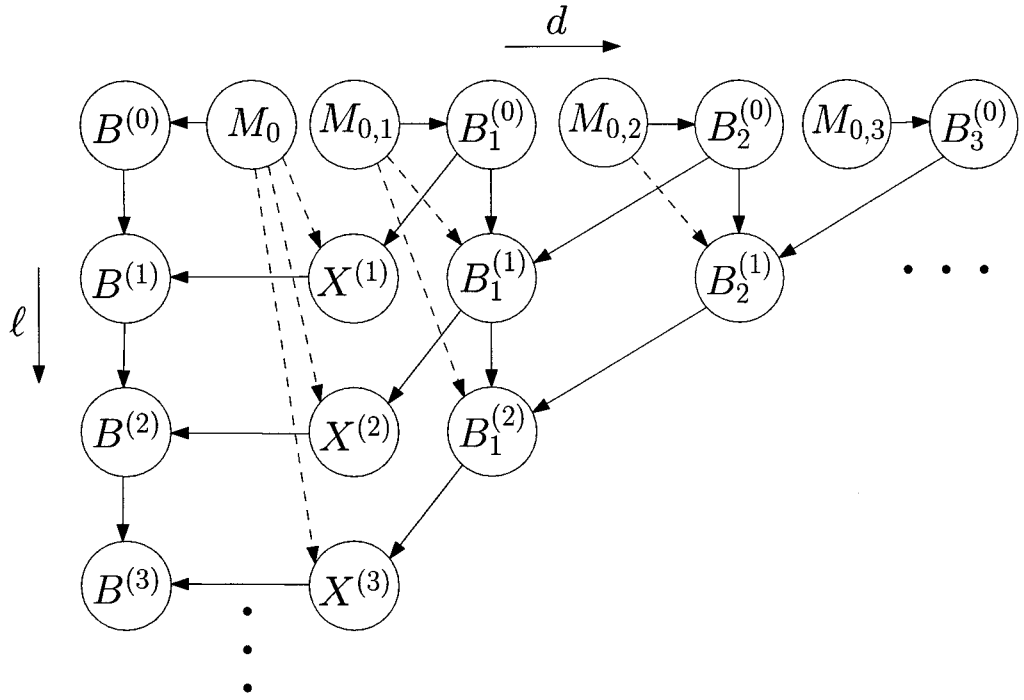


Figure 7: Bayesian network of iterative decoders with memory based on the model of Fig. 2(b).

Fig. 8. This basic graphical model structure repeats itself for each iteration ℓ. The following relationships exist between the nodes shown in Fig. 8:

Let $B_1^{(\ell-1)} = \{B_{1,1}, ..., B_{d_v-1,1}, ..., B_{1,d_c-1}, ..., B_{d_v-1,d_c-1}\}$ which a set of

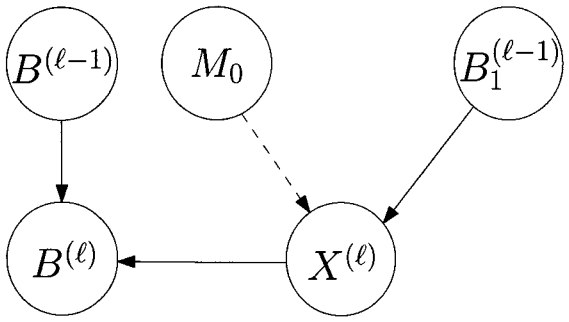


Figure 8: A building block of the Bayesian network in Fig. 7.

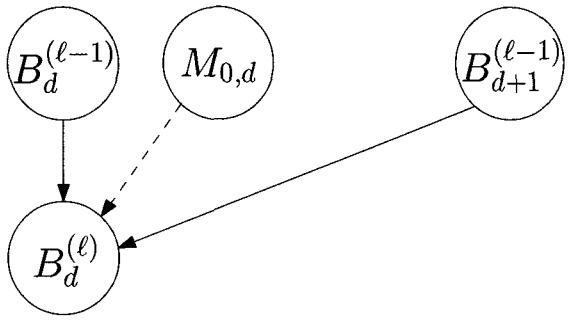


Figure 9: Another building block of the Bayesian network in Fig. 7.

$(d_v - 1)(d_c - 1)$ i.i.d. random variables. Then we having the following:

$$\begin{aligned} X^{(\ell)} &= \Psi_V(\Psi_C(B_{1,1}, ..., B_{d_v-1,1}), ..., \Psi_C(B_{1,d_c-1}, ..., B_{d_v-1,d_c-1}), M_0), \\ B^{(\ell)} &= \Psi_B\big(B^{(\ell-1)}, X^{(\ell)}\big). \end{aligned} \tag{25}$$

We can also look at a set of memory nodes at a particular depth d and iteration ℓ as shown in Fig. 9. However, this is simply a compact way of representing $((d_v - 1)(d_c - 1))^d$ independent copies of the structure shown in Fig. 8.

We now show how one might go about finding the probability mass function (p.m.f.) of $B^{(3)}$ using the Bayesian network in Fig. 7. For this example we are assuming the case for DD-BMP in which the dashed edges in Fig. 7 are removed. $B^{(3)}$ is a deterministic function of $B^{(2)}$ and $X^{(3)}$ (through the function Ψ_B). Hence we could write the following:

$$\mathbb{P}\left(B^{(3)}\right) = \sum_{b^{(2)}} \sum_{x^{(3)}} \mathbb{P}\left(B^{(3)} | B^{(2)}, X^{(3)}\right) \mathbb{P}\left(B^{(2)}, X^{(3)}\right). \tag{26}$$

However, finding the joint distribution of $B^{(2)}$ and $X^{(3)}$ is not trivial since they are not independent. In Fig. 7 we can use Bayes' Ball algorithm to show that that they are dependent. We consider two paths going from node $B^{(2)}$ to node $X^{(3)}$ with one passing through $X^{(1)}$ and the other through $X^{(2)}$. The first path is $B^{(2)} \leftarrow X^{(2)} \leftarrow B_1^{(1)} \rightarrow B_1^{(2)} \rightarrow X^{(3)}$ and the other one being

$B^{(2)} \leftarrow B^{(1)} \leftarrow X^{(1)} \leftarrow B_1^{(0)} \rightarrow B_1^{(1)} \rightarrow B_1^{(2)} \rightarrow X^{(3)}$. Using our rules of conditional independence from Fig. 4, there is no blockage in either path and hence the two random variables are dependent. However, $B^{(2)}$ is conditionally independent of $X^{(3)}$ given $\left\{X^{(1)}, X^{(2)}\right\}$. In other words, $\left\{X^{(1)}, X^{(2)}\right\}$ d-separates nodes $B^{(2)}$ and $X^{(3)}$. In fact, this is the minimum separator for $B^{(2)}$ and $X^{(3)}$. We will make use of these techniques in the full analysis of calculating $\mathbb{P}\left(B^{(n)}\right)$.

It is also possible to describe $B^{(3)}$ as a function of i.i.d. random variables. $B^{(3)}$ is a constant given $B_{0\rightarrow 3}^{(0)} := \left\{B^{(0)}, B_1^{(0)}, B_2^{(0)}, B_3^{(0)}\right\}$. The number of memory nodes contained in the conditioning set $B_{0\rightarrow 3}^{(0)}$ for a (d_v, d_c) regular code is given by $\sum_{i=0}^{3}\left((d_v-1)(d_c-1)\right)^i = \frac{1-((d_v-1)(d_c-1))^4}{1-(d_v-1)(d_c-1)}$. We can then find the probability mass function of $B^{(3)}$ as follows:

$$\mathbb{P}\left(B^{(3)}\right) = \sum_{b \in B_{0\rightarrow 3}^{(0)}} \mathbb{P}\left(B^{(3)} | B_{0\rightarrow 3}^{(0)} = b\right) P\left(B_{0\rightarrow 3}^{(0)} = b\right). \tag{27}$$

However, this requires summing over $|S_B|^{\frac{1-((d_v-1)(d_c-1))^4}{1-(d_v-1)(d_c-1)}}$ elements. In general, calculating the p.m.f of $B^{(\ell)}$ as of a function of i.i.d. random variables $B_{0\rightarrow \ell}^{(0)}$ would require summing over approximately $|S_B|^{((d_v-1)(d_c-1))^\ell}$ elements. Clearly the calculation complexity is too high motivating us to find the joint distribution of dependent random variables.

With the help of the Bayesian network of Fig. 7, we now proceed to giving the simple representation and derivation of the probability mass function of $B^{(\ell)}$ denoted as $\mathbb{P}\left(B^{(\ell)}\right)$ or $P_B^{(\ell)}$ for the general decoding algorithm with memory as described in Chapter 2.

Chapter 4

Full Analysis

4.1 Memory Initialization

We consider the transmission of the all-zero codeword over an AWGN memoryless channel. The received value y_i at each variable node is i.i.d. with a Gaussian distribution of mean 1 and variance σ^2. Assuming that the AWGN channel has a power spectral density of $N_0/2$, we have $\sigma = 1/\sqrt{2RE_b/N_0}$, where E_b and R denote the energy per information bit and the code rate, respectively.

The received values y_i at each variable node are clipped symmetrically at a threshold c_{th} and then uniformly quantized in the range $[-c_{th}, c_{th}]$. Using q quantization bits, there are $2^q - 1$ quantization intervals, symmetric with respect to the origin. Each quantization interval has length $\Delta = 2c_{th}/(2^q - 1)$. $K_B(y_i)$ is used to denote the quantized value of y_i. For every edge in the bipartite graph of the code, there is a q-bit memory associated with it. For each variable node v_i, the memory elements between v_i and all d_c adjacent check nodes are initialized to $K_B(y_i)$. Since the memory content is initialized by a function of y_i and the channel is memoryless, the memory contents on each edge of the associated Tanner graph of the code are independent of each other and identically distributed.

Finding $\mathbb{P}\left(B^{(0)}\right)$ requires finding the distribution of $K_B(y_i)$.

$$\begin{aligned} P\left(B^{(0)}=b\right) = & \sum_{i=-(2^{q-1}-2)}^{2^{q-1}-2} \delta\left[i-b\right]\xi\left(i\right) \\ & + \delta\left[2^{q-1}-1-b\right] Q\left(\frac{\Delta(2^q-3)-2}{2\sigma}\right) \\ & + \delta\left[-2^{q-1}+1-b\right] Q\left(\frac{\Delta(2^q-3)+2}{2\sigma}\right),\ b \in S_B \end{aligned} \tag{28}$$

where

$$\xi(x) = Q\left(\frac{\Delta(2x-1)-2}{2\sigma}\right) - Q\left(\frac{\Delta(2x+1)-2}{2\sigma}\right),$$

$$Q(\alpha) = \frac{1}{\sqrt{2\pi}} \int_{\alpha}^{\infty} e^{-\frac{x^2}{2}} dx,$$

and

$$\delta[x] = \begin{cases} 1,\ x = 0 \\ 0,\ x \neq 0 \end{cases}$$

Example 2. *Consider the number of quantization bits $q = 3$ and $c_{th} = 3$. Therefore, $S_B = \{-3, -2, -1, 0, 1, 2, 3\}$. If the transmitted signal is +1 and sent over an AWGN channel with mean 0 and variance(σ^2) 2, then the received signal y will have a Gaussian distribution with mean 1 and variance 2 as shown in Fig. 10 . Included are the quantization intervals showing the mapping $y \xrightarrow{K_B} S_B$. The corresponding probability of each element is simply the area of its corresponding region. For example, $P\left(B^{(0)}=2\right)$ is the area of the shaded region in Fig. 10.*

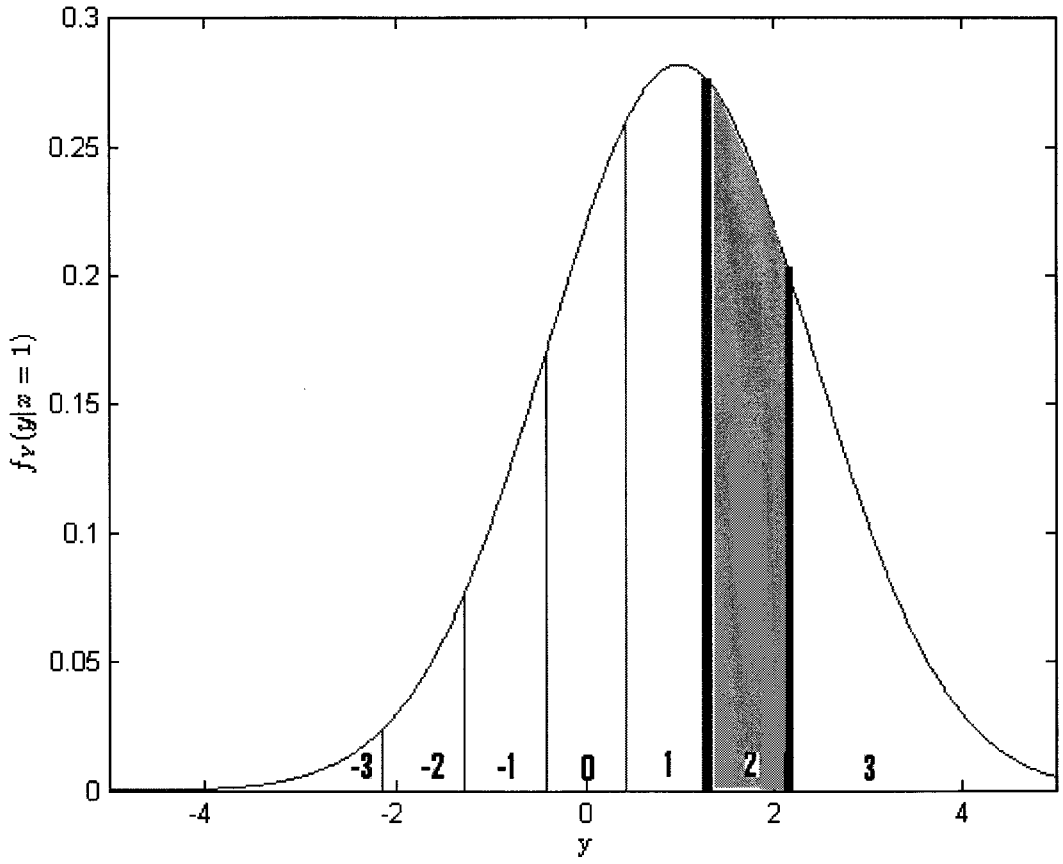


Figure 10: *Probability Density Function of the received signal given $x = +1$ and the 3-bit quantization scheme*

4.2 $\mathbb{P}\left(B^{(1)}\right)$: Memory content distribution after the first iteration

We will see shortly that finding $\mathbb{P}\left(B^{(\ell)}\right)$ for $\ell > 1$ is much more involved as we will be dealing with dependent random variables. For the case of $\ell = 1$, we show how to calculate the probability mass function of the memory content. For a particular memory node B adjacent to variable node V_B and check node C_B, we introduce the following notations. Let the incoming messages to V_B at iteration ℓ be denoted by the set $\left\{U_i^{(\ell)} : i = 1, ..., d_v - 1\right\}$. Let $\{C_i : i = 1, ..., d_v - 1\}$ be the neighbouring check nodes to V_B excluding C_B. The incoming messages to C_i will be denoted by the set $\left\{m_{i,j}^{(\ell-1)} : j = 1, ..., d_c - 1\right\}$. Since all the received messages at a particular node are i.i.d., $B^{(1)}$ is a function of i.i.d. random variables.

$$\begin{aligned} B^{(1)} &= \Psi_B\left(B^{(0)}, X^{(1)}\right) \\ &= \Psi_B\left(B^{(0)}, \Psi_V(U_1, ..., U_{d_v-1}, M_0)\right) \\ &= \Psi_B\left(B^{(0)}, \Psi_V\left(\Psi_C\left(m_{1,1}^{(0)}, ..., m_{1,d_c-1}^{(0)}\right), ..., \Psi_C\left(m_{d_v-1,1}^{(0)}, ..., m_{d_v-1,d_c-1}^{(0)}\right), M_0\right)\right) \end{aligned} \tag{29}$$

Since $X^{(1)}$ is the addition of $d_v - 1$ i.i.d. random variables $U_i^{(1)}$ and M_0, we have the following (note that $\otimes$ is used as the convolution operator):

$$\mathbb{P}\left(X^{(1)}\right) = \mathbb{P}\left(U_1{}^{(1)}\right) \otimes \mathbb{P}\left(U_2{}^{(1)}\right) \otimes \cdots \otimes \mathbb{P}\left(U_{d_v-1}{}^{(1)}\right) \otimes \mathbb{P}\left(M_0\right). \tag{30}$$

For the case of DD-BMP, M_0 is not included in the calculation of $X^{(1)}$, therefore $\mathbb{P}\left(X^{(1)}\right)$ is simply a binomial distribution. The p.m.f. of $U_i^{(1)}$ can also be easily found since it is a function of $d_c - 1$ i.i.d. random variables $m_{i,j}^{(0)}$. Now that we have the p.m.f. of $B^{(0)}$ and $X^{(1)}$, we focus on the memory update function as defined in section 2.3.3 in order to find $\mathbb{P}\left(B^{(1)}\right)$. We define $B_{uc}^{(1)}$ be the unclipped memory content, i.e.

before it has passed through the K_c clipping function.

$$\begin{aligned} B^{(1)} &= \Psi_B\big(B^{(0)}, X^{(1)}\big) \\ &= K_c\big(\alpha B^{(0)} + \beta X^{(1)}\big) \\ &= K_c\big(B_{uc}^{(1)}\big). \end{aligned} \tag{31}$$

Since $B_{uc}^{(1)}$ is the addition of two scaled independent random variables, $\mathbb{P}\left(B_{uc}^{(1)}\right) = \mathbb{P}\left(\alpha B^{(0)}\right) \otimes \mathbb{P}\left(\beta X^{(1)}\right)$. Hence we can find $\mathbb{P}\left(B^{(1)}\right)$ as follows:

$$\begin{aligned} P\big(B^{(1)} = b\big) &= P\big(B_{uc}^{(1)} = b\big), b \in \left\{-(2^{q-1}-2), -(2^{q-1}-3), ..., 2^{q-1}-2\right\}, \\ P\big(B^{(1)} = -(2^{q-1}-1)\big) &= P\big(B_{uc}^{(1)} \leq -(2^{q-1}-1)\big), \\ P\big(B^{(1)} = 2^{q-1}-1\big) &= P\big(B_{uc}^{(1)} \geq -2^{q-1}-1\big) \end{aligned} \tag{32}$$

Example 3. *Consider we are applying the DD-BMP algorithm. In this case,* $S_M = \{-1, 1\}$. *Let* $\delta = P\big(\mathrm{sgn_r}\big(B^{(0)}\big) = -1\big)$. *Each incoming message* $m_{i,j}^{(0)}$ *is i.i.d. with probabilities* $P\left(m_{i,j}^{(0)} = -1\right) = \delta$ *and* $P\left(m_{i,j}^{(0)} = 1\right) = 1 - \delta$. *This is analogous to writing* $\mathbb{P}\left(m_{i,j}^{(0)}\right) = (\delta, 1-\delta)$. *We will use the fact that for any Bernoulli random variable* X_{Br} *with sample space* $\{-1, 1\}$ *and p.m.f.* $\mathbb{P}(X_{Br}) = (\delta, 1-\delta)$, $\mathbb{E}[X_{Br}] = 1 - 2\delta$ *and* $P(X_{Br} = 1) = \frac{\mathbb{E}[X_{Br}]+1}{2}$. *Since the check node processing is simply the product function applied over i.i.d. random variables, we can write the following:*

$$\begin{aligned} U_i^{(1)} &= \prod_{j=1}^{d_c-1} m_{i,j}^{(0)} \\ \mathbb{E}\left[U_i^{(1)}\right] &= \mathbb{E}\left[\prod_{j=1}^{d_c-1} m_{i,j}^{(0)}\right] \\ &= \prod_{j=1}^{d_c-1} \mathbb{E}\left[m_{i,j}^{(0)}\right] \\ &= \prod_{j=1}^{d_c-1} (1-2\delta) \\ &= (1-2\delta)^{d_c-1}. \end{aligned} \tag{33}$$

We now have all the information we need to determine the distribution of $U_i^{(1)}$*, given as follows:*

$$\mathbb{P}\left(U_i^{(1)}\right) = \left(\frac{1}{2} - \frac{1}{2}(1-2\delta)^{d_c-1}, \frac{1}{2} + \frac{1}{2}(1-2\delta)^{d_c-1}\right). \tag{34}$$

4.3 $\mathbb{P}\left(B^{(\ell)}\right)$: Memory content distribution after ℓ iterations, $\ell > 1$

$$\begin{aligned}\mathbb{P}\left(B^{(\ell)}\right) &= \sum_{x^{(\ell)}} \sum_{b^{(\ell-1)}} \mathbb{P}\left(B^{(\ell)}, B^{(\ell-1)}, X^{(\ell)}\right) \\ &= \sum_{x^{(\ell)}} \sum_{b^{(\ell-1)}} \mathbb{P}\left(B^{(\ell)} | B^{(\ell-1)}, X^{(\ell)}\right) \mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)\end{aligned} \tag{35}$$

Since $B^{(\ell)}$ is a deterministic function of $B^{(\ell-1)}$ and $X^{(\ell)}$, we have the following:

$$\begin{aligned}&P\Big(B^{(\ell)} = b | B^{(\ell-1)} = b', X^{(\ell)} = x\Big), \quad b, b' \in S_B, x \in S_X \\ &= \begin{cases} 1, & \text{if } b = \Psi_B\left(b', x\right) \\ 0, & \text{otherwise} \end{cases}\end{aligned} \tag{36}$$

We can then re-write Equation (35) as follows:

$$P\big(B^{(\ell)} = b\big) = \sum_{(b',x)|b=\Psi_B\left(b',x\right)} P\Big(B^{(\ell-1)} = b', X^{(\ell)} = x\Big) \tag{37}$$

The total number of additions required using Equation (37) is $(\min\{|S_X|, |S_B|\} - 1) \times |S_B|$. The argument of $\min\{|S_X|, |S_B|\}$ comes from the fact that if $|S_X| < |S_B|$ (implies $S_X \subset S_B$), then for each $x \in S_X$, there exists one $b' \in S_B$ such that $b = \Psi_B\left(b', x\right), b \in S_B$. The same argument can be used for the case where $|S_B| < |S_X|$. For DD-BMP, $|S_X| = d_v$ and $|S_B| = 2^q - 1$, thus the number

of additions is represented by $O(d_v 2^q)$. For SR-MS-LLR, $|S_X| = ((2^q - 2)d_v + 1)$ and $|S_B| = 2^q - 1$, thus the number of additions is given by $O(d_v 4^q)$. However, the joint distribution $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$ is not trivial to find since $B^{(\ell-1)}$ and $X^{(\ell)}$ are dependent random variables.

Subsequent analysis will be split up into two cases. In the first case, we analyze the calculation of $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$ where $\gamma = 0$. This is true when analyzing the DD-BMP algorithm. In the second case, we assume that $\gamma \neq 0$ which is equivalent to analyzing the SR-MS-LLR algorithm.

4.4 Calculating $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$ for $\gamma = 0$

In the remainder of this work we will often be dealing with random vectors. A *random vector* is defined by $X^{(j\to k)} = \left\{X^{(j)}, X^{(j+1)}, ..., X^{(k-1)}, X^{(k)}\right\}$, $j < k$, where each $X^{(i)}, j \leq i \leq k$, is a random variable with the same sample space S_X.

For algorithmic purposes, we may represent a probability mass function by a table, which is an array of numbers representing the probability of a particular outcome. For example, the conditional probability mass function $\mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(\ell)}\right)$ can be represented by a table A which is a 3 dimensional array of size $|S_B| \times |S_B| \times |S_X|$. We assume that indexing of these tables begins at 1. Thus, for a fixed value of $B^{(\ell-1)} = b$, $\mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)} = b, X^{(\ell)}\right)$ is represented by $A(:, b + \max(S_B) + 1, :)$.

When $\gamma = 0$, the variable node operation does not include any part of the originally received message. Hence, no edges exist between M_0 and any of the $X^{(i)}$'s as seen in the graphical model of Fig. 11.

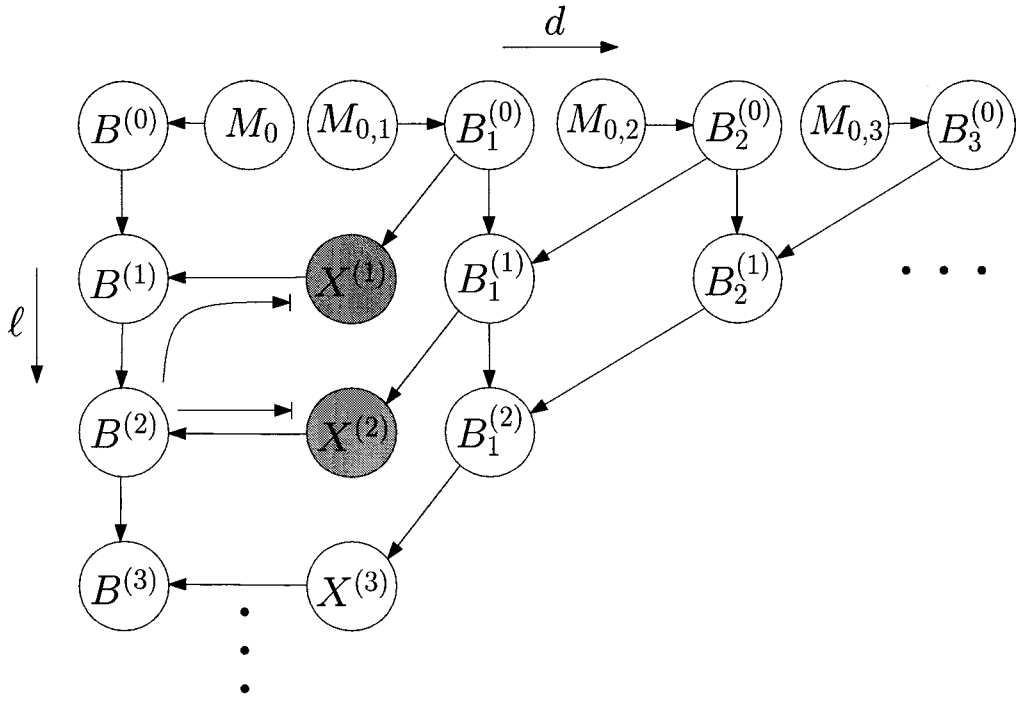


Figure 11: Bayesian network of iterative decoders with memory with $\gamma = 0$. $\{X^{(1)}, X^{(2)}\}$ d-separates $B^{(2)}$ from $X^{(3)}$.

In order to solve for $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$, we are looking for a set of random variables Ω_ℓ so that $B^{(\ell-1)} \perp\!\!\!\perp X^{(\ell)}|\Omega_\ell$. Using the *Total Probability Theorem*, we can then find the joint distribution as follows:

$$\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right) = \sum_{\omega_\ell \in \Omega_\ell} \mathbb{P}\left(B^{(\ell-1)}|\Omega_\ell = \omega_\ell\right)\mathbb{P}\left(X^{(\ell)}|\Omega_\ell = \omega_\ell\right)P(\Omega_\ell = \omega_\ell). \tag{38}$$

There are many Ω_ℓ that would satisfy Equation (38), however, what makes a particular Ω_ℓ a good choice? In terms of complexity, it would be ideal for the sample space of Ω_ℓ to be small to reduce the number of elements we need to sum over. However, more importantly, Ω_ℓ must be chosen so that we can compute $\mathbb{P}\left(B^{(\ell-1)}|\Omega_\ell\right)$, $\mathbb{P}\left(X^{(\ell)}|\Omega_\ell\right)$, and $\mathbb{P}\left(\Omega_\ell\right)$ using the same methods for all $\ell > 1$. This would be an iterative approach, so that we may reuse the distributions computed in iteration ℓ to compute distributions at iteration $\ell + 1$.

First we give an example of a poor choice of Ω_ℓ. $B^{(\ell-1)}$ is a deterministic function of $M_{0,0\to\ell-1} := \{M_0, M_{0,1}, M_{0,2}, ..., M_{0,\ell-1}\}$ and $X^{(\ell)}$ is a deterministic function of $M_{0,1\to\ell}$. Since all the single random variables in $M_{0,0\to\ell}$ are i.i.d., $B^{(\ell-1)}$ is independent of $X^{(\ell)}$ given $M_{0,1\to\ell-1}$. Therefore, in this case $\Omega_\ell = M_{0,1\to\ell-1}$ which is a random vector consisting of $\frac{(d_v-1)(d_c-1)-((d_v-1)(d_c-1))^\ell}{1-(d_v-1)(d_c-1)}$ random variables. The number of random variables grows exponentially with ℓ which is not desirable since the number of computations in this case will grow doubly exponential. We thus look for a better choice of Ω_ℓ.

Through the use of Bayes' Ball Algorithm, we can make the following claims:

$$\begin{aligned} B^{(0)} &\perp\!\!\!\perp X^{(1)} \\ B^{(k)} &\perp\!\!\!\perp X^{(k+1)} | X^{(1\to k)},\ k > 1. \end{aligned} \tag{39}$$

For example, we have shown in Fig. 11 that $B^{(2)} \perp\!\!\!\perp X^{(3)} | \{X^{(1)}, X^{(2)}\}$ since through Bayes' Ball algorithm, you would have a blockage at node $X^{(1)}$ and at node $X^{(2)}$. Therefore,

$$\begin{aligned} &\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right) \\ &= \sum_{x^{(1\to\ell-1)}} \mathbb{P}\left(B^{(\ell-1)} | X^{(1\to\ell-1)}\right) \mathbb{P}\left(X^{(\ell)} | X^{(1\to\ell-1)}\right) \mathbb{P}\left(X^{(1\to\ell-1)}\right) \\ &= \sum_{x^{(1\to\ell-1)}} \mathbb{P}\left(B^{(\ell-1)} | X^{(1\to\ell-1)}\right) \mathbb{P}\left(X^{(1\to\ell)}\right). \end{aligned} \tag{40}$$

$X^{(1\to\ell-1)}$ is a good choice for Ω_ℓ since the number of random variables grows only linearly in ℓ. From Equation (40) we can see that computing $\mathbb{P}\left(B^{(\ell)}\right)$ requires finding a conditional probability mass function containing $|S_B| \times |S_X|^{\ell-1}$ elements and a joint distribution containing $|S_X|^\ell$ elements. Therefore, the size of the tables representing probability mass functions with respect to the number of elements grows exponentially

with the number of iterations. In terms of computational complexity, the number of additions and multiplications are both bounded by $O\Big(|S_X|^{\ell-1} \times |S_B|\Big)$.

4.4.1 Calculating $\mathbb{P}\left(B^{(\ell)}|X^{(1\to\ell)}\right)$

For $\ell \geq 2$ we have the following expression:

$$\begin{aligned}
&\mathbb{P}\left(B^{(\ell)}|X^{(1\to\ell)}\right) \\
&= \sum_{b^{(\ell-1)}} \mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(1\to\ell)}\right)\mathbb{P}\left(B^{(\ell-1)}|X^{(1\to\ell)}\right) \\
&= \sum_{b^{(\ell-1)}} \mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(\ell)}\right)\mathbb{P}\left(B^{(\ell-1)}|X^{(1\to\ell-1)}\right),
\end{aligned} \tag{41}$$

where at the last step, we have used $B^{(\ell)} \perp\!\!\!\perp X^{(1\to\ell-1)}|\{X^{(\ell)}, B^{(\ell-1)}\}$ and $B^{(\ell-1)} \perp\!\!\!\perp X^{(\ell)}|X^{(1\to\ell-1)}$, both based on Fact 1. For completeness, we can write the following recurrence relation:

$$\mathbb{P}\left(B^{(\ell)}|X^{(1\to\ell)}\right) = \begin{cases} \sum_{b^{(\ell-1)}} \mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(\ell)}\right)\mathbb{P}\left(B^{(\ell-1)}|X^{(1\to\ell-1)}\right), & \ell \geq 2 \\ \sum_{b^{(0)}} \mathbb{P}\left(B^{(1)}|B^{(0)}, X^{(1)}\right)\mathbb{P}\left(B^{(0)}\right), & \ell = 1 \\ \mathbb{P}\left(B^{(0)}\right), & \ell = 0 \end{cases} \tag{42}$$

First let us consider the "naive" way of calculating Equation (41). In this case, we assume that $0 < \mathbb{P}\left(B^{(\ell)} = b|B^{(\ell-1)} = b', X^{(\ell)} = x\right) < 1$ for $b, b' \in S_B, x \in S_X$.

inputs:

- $\mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(\ell)}\right) \longrightarrow A_{|S_B|\times|S_B|\times|S_X|}$
- $\mathbb{P}\left(B^{(\ell-1)}|X^{(1\to\ell-1)}\right) \longrightarrow B_{|S_B|\times|S_X|^{\ell-1}}$

outputs:

- $\mathbb{P}\left(B^{(\ell)}|X^{(1\rightarrow\ell)}\right) \longrightarrow C_{|S_B|\times|S_X|\times|S_X|^{\ell-1}}$

1: **for** $i := 1$ **to** $|S_B|$ **do** {loop through $B^{(\ell-1)}$}
2: **for** $j := 1$ **to** $|S_B|$ **do** {loop through $B^{(\ell)}$}
3: **for** $k := 1$ **to** $|S_X|$ **do** {loop through $X^{(\ell)}$}
4: $C(j,k,:) \leftarrow C(j,k,:) + A(j,i,k) \times B(i,:)$
5: **end for**
6: **end for**
7: **end for**

The above algorithm requires a total of $|S_B|^2 \times |S_X|^{\ell} \times |S_X|^{\ell-1} = |S_B|^2 \times |S_X|^{2\ell-1}$ multiplications and additions. In terms of memory requirements, each element in tables A, B and C is a double-precision floating point which consists of 8 bytes. A more efficient approach would be to realize that $\mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(\ell)}\right)$ contains only the values 0 or 1 in order to reduce the number of additions and remove the need for all multiplications. This also reduces the memory requirements for table A since each element can be represented by a single logical bit. Here is the pseudo code for the less complex algorithm:

inputs:

- $\mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(\ell)}\right) \longrightarrow A_{|S_B|\times|S_B|\times|S_X|}$
- $\mathbb{P}\left(B^{(\ell-1)}|X^{(1\rightarrow\ell-1)}\right) \longrightarrow B_{|S_B|\times|S_X|^{\ell-1}}$

outputs:

- $\mathbb{P}\left(B^{(\ell)}|X^{(1\rightarrow\ell)}\right) \longrightarrow C_{(|S_B|\times|S_X|)\times|S_X|^{\ell-1}}$

1: **for** $i := 1$ **to** $|S_B|$ **do** {loop through $B^{(\ell-1)}$}
2: $A' \leftarrow A(:,i,:)$
3: $B' \leftarrow B(i,:)$

4: $B'' \leftarrow$repmat$(B', [|S_X| \ 1])$ $\{B''(j,:) = B', 1 \leq j \leq |S_X|\}$

5: $index \leftarrow$find$(A' == 1)$ $\{|index| = |S_X|\}$

6: $C(index,:) \leftarrow C(index,:) + B''$

7: **end for**

In line 4, B'' is a tiled version of B', i.e. B'' is B' replicated $|S_X|$ times. In line 5, *index* contains the positions in table A' which are non-zero (one). We can see that in line 6 we require $|S_X|^\ell$ additions, for a total of $|S_B| \times |S_X|^\ell$ additions. Therefore we have removed the need for all multiplications and reduced the number of additions by a factor of $|S_B|$. It is also important to note that $\mathbb{P}\left(B^{(\ell)}|B^{(\ell-1)}, X^{(\ell)}\right)$ is iteration-invariant so it does not need to be computed for each new iteration ℓ.

4.4.2 Calculating $\mathbb{P}\left(X^{(1\to\ell)}\right)$

We define the set of messages incoming at V_B at iteration ℓ as $U^{(\ell)} = \left\{U_1^{(\ell)}, U_2^{(\ell)}, ..., U_{d_v-1}^{(\ell)}\right\}$ where $U_i^{(\ell)}$ has the sample space S_M. The $U_i^{(\ell)}$'s are i.i.d. as the decoding neighbourhood is assumed to be tree-like. This is further shown in Fig. 12. In this figure we label the incoming messages to a variable node at iteration ℓ. Now each particular message is a consequence of the nodes in its corresponding sub-tree. This is due to the graph (code) being cycle-free, hence each decoding neighbourhood is tree-like. Note also that the independence between this set of messages does not depend on the iteration ℓ, i.e. the tree-like structure remains intact at each iteration.

Since $X^{(\ell)} = \Psi_v\left(U_1^{(\ell)}, U_2^{(\ell)}, ..., U_{d_v-1}^{(\ell)}\right)$, $X^{(\ell)}$ is conditionally independent of all other random variables given $U^{(\ell)}$. Alternatively, this holds true since $pa\left(X^{(\ell)}\right) = \left\{U_1^{(\ell)}, U_2^{(\ell)}, ..., U_{d_v-1}^{(\ell)}\right\}$. Therefore, we can compute $\mathbb{P}\left(X^{(1\to\ell)}\right)$ as follows:

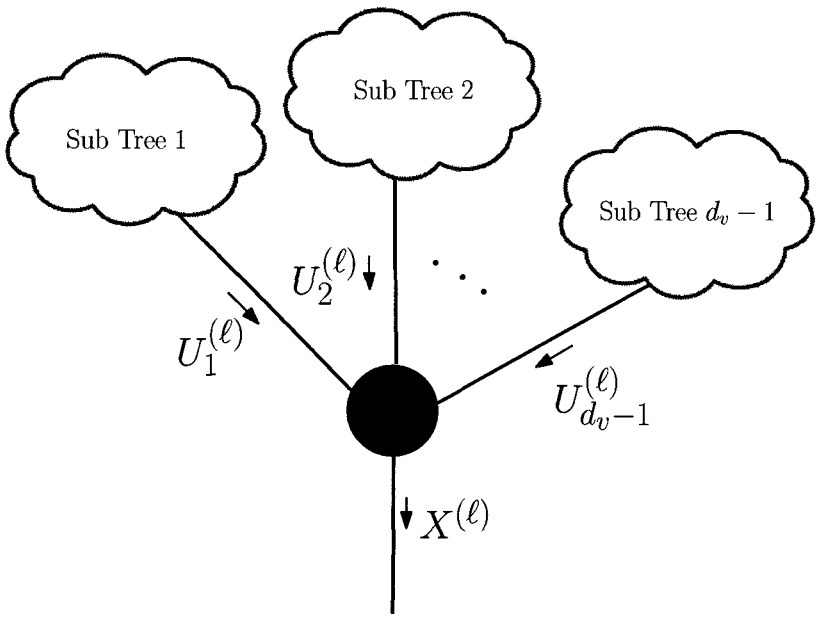


Figure 12: Messages incoming to a variable node and the corresponding sub-trees.

$$\begin{aligned}\mathbb{P}\left(X^{(1\to\ell)}\right) &= \sum_{u^{(1\to\ell)}} \mathbb{P}\left(X^{(1\to\ell)}|U^{(1\to\ell)}\right)\mathbb{P}\left(U^{(1\to\ell)}\right) \\ &= \sum_{u^{(1\to\ell)}} \prod_{i=1}^{\ell} \mathbb{P}\left(X^{(i)}|U^{(i)}\right) \prod_{j=1}^{d_v-1} \mathbb{P}\left(U_j^{(1\to\ell)}\right)\end{aligned} \tag{43}$$

Using Equation (43), for each $x \in S_X^{\ell}$ we need to sum over all $u \in S_M^{\ell(d_v-1)}$ which consists of $|S_M|^{\ell(d_v-1)}$ elements. Therefore, computing $\mathbb{P}\left(X^{(1\to\ell)}\right)$ requires $|S_X|^{\ell} \times |S_M|^{\ell(d_v-1)}$ additions and $|S_X|^{\ell} \times |S_M|^{\ell(d_v-1)} \times (\ell-1)(d_v-2)$ multiplications. The amount of memory required is $\left(|S_X| \times |S_M|^{d_v-1} + |S_M|^{\ell}\right) \times 8$ bytes to store the input tables corresponding to $\mathbb{P}\left(X^{(i)}|U^{(i)}\right)$ and $\mathbb{P}\left(U_j^{(1\to\ell)}\right)$, as well as $|S_X|^{\ell} \times 8$ bytes to store the output table corresponding to $\mathbb{P}\left(X^{(1\to\ell)}\right)$. This is very computationally intensive and shows another example of how the computation/memory complexity grows exponentially with increasing number of iterations.

Let us define a function $\vec{\Psi}_V : S_M^{\ell(d_v-1)} \rightarrow S_X^{\ell}$ such that
$\vec{\Psi}_V\left(U^{(1\rightarrow\ell)}\right) = \left\{\Psi_V\left(U^{(1)}\right), \Psi_V\left(U^{(2)}\right), ..., \Psi_V\left(U^{(\ell)}\right)\right\}$.

Since $X^{(1\rightarrow\ell)}$ is a deterministic function of $U^{(1\rightarrow\ell)}$, we have the following:

$$P\left(X^{(1\rightarrow\ell)} = x \,|\, U^{(1\rightarrow\ell)} = u\right), \quad x \in S_X^{\ell}, u \in S_M^{\ell(d_v-1)}$$
$$= \begin{cases} 1, & \text{if } x = \vec{\Psi}_V(u) \\ 0, & \text{otherwise} \end{cases} \tag{44}$$

Let us define $\equiv$ to be an equivalence on $S_M^{\ell(d_v-1)}$ so that for $a, b \in S_M^{\ell(d_v-1)}$, we write $a \equiv b$ to mean $\vec{\Psi}_V(a) = \vec{\Psi}_V(b)$. Thus we have the equivalence class $R_x \subset S_M^{\ell(d_v-1)}$, where $R_x = \left\{u \in S_M^{\ell(d_v-1)} | \vec{\Psi}_V(u) = x, x \in S_X^{\ell}\right\}$. Note that for $a, b \in S_X^{\ell}$, $R_a \cap R_b = \emptyset$ and $\cup_{x \in S_X^{\ell}} R_x = S_M^{\ell(d_v-1)}$. Since $r \in R_x \subset S_M^{\ell(d_v-1)}$, it can be represented by the set $\{r_1, r_2, ..., r_{d_v-1}\}, r_i \in S_M^{\ell}$. We can then re-write Equation (43) as follows:

$$P\left(X^{(1\rightarrow\ell)} = x\right) = \sum_{r \in R_x} \prod_{j=1}^{d_v-1} P(r_j) \tag{45}$$

Further simplifications can be made by observing that there may be a collection of distinct elements in R_x which have the same probability of occurrence. For this we define $J_x \subset R_x$ to be the maximal subset of R_x such that for any $a, b \in J_x$, $P(a) \neq P(b)$. Next we define a function $A_x : J_x \rightarrow \mathbb{N}$ such that $A_x(p), p \in J_x, x \in S_X^{\ell}$, returns the number of distinct elements (including p) in R_x which have equal probability. Similarly as before, we may represent each p by the set $\{p_1, p_2, ..., p_{d_v-1}\}, p_i \in S_M^{\ell}$. With these notions in place, we can now re-write Equation (45) to simplify it even further:

$$P\left(X^{(1\rightarrow\ell)} = x\right) = \sum_{p \in J_x} A_x(p) \prod_{j=1}^{d_v-1} P(p_j) \tag{46}$$

This leads us into the question of how to find R_x and J_x for each $x \in S_X^{\ell}$.

Computing R_x

1: $S \leftarrow 0$

2: **for** $i := 0$ **to** $\ell(d_v - 1) - 1$ **do**

3: $\quad n \leftarrow i \bmod \ell$

4: $\quad S_1 \leftarrow S + |S_X|^n$

5: $\quad$ **for** $j := 2$ **to** $|S_M|$ **do**

6: $\quad\quad S \leftarrow [S \ S_1]$ {Concatenate the vector S with S_1}

7: $\quad\quad S_1 \leftarrow S_1 + |S_X|^n$

8: $\quad$ **end for**

9: **end for**

The output of this algorithm is S which is an array of length $S_M^{\ell(d_v-1)}$. Next we define an index as follows:

$$I_x = \{a \mid S[a] = \mathcal{N}(x), x \in S_X^{\ell}\}. \tag{47}$$

Note that $\mathcal{N}$ is a function that assigns a unique integer to each element in a set. More specifically, we are defining an ordered index set corresponding to the elements of an alphabet space. Therefore, in our algorithms, we can refer to a particular element in the alphabet space by a unique integer instead of a vector. For a complete discussion on representing elements through integers, refer to Appendix A.

We then have that $R_x = S_M^{\ell(d_v-1)}(I_x)$. We can show this process through a simple example:

Example 4. *Assume we are working under DD-BMP decoding with $d_v = 3$ at iteration $\ell = 2$. Therefore, $S_M = \{-1, 1\}$ and $S_X = \{-2, 0, 2\}$. After running our above algorithm with these parameters, we receive the following output: $S = [0\ 1\ 3\ 4\ 1\ 2\ 4\ 5\ 3\ 4\ 6\ 7\ 4\ 5\ 7\ 8]$. We now wish to find R_x. For this example let $x = \{-2, 0\}$, i.e. $X^{(1)} = -2$ and $X^{(2)} = 0$. Then $\vec{\mathcal{N}}(x) = 3$. Therefore, we search*

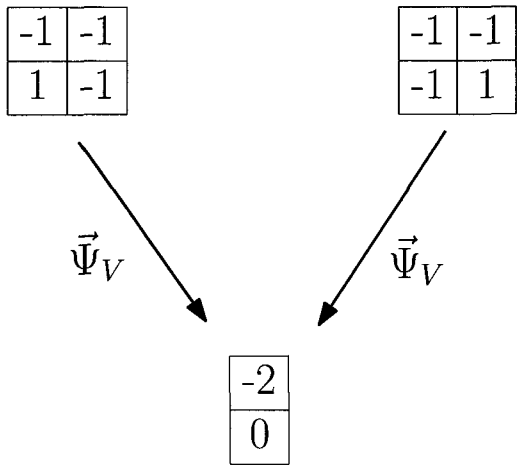


Figure 13: Mapping of elements from S_M^4 to elements of S_X^2 through the function $\vec{\Psi}_V$.

the vector S and return all the indices corresponding to 3. In this case, $I_x = \{2, 8\}$. Now $S_M^4(2)$ and $S_M^4(8)$, which make up the set R_x, are elements of S_M^4, thus can be represented by a 2×2 $(\ell \times (d_v - 1))$ table as shown in Fig. 13. Note that applying the function $\vec{\Psi}_V$ to each of the elements results in the desired x.

Computing J_x

Let $\boldsymbol{v} = \left[1 \ |S_M|^{\ell} \ |S_M|^{2\ell} \ldots |S_M|^{(d_v-2)\ell} \right]$. We define a matrix $\boldsymbol{P_\pi}$ of size $(d_v - 1) \times (d_v - 1)!$ where each column is a particular permutation of $\boldsymbol{v}^T$.

We now show how to compute J_x given R_x as well as assigning an output for each element in J_x as defined by the function A_x.

1: $J_x \leftarrow \emptyset$
2: **while** $R_x \neq \emptyset$ **do**
3: $r \leftarrow$ Choose any element from R_x
4: $J_x \leftarrow J_x \cup r$
5: $r' \leftarrow [\mathcal{N}(r_1) \ \mathcal{N}(r_2) \ldots \mathcal{N}(r_{d_v-1})]$

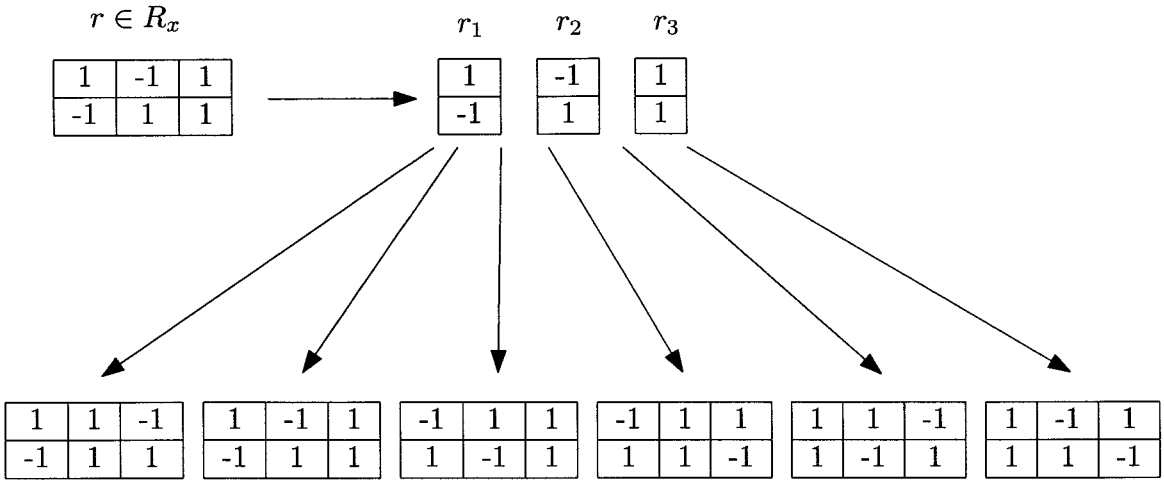


Figure 14: Unique elements in R_x mapping to the same $x \in S_X^2$ and having the same probability.

6: $e \leftarrow r^{'}\boldsymbol{P}_{\pi}$

7: $n \leftarrow$ # of unique elements in e

8: $A_x(r) \leftarrow n$

9: $R_x \leftarrow R_x \backslash R_x(e)$

10: **end while**

The above algorithm can be explained as follows with the help of Fig. 14. Assume $d_v = 4$, $\ell = 2$, and $x = \{1, 1\}$ under DD-BMP decoding. We choose some $r \in R_x$, place it in the set J_x, and split up their respective columns as shown by r_1, r_2, and r_3. Now each column is independent of each other, therefore, permuting the columns does not effect the probability of the new set of r's created. Also, it does not effect the outcome of x since addition is commutative (Ψ_V is simply the addition function). Therefore, we are only interested in the fact that 6 different copies in R_x map to the same x and have the same probability. We remove these copies from R_x and repeat the process until R_x is empty.

Example 5. *In this example we will show the reduction in calculation complexity between equations (43), (45), and (46). We will set the parameters as follows:* $\ell =$

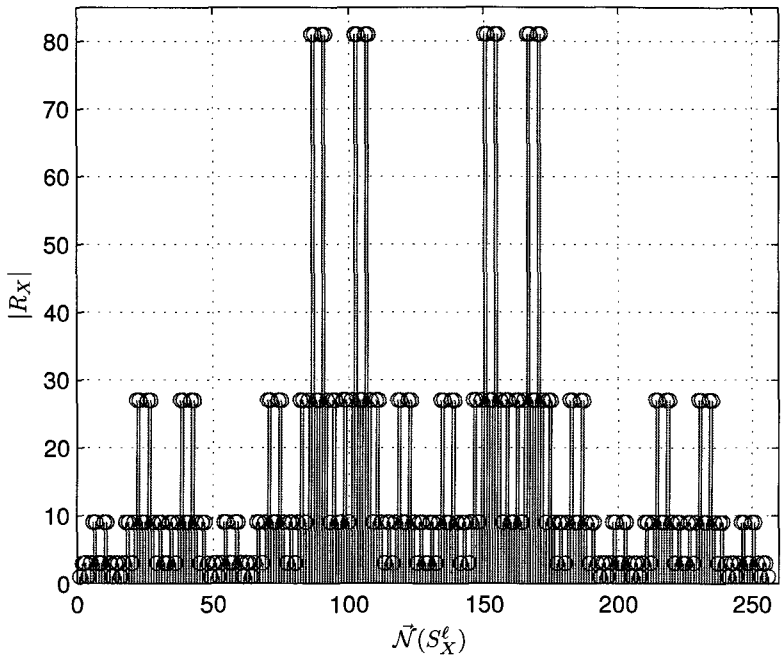


Figure 15: Number of elements contained in R_x for each $x \in S_X^\ell$

4, $d_v = 4$, *and* $S_M = \{-1, 1\}$. *Since* Ψ_V *is simply the addition function,* $S_X = \{-3, -1, 1, 3\}$. *Using these parameters, calculating* $\mathbb{P}\left(X^{(1\to\ell)}\right)$ *using Equation (43) requires* $4^4 \times 2^{4(4-1)} = 1,048,576$ *summations where each summation corresponds to* $\ell - 1 + d_v - 2 + 1 = 6$ *multiplications for a total of* $6,291,456$ *multiplications. Using Equation (45) we have a total of* $\sum_{x \in S_X^\ell} |R_x| = |S_M|^{\ell(d_v-1)} = 4096$ *summations with each summation corresponding to* $d_v - 2 = 2$ *multiplications for a total of 8,192 multiplications. Lastly, Equation (46) requires* $\sum_{x \in S_X^\ell} |J_x| = 816$ *summations and a total of* $816 \times (d_v - 1) = 2,448$ *multiplications. Figures 15 and 16 show the number of summations required for each* $x \in S_X^\ell$ *using Equations (45) and (46) respectively. The reduction in complexity here is apparent.*

In terms of memory requirements, the number of elements required to store all of the R_x's is given as $\sum_{x \in S_X^\ell} |R_x| = |S_M|^{\ell(d_v-1)}$. The number of elements needed to

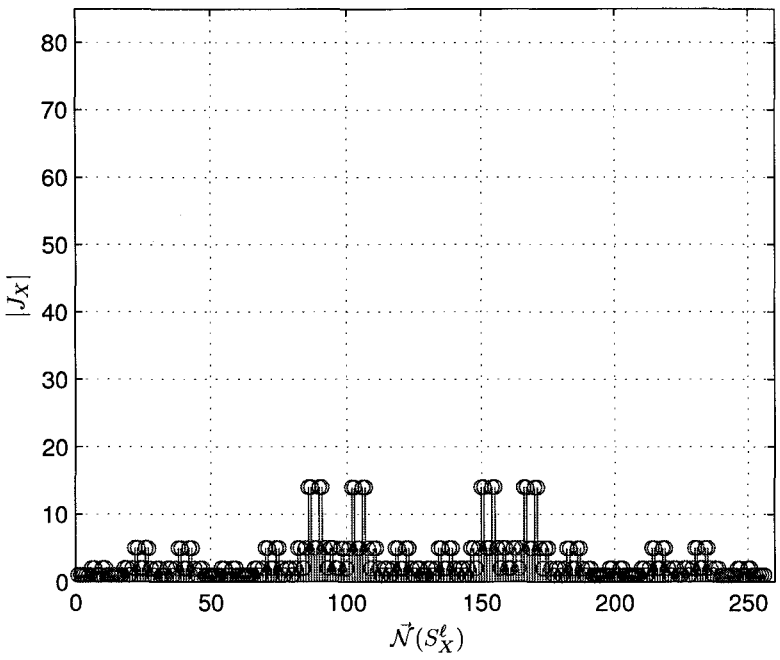


Figure 16: Number of elements contained in J_x for each $x \in S_X^\ell$

store all of the J_x's is $\sum_{x \in S_X^\ell} |J_x| = \begin{pmatrix} |S_M|^\ell + d_v - 2 \\ \\ d_v - 1 \end{pmatrix}$. It can be argued that the reduction in complexity gained by using Equation (46) is counterbalanced by the fact that we must compute R_x and J_x. However, these sets only need to be computed once for a particular ℓ, d_v, and S_M. Hence, they do not depend on the channel SNR or the check node degree d_c. Once the code parameters are set, varying the channel parameters (as is done to find the optimal threshold) does not introduce more computations. However, for each new iteration ℓ, R_x and J_x have to be re-computed from scratch. The computational complexity will grow exponentially with ℓ and quickly becomes infeasible. However, as we will see in Chapter 5, once memory truncation is introduced, we only need to compute R_x and J_x up to a particular iteration n, and may be reused for every iteration $\ell > n$.

In order to perform any of the above analysis, we need the distribution of $U_i^{(1\to\ell)}$, $i \in 1, ..., d_v - 1$. Since these are i.i.d., we do not need to differentiate between the varying subscripts.

4.4.3 Calculating $\mathbb{P}\left(U_i^{(1\to\ell)}\right)$

Define the set of messages incoming to any one of the check nodes in $\{C_i : i = 1, ..., d_v - 1\}$ at iteration ℓ as $T^{(\ell-1)} = \left\{T_1^{(\ell-1)}, T_2^{(\ell-1)}, ..., T_{d_c-1}^{(\ell-1)}\right\}$ (We ignore the subscript i since these set of messages have the same distribution regardless of which check node we choose to analyze). Once again, because the decoding neighbourhood is assumed to be tree-like, the $T_i^{(\ell-1)}$'s are i.i.d. Since $U_i^{(\ell)} = \Psi_c\left(T_1^{(\ell-1)}, T_2^{(\ell-1)}, ..., T_{d_c-1}^{(\ell-1)}\right)$, $U_i^{(\ell)}$ is conditionally independent of all other random variables given $T^{(\ell-1)}$. Hence we have the following:

$$\begin{aligned}\mathbb{P}\left(U_i^{(1\to\ell)}\right) &= \sum_{t^{(0\to\ell-1)}} \mathbb{P}\left(U_i^{(1\to\ell)}|T^{(0\to\ell-1)}\right)\mathbb{P}\left(T^{(0\to\ell-1)}\right) \\ &= \sum_{t^{(0\to\ell-1)}} \prod_{j=1}^{\ell} \mathbb{P}\left(U_i^{(j)}|T^{(j-1)}\right) \prod_{j=1}^{d_c-1} \mathbb{P}\left(T_j^{(0\to\ell-1)}\right)\end{aligned} \tag{48}$$

Using Equation (48), for each element in the sample space of $U_i^{(1\to\ell)}$, we need to sum over all the elements in the sample space of $T^{(0\to\ell-1)}$ which consists of $|S_M|^{\ell(d_c-1)}$ elements. Therefore, computing $\mathbb{P}\left(U_i^{(1\to\ell)}\right)$ requires a total of $|S_M|^{\ell d_c}$ additions and $|S_M|^{\ell d_c} \times (\ell - 1)(d_c - 2)$ multiplications. This is very computationally intensive and shows another example of how the computation complexity grows exponentially with increasing number of iterations.

Let us define a function $\vec{\Psi_C} : S_M^{\ell(d_c-1)} \to S_M^{\ell}$ such that
$\vec{\Psi_C}\left(T^{(0\to\ell-1)}\right) = \left\{\Psi_C\left(T^{(0)}\right), \Psi_C\left(T^{(1)}\right), ..., \Psi_C\left(T^{(\ell-1)}\right)\right\}$.

Since $U_i^{(1\to\ell)}$ is a deterministic function of $T^{(0\to\ell-1)}$, we have the following:

$$P\Big(U_i^{(1\to\ell)} = u \,|\, T^{(0\to\ell-1)} = t\Big), \quad t \in S_M^{\ell(d_c-1)}, u \in S_M^{\ell}$$
$$= \begin{cases} 1, & \text{if } u = \vec{\Psi}_C(t) \\ 0, & \text{otherwise} \end{cases} \tag{49}$$

We can then re-write Equation (48) as follows:

$$P\Big(U_i^{(1\to\ell)} = u\Big) = \sum_{t|u=\vec{\Psi}_C(t)} \prod_{j=1}^{d_c-1} P(t_j), \{t_1, ..., t_{d_c-1}\} \in S_M^{\ell}. \tag{50}$$

Using Equation (50), calculating $\mathbb{P}\left(U_i^{(1\to\ell)}\right)$ requires $|S_M|^{\ell(d_c-1)}$ additions and $|S_M|^{\ell(d_c-1)} \times (d_c - 2)$ multiplications. We wish to reduce the computation complexity even further similar to the methods in section 4.4.2.

Define J_u for $u \in S_M^{\ell}$ to be the maximal subset of $S_M^{\ell(d_c-1)}$ such that for any $a, b \in J_u$, $\vec{\Psi_C}(a) = \vec{\Psi_C}(b) = u$ and $P(a) \neq P(b)$. Next we define a function $A_u : J_u \to \mathbb{N}$ such that $A_u(t)$, $t \in S_M^{\ell(d_c-1)}$ returns the number of distinct elements (including t) in $S_M^{\ell(d_c-1)}$ which have equal probability. Hence, Equation (50) reduces to the following:

$$P\Big(U_i^{(1\to\ell)} = u\Big) = \sum_{t\in J_u} A_u(t) \prod_{j=1}^{d_c-1} P(t_j). \tag{51}$$

The number of additions required to calculate $\mathbb{P}\left(U_i^{(1\to\ell)}\right)$ using Equation (51) can be found by the following:

$$\begin{aligned}
&\sum_{u \in S_M^\ell} |J_u| \\
&= \sum_{i=1}^{|S_M|^\ell} \left(\sum_{i_1=i}^{|S_M|^\ell} \left(\sum_{i_2=i_1}^{|S_M|^\ell} \left(\cdots \sum_{i_{d_c-2}=i_{d_c-3}}^{|S_M|^\ell} \left(\sum_{i_{d_c-1}=i_{d_c-2}}^{|S_M|^\ell} 1 \right) \cdots \right) \right) \right) \\
&= \begin{pmatrix} |S_M|^\ell + d_c - 2 \\ d_c - 1 \end{pmatrix}.
\end{aligned} \tag{52}$$

Equation (52) is derived by asking how many different ways there are to choose $d_c - 1$ objects from $|S_M|^\ell$ elements (with replacement) such that no two selections of $d_c - 1$ objects can be made equal by permutation. This is because the 'min' and product function used in check nodes are associative.

Using Equation (52), the number of multiplications required is $\begin{pmatrix} |S_M|^\ell + d_c - 2 \\ d_c - 1 \end{pmatrix} \times (d_c - 1)$.

The questions is, how does this compare with the complexity in (50). Let θ represent a multiplicative factor which gives a ratio between the number of additions required in (50) and (51). For this, we let $\mu = d_c - 1$ and $\eta = |S_M|$.

$$
\begin{aligned}
\theta &= \frac{\eta^{\ell\mu}}{\begin{pmatrix} \eta^{\ell}+\mu-1 \\ \\ \mu \end{pmatrix}} \\
&= \frac{\eta^{\ell\mu}(\eta^{\ell}-1)!\mu!}{(\eta^{\ell}+\mu-1)!} \\
&= \frac{\eta^{\ell\mu}\mu!}{(\eta^{\ell}+\mu-1)(\eta^{\ell}+\mu-2)...\eta^{\ell}} \\
&= \frac{\eta^{\ell(\mu-1)}\mu!}{(\eta^{\ell}+\mu-1)(\eta^{\ell}+\mu-2)...(\eta^{\ell}+1)} \\
&\approx \frac{\eta^{\ell(\mu-1)}\mu!}{\eta^{\ell(\mu-1)}} \ (\text{for } \eta^{\ell} >> \mu) \\
&= \mu! = (d_c-1)!
\end{aligned}
\tag{53}
$$

For example, if $d_c = 8$, after a certain number of iterations ℓ so that $|S_M|^{\ell} >> 7$, using Equation (50) will require about $7! = 5040$ times more additions compared with using Equation (51). Using a similar approach as in Equation (53), the multiplicative factor giving the ratio between the number of multiplications required in (50) and (51) is approximately $(d_c-2)^2(d_c-3)!$.

In order to compute the analysis described in this section, we need the distribution of $T_i^{(0\to\ell-1)}$, $i \in 1, ..., d_c-1$. Since these are i.i.d., it is enough to find $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$ as described in the following section (we do not need to differentiate between the varying subscripts).

4.4.4 Calculating $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$

Let us define the random vector $B^{(0\to\ell-1)} = \left\{B^{(0)}, B^{(1)}, ..., B^{(\ell-1)}\right\}$. We can now show how to calculate the joint probability mass function of $T_1^{(0\to\ell-1)}$.

$$\begin{aligned}\mathbb{P}\left(T_0^{(1\to\ell-1)}\right) &= \sum_{b^{(0\to\ell-1)}} \mathbb{P}\left(T_1^{(0\to\ell-1)}|B^{(0\to\ell-1)}\right)\mathbb{P}\left(B^{(0\to\ell-1)}\right) \\ &= \prod_{i=0}^{\ell-1}\left(\sum_{b^{(i)}} \mathbb{P}\left(T_1^{(i)}|B^{(i)}\right)\right)\mathbb{P}\left(B^{(0\to\ell-1)}\right)\end{aligned} \tag{54}$$

In Equation (54), we have made use of the fact that in a DAG, a random variable is conditionally independent of all other random variables given its parents if it is a deterministic function of its parents. We are now faced with the task of finding $\mathbb{P}\left(B^{(0\to\ell-1)}\right)$.

$$\begin{aligned}&\mathbb{P}\left(B^{(0\to\ell-1)}\right) \\ &= \mathbb{P}\left(B^{(\ell-1)}|B^{(0\to\ell-2)}\right)\mathbb{P}\left(B^{(0\to\ell-2)}\right) \\ &= \sum_{x^{(\ell-1)}} \mathbb{P}\left(B^{(\ell-1)}|B^{(\ell-2)}, X^{(\ell-1)}\right)\mathbb{P}\left(B^{(0\to\ell-2)}, X^{(\ell-1)}\right) \\ &= \sum_{x^{(\ell-1)}} \mathbb{P}\left(B^{(\ell-1)}|B^{(\ell-2)}, X^{(\ell-1)}\right) \sum_{x^{(\ell-2)}} \mathbb{P}\left(B^{(\ell-2)}|B^{(\ell-3)}, X^{(\ell-2)}\right)\mathbb{P}\left(B^{(0\to\ell-3)}, X^{(\ell-1)}, X^{(\ell-2)}\right) \\ &\cdot \\ &\cdot \\ &\cdot \\ &= \prod_{i=1}^{\ell-1}\sum_{x^{(i)}} \mathbb{P}\left(B^{(i)}|B^{(i-1)}, X^{(i)}\right)\mathbb{P}\left(B^{(0)}, X^{(1\to\ell-1)}\right) \\ &= \mathbb{P}\left(B^{(0)}\right)\prod_{i=1}^{\ell-1}\sum_{x^{(i)}} \mathbb{P}\left(B^{(i)}|B^{(i-1)}, X^{(i)}\right)\mathbb{P}\left(X^{(1\to\ell-1)}\right)\end{aligned} \tag{55}$$

Combining Equation (55) with Equation (54), we have the following:

$$\mathbb{P}\left(T_0^{(0\to\ell-1)}\right)$$
$$= \sum_{b^{(0)}} \mathbb{P}\left(T_1^{(0)}|B^{(0)}\right)\mathbb{P}\left(B^{(0)}\right) \prod_{i=1}^{\ell-1} \left(\sum_{b^{(i)}} \mathbb{P}\left(T_1^{(i)}|B^{(i)}\right) \sum_{x^{(i)}} \mathbb{P}\left(B^{(i)}|B^{(i-1)}, X^{(i)}\right) \right) \mathbb{P}\left(X^{(1\to\ell-1)}\right) \tag{56}$$

We now have all the information we need to calculate $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$ since $\mathbb{P}\left(X^{(1\to\ell-1)}\right)$ is calculated in the previous iteration, $\mathbb{P}\left(B^{(i)}|B^{(i-1)}, X^{(i)}\right)$ is given in Equation (36), and similarly,

$$\begin{aligned} &\mathbb{P}\left(T_1^{(n)} = t|B^{(n)} = b\right), \quad t \in S_M, b \in S_B \\ &= \begin{cases} 1, & if \ t = \Psi_M(b) \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{57}$$

This concludes our steps for calculating $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$. It is important to note that we use the distributions required to find $\mathbb{P}\left(B^{(\ell)}\right)$ in order to calculate distributions required to calculate $\mathbb{P}\left(B^{(\ell+1)}\right)$. This is shown more explicitly in Fig. 17.

4.5 Calculating $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$ for $\gamma \neq 0$

When $\gamma \neq 0$, M_0 is now included in each variable node operation. Without loss of generality in terms of analysis, we can say that $\gamma B^{(0)}$ is included in each variable node operation. Hence, our statements of conditional independence change resulting in a different way of calculating $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$. We show the modified graphical model in Fig. 18 which similar to Fig. 7 except that now each $X^{(i)}$ is updated with the originally received message used to initialize the memory content $B^{(0)}$.

Through the use of Bayes' Ball Algorithm, we can make the following claims:

$$\begin{aligned} B^{(0)} &\perp\!\!\!\perp X^{(1)}|B^{(0)} \\ B^{(k)} &\perp\!\!\!\perp X^{(k+1)}|\left\{X^{(1\to k)}, B^{(0)}\right\}, \ k \geq 1. \end{aligned} \tag{58}$$

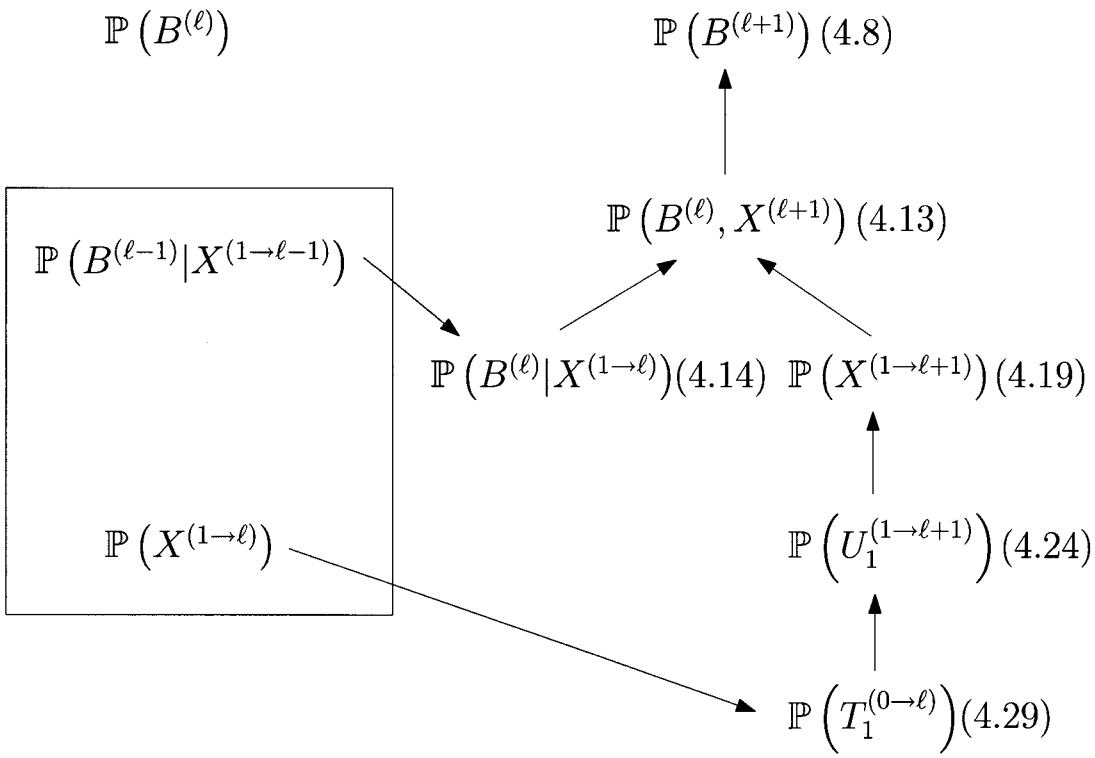


Figure 17: Using distributions calculated at iteration ℓ to help calculate distributions at iteration $\ell + 1$ for $\gamma = 0$.

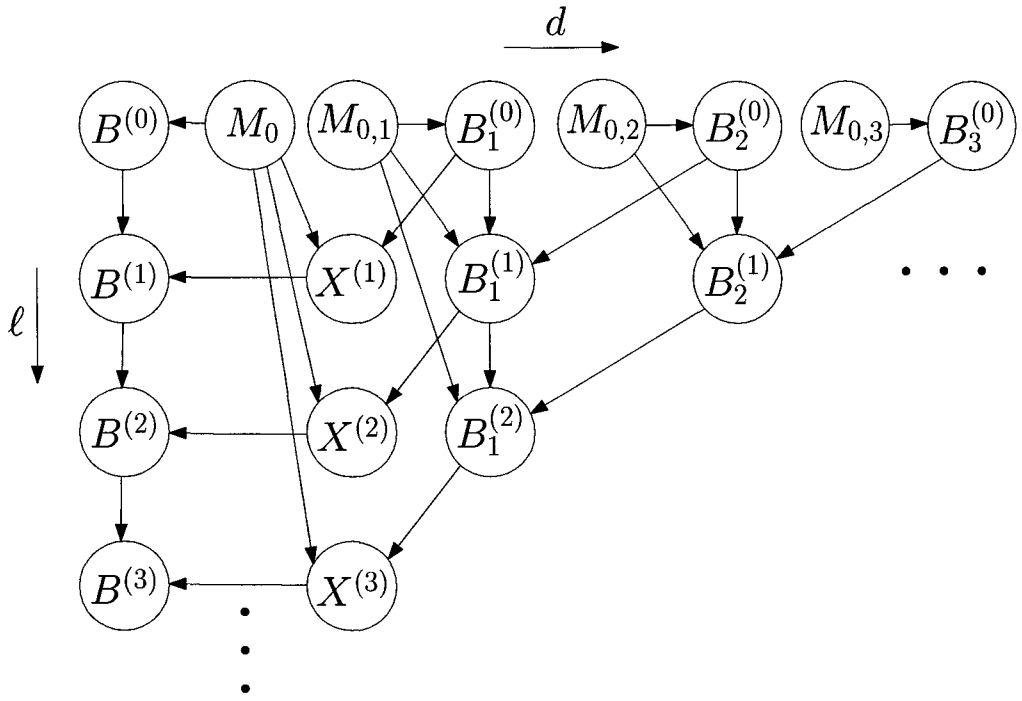


Figure 18: Bayesian Network showing all the random variables involved in calculating the probability mass function of $B^{(3)}$ with $\gamma \neq 0$

Using the claims in (58) we have the following:

$$\begin{aligned}
&\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right) \\
&= \sum_{x^{(1\to\ell-1)}} \sum_{b^{(0)}} \mathbb{P}\left(B^{(\ell-1)} | X^{(1\to\ell-1)}, B^{(0)}\right) \mathbb{P}\left(X^{(\ell)} | X^{(1\to\ell-1)}, B^{(0)}\right) \mathbb{P}\left(X^{(1\to\ell-1)}, B^{(0)}\right) \\
&= \sum_{x^{(1\to\ell-1)}} \sum_{b^{(0)}} \mathbb{P}\left(B^{(\ell-1)} | X^{(1\to\ell-1)}, B^{(0)}\right) \mathbb{P}\left(X^{(1\to\ell)}, B^{(0)}\right).
\end{aligned} \tag{59}$$

In comparison to Section 4.4, the size of each probability mass function table grows by a factor of $|S_B|$ with the introduction of $B^{(0)}$. Similarly, the computational complexity grows by a factor of $|S_B|$, now being bounded by $O\left(|S_X|^{\ell-1} \times |S_B|^2\right)$.

4.5.1 Calculating $\mathbb{P}\left(B^{(\ell-1)}|X^{(1\to\ell-1)}, B^{(0)}\right)$

$$\begin{aligned}
&\mathbb{P}\left(B^{(\ell-1)}|X^{(1\to\ell-1)}, B^{(0)}\right) \\
&= \sum_{b^{(\ell-2)}} \mathbb{P}\left(B^{(\ell-1)}|X^{(1\to\ell-1)}, B^{(\ell-2)}, B^{(0)}\right)\mathbb{P}\left(B^{(\ell-2)}|X^{(1\to\ell-1)}, B^{(0)}\right) \\
&= \sum_{b^{(\ell-2)}} \mathbb{P}\left(B^{(\ell-1)}|X^{(\ell-1)}, B^{(\ell-2)}\right)\mathbb{P}\left(B^{(\ell-2)}|X^{(1\to\ell-2)}, B^{(0)}\right),
\end{aligned} \tag{60}$$

since $B^{(\ell-2)}$ is conditionally independent of $X^{(\ell-1)}$ given $X^{(1\to\ell-2)}$ and $B^{(0)}$. The conditional $\mathbb{P}\left(B^{(\ell-2)}|X^{(1\to\ell-2)}, B^{(0)}\right)$ is obtained from the previous iteration.

4.5.2 Calculating $\mathbb{P}\left(X^{(1\to\ell)}, B^{(0)}\right)$

For this section, it is easier to perform the analysis when the variable node process is split into two parts. This is shown in Fig. 19. We first process the incoming messages through the function $\tilde{\Psi}_V$ which is equivalent to Ψ_V for the case where $\gamma = 0$. We call this output $Q^{(\ell)}$ and then add $\gamma B^{(0)}$ to produce the final output $X^{(\ell)}$.

Let $Q^{(1\to\ell)} = \left\{Q^{(1)}, Q^{(2)}, ..., Q^{(\ell)}\right\}$.

Then $X^{(1\to\ell)} = \left\{Q^{(1)} + \gamma B^{(0)}, Q^{(2)} + \gamma B^{(0)}, ..., Q^{(\ell)} + \gamma B^{(0)}\right\}$.

If we have $\mathbb{P}\left(Q^{(1\to\ell)}\right)$ then we can calculate $\mathbb{P}\left(X^{(1\to\ell)}, B^{(0)}\right)$ as follows:

$$\begin{aligned}
\mathbb{P}\left(X^{(1\to\ell)}, B^{(0)}\right) &= \mathbb{P}\left(X^{(1\to\ell)}|B^{(0)}\right)\mathbb{P}\left(B^{(0)}\right) \\
&= \mathbb{P}\left(B^{(0)}\right) \sum_{q^{(1\to\ell)}} \mathbb{P}\left(X^{(1\to\ell)}|Q^{(1\to\ell)}, B^{(0)}\right)\mathbb{P}\left(Q^{(1\to\ell)}\right) \\
&= \mathbb{P}\left(B^{(0)}\right) \prod_{i=1}^{\ell} \left(\sum_{q^{(i)}} \mathbb{P}\left(X^{(i)}|Q^{(i)}, B^{(0)}\right) \right) \mathbb{P}\left(Q^{(1\to\ell)}\right)
\end{aligned} \tag{61}$$

Since $X^{(i)}$ is a deterministic function of $Q^{(i)}$ and $B^{(0)}$, we have the following:

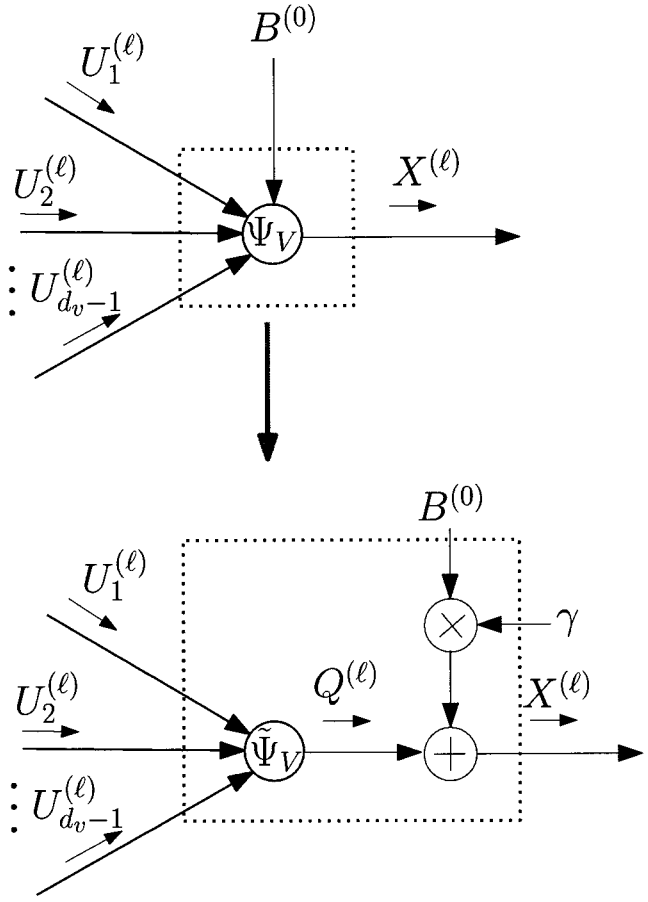


Figure 19: Variable Node operation with $\gamma \neq 0$

$$\mathbb{P}\left(X^{(i)} = x | Q^{(i)} = q, B^{(0)} = b\right), \quad x \in S_X, q \in S_Q, b \in S_B$$

$$= \begin{cases} 1, & \text{if } x = q + \gamma b \\ 0, & \text{otherwise} \end{cases} \tag{62}$$

$\mathbb{P}\left(Q^{(1\to\ell)}\right)$ can be found using the exact same methods as finding $\mathbb{P}\left(X^{(1\to\ell)}\right)$ in section 4.4.2 assuming we have $\mathbb{P}\left(U_1^{(1\to\ell)}\right)$. $\mathbb{P}\left(U_1^{(1\to\ell)}\right)$ can be found using the exact same methods as described in section 4.4.3 assuming we have $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$. In comparison to the case where $\gamma = 0$, the added complexity lies in finding $\mathbb{P}\left(X^{(1\to\ell)}, B^{(0)}\right)$. With $\gamma \neq 0$, calculating $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$ differs from the methods given in section 4.4.4 as we will see in the following section.

4.5.3 Calculating $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$ for $\gamma \neq 0$

In Equation (55) we have used the fact that $B^{(0)}$ is independent of $X^{(1\to\ell-1)}$ in order to split up their joint probability. However, this cannot be done for the case of $\gamma \neq 0$. Therefore, we end up with the following expression for calculating $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$.

$$\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$$
$$= \sum_{b^{(0)}} \mathbb{P}\left(T_1^{(0)} | B^{(0)}\right) \left(\prod_{i=1}^{\ell-1} \sum_{b^{(i)}} \mathbb{P}\left(T_1^{(i)} | B^{(i)}\right) \sum_{x^{(i)}} \mathbb{P}\left(B^{(i)} | B^{(i-1)}, X^{(i)}\right) \right) \mathbb{P}\left(X^{(1\to\ell-1)}, B^{(0)}\right) \tag{63}$$

$\mathbb{P}\left(X^{(1\to\ell-1)}, B^{(0)}\right)$ is calculated in the previous iteration, hence we have all the information we need to calculate $\mathbb{P}\left(T_1^{(0\to\ell-1)}\right)$.

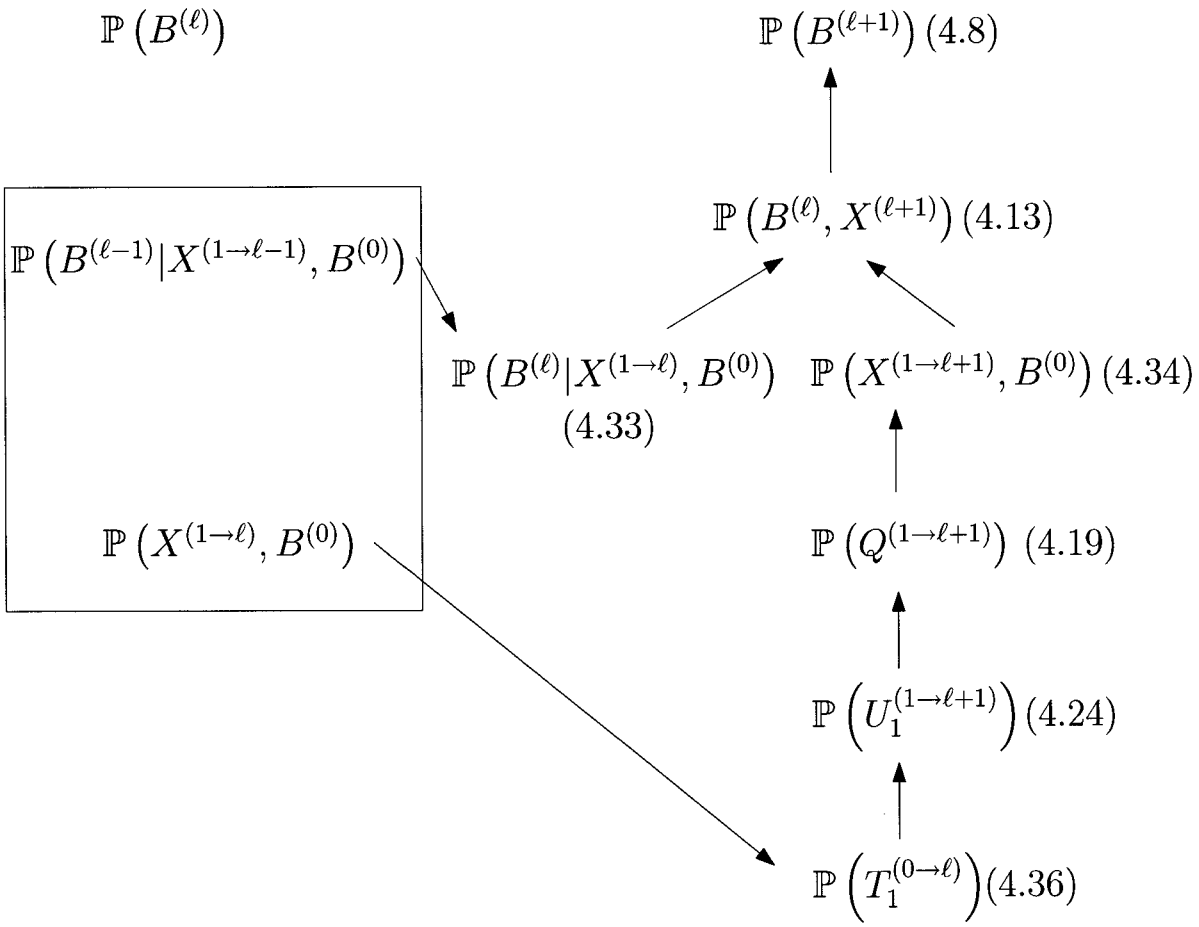


Figure 20: Using distributions calculated at iteration ℓ to help calculate distributions at iteration $\ell + 1$ for $\gamma \neq 0$.

Chapter 5

Memory Truncation

As we have seen in the analysis of DE, the computational complexity and the size of the number of elements in the tables needed to store the p.m.f.'s grows exponentially with increasing iteration ℓ. This is not tractable or feasible for implementation in terms of both time and memory, so it is natural to look for an approximation in order to halt the growing calculation complexity. We first propose how such an approximation can be found. Once a suitable approximation is found, we reformulate the equations for density evolution. In particular, our original Bayesian Network will be modified to represent the approximations, causing new conditional (in)dependence between sets of nodes. Lastly, we give some simulation results to justify the approximation. It is important to note that in our memory truncation analysis, we work with the iterative decoding model where $\gamma = 0$. A suitable approximation has not yet been found for the case where $\gamma \neq 0$.

5.1 Proposed Solution

Consider the calculation of the joint distribution $\mathbb{P}\left(B^{(\ell-1)}, X^{(\ell)}\right)$ as computed in section 4.4. This computation uses the fact that $B^{(\ell-1)} \perp\!\!\!\perp X^{(\ell)} | X^{(1\to\ell-1)}$. The problem lies in the fact that the conditioning set $X^{(1\to\ell-1)}$ grows exponentially in the size of

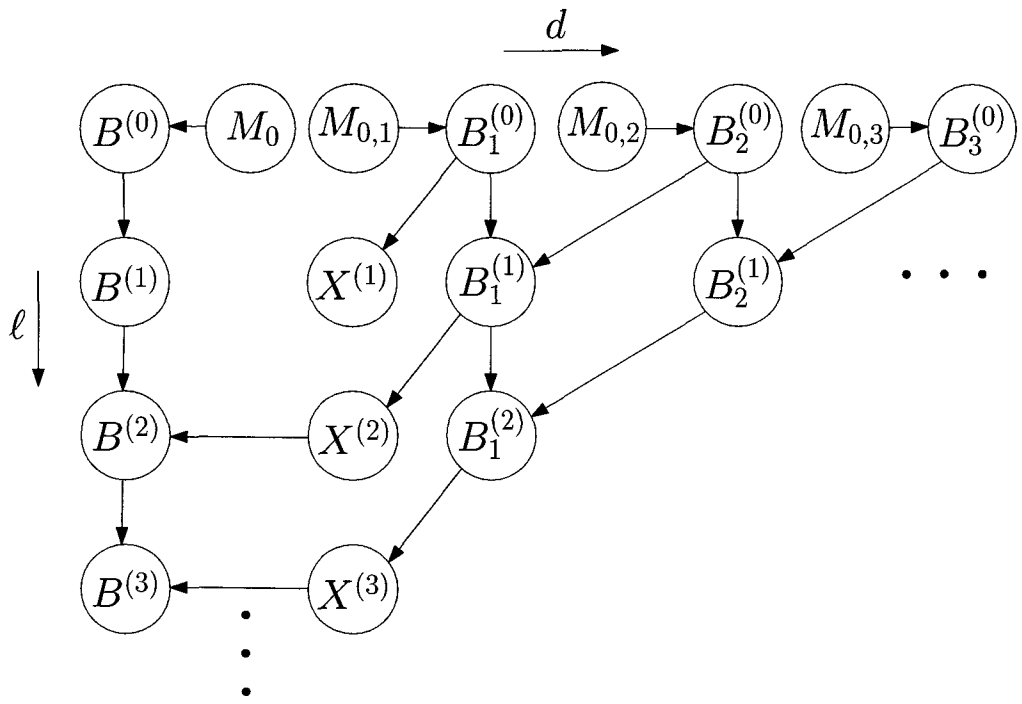


Figure 21: Removing links for Memory Truncation

the number of elements contained in its sample space. Now, consider we make the following approximation:

$$B^{(\ell-1)} \perp\!\!\!\perp X^{(\ell)} | X^{(\ell-n+1 \to \ell-1)}, \; n \geq 2. \tag{64}$$

Regardless of ℓ, the conditioning set in (64) will always have size $|S_X|^{n-1}$ in terms of the number of random variables it contains.

How does this approximation affect other nodes in the Bayesian network representation? We first show this through an example. Consider $\ell = 3$ and $n = 2$. Therefore we have $B^{(2)} \perp\!\!\!\perp X^{(3)} | X^{(2)}$. In order for this to hold true from a Bayesian network point of view, the link joining $X^{(1)}$ to $B^{(1)}$ must be removed. This is shown in Fig. 21.

A consequence of this new representation is that now $\mathbb{P}\left(B^{(3)} | B^{(2)}, B^{(1)}, B^{(0)}\right) = \mathbb{P}\left(B^{(3)} | B^{(2)}, B^{(1)}\right)$. This is obvious since every path from $B^{(0)}$ to $B^{(3)}$ must pass through the path $B^{(0)} \to B^{(1)} \to B^{(2)}$. Since this path forms

a Markov chain and $B^{(1)}$ is observed, we clearly have d-separation. We now formally give our approximation known as memory truncation:

$B^{(0\to\ell)}$ can be thought of as a random process. Clearly, it is not a Markov Process of some finite order. Thus, we truncate the memory in order to approximate $B^{(0\to\ell)}$ by a Markov Process of order n, $n \in \mathbb{N}$. This is represented by the following:

$$\mathbb{P}\left(B^{(n+k)}|B^{(n+k-1)},...,B^{(0)}\right) \approx \mathbb{P}\left(B^{(n+k)}|B^{(n+k-1)},...,B^{(k)}\right) \tag{65}$$

When making the above approximation in the analysis, we refer to it as *memory truncation of order n* ($\mathrm{MT}^{(n)}$). A consequence of this memory truncation is that now $B^{(\ell-1)}$ is conditionally independent of $X^{(\ell)}$ given $X^{(\ell-n+1\to\ell-1)}$. This approximation is necessary in order to practically implement density evolution for iterative decoding algorithms with memory. Without memory truncation we would need an infinite amount of time and memory to calculate the *exact* threshold with perfect precision. By using (65), we can halt the growing complexity of the problem by using only the previous n memory contents as the dependency. Therefore, the computation complexity remains constant for $\ell \geq n$. In other words, calculating $\mathbb{P}\left(B^{(\ell)}\right)$ for $\ell \geq n$ has the same complexity as computing $\mathbb{P}\left(B^{(n)}\right)$ exactly. We note that when using (65), the computation of $\mathbb{P}\left(B^{(\ell)}\right)$ is exact for $\ell \leq n$.

5.2 Mathematical Analysis of Memory Truncation

We now present the mathematical analysis for approximating $\mathbb{P}\left(B^{(\ell)}\right)$ using our proposed method of memory truncation.

For the remainder of the section, we assume that we are using a memory truncation of order n. Therefore, we have the exact computation for $\mathbb{P}\left(B^{(n)}\right)$ using the methods

described in chapter 4. After this computation, we have the following probability mass functions:

- $\mathbb{P}\left(B^{(i)}, X^{(i+1)}\right)$ for $i = 0, ..., n-1$
- $\mathbb{P}\left(B^{(n-1)}|X^{(1\to n-1)}\right)$
- $\mathbb{P}\left(X^{(1\to n)}\right)$
- $\mathbb{P}\left(U_1^{(1\to n)}\right)$
- $\mathbb{P}\left(T_1^{(0\to n-1)}\right)$

Following a similar approach to that used in section 4.3, we derive the probability mass function of $B^{(n+1)}$ which is the first memory content distribution to be approximated.

We start with Equation (37),

$$P\left(B^{(n+1)} = b\right) = \sum_{(b',x)|b=\Psi_B\left(b',x\right)} P\left(B^{(n)} = b', X^{(n+1)} = x\right),$$

but now calculating the joint distribution $\mathbb{P}\left(B^{(n)}, X^{(n+1)}\right)$ will be different.

5.2.1 Calculating $\mathbb{P}\left(B^{(n)}, X^{(n+1)}\right)$ for $\mathrm{MT}^{(n)}$

$$\begin{aligned}
&\mathbb{P}\left(B^{(n)}, X^{(n+1)}\right) \\
&= \sum_{x^{(2\to n)}} \mathbb{P}\left(B^{(n)}|X^{(2\to n)}\right)\mathbb{P}\left(X^{(n+1)}|X^{(2\to n)}\right)\mathbb{P}\left(X^{(2\to n)}\right) \\
&= \sum_{x^{(2\to n)}} \mathbb{P}\left(B^{(n)}|X^{(2\to n)}\right)\mathbb{P}\left(X^{(2\to n+1)}\right),
\end{aligned} \tag{66}$$

where we have used the fact that $B^{(n)} \perp\!\!\!\perp X^{(n+1)}|X^{(2\to n)}$.

5.2.2 Calculating $\mathbb{P}\left(B^{(n)}|X^{(2\to n)}\right)$

$$\mathbb{P}\left(B^{(n)}|X^{(2\to n)}\right) = \sum_{b^{(n-1)}} \mathbb{P}\left(B^{(n)}|X^{(n)}, B^{(n-1)}\right)\mathbb{P}\left(B^{(n-1)}|X^{(2\to n)}\right). \tag{67}$$

5.2.3 Calculating $\mathbb{P}\left(B^{(n-1)}|X^{(2\to n)}\right)$

$$\begin{aligned}\mathbb{P}\left(B^{(n-1)}|X^{(2\to n)}\right) &= \sum_{x^{(1)}} \mathbb{P}\left(B^{(n-1)}|X^{(1\to n)}\right)\mathbb{P}\left(X^{(1)}|X^{(2\to n)}\right) \\ &= \frac{1}{\mathbb{P}\left(X^{(2\to n)}\right)} \sum_{x^{(1)}} \mathbb{P}\left(B^{(n-1)}|X^{(1\to n-1)}\right)\mathbb{P}\left(X^{(1\to n)}\right),\end{aligned} \tag{68}$$

where $\mathbb{P}\left(B^{(n-1)}|X^{(1\to n-1)}\right)$ and $\mathbb{P}\left(X^{(1\to n)}\right)$ are obtained from the previous iteration and $\mathbb{P}\left(X^{(2\to n)}\right) = \sum_{x^{(1)}} \mathbb{P}\left(X^{(1\to n)}\right)$.

5.2.4 Calculating $\mathbb{P}\left(X^{(2\to n+1)}\right)$

Calculating $\mathbb{P}\left(X^{2\to n+1}\right)$ can be performed the exact same as in section 4.4.2 given that we have $\mathbb{P}\left(U_1^{(2\to n+1)}\right)$. Similarily, $\mathbb{P}\left(U_1^{(2\to n+1)}\right)$ can be found using the same methods as in section 4.4.3 given that we have $\mathbb{P}\left(T_1^{(1\to n)}\right)$. Now, because we have $\mathbb{P}\left(X^{(1\to n)}\right)$ from the previous iteration, we can simply calculate $\mathbb{P}\left(T_1^{(0\to n)}\right)$ as in section 4.4.4 and then marginalize over $T_1^{(0)}$ to obtain $\mathbb{P}\left(T_1^{(1\to n)}\right)$.

This approach to calculating $\mathbb{P}\left(B^{(n+1)}\right)$ is a special case since we are using exact p.m.f.'s that have been calculated in the previous iteration. We will now show the analysis for the more general case where the p.m.f.'s used from the previous iteration are now approximated versions of the actual p.m.f.'s.

5.3 Mathematical Analysis of Memory Truncation (General Case)

Once again, we are assuming that we are approximating with a memory truncation of order n. Consider that we have calculated (approximated) $\mathbb{P}\left(B^{(k)}\right)$, $k > n$. Fig. 22 shows the Bayesian network to help calculate $\mathbb{P}\left(B^{(k)}\right)$ under memory truncation of order n. Notice that we have removed the nodes which are not important to our discussion. For example, a connection between $X^{(i)}$ and $X^{(i+1)}$ is represented by a dotted line which is used to represent the fact that these two nodes share some dependency, however, they are not directly connected. Notice that in Fig. 22, the edges have been removed between all $X^{(i)}$ and $B^{(i)}$ for $i \leq k - n$. This is due to the memory truncation approximation. After computing $B^{(k)}$, we have the following probability mass functions:

- $\mathbb{P}\left(B^{(k-1)}, X^{(k)}\right)$
- $\mathbb{P}\left(B^{(k-1)} | X^{(k-n+1 \to k-1)}\right)$
- $\mathbb{P}\left(X^{(k-n+1 \to k)}\right)$
- $\mathbb{P}\left(U_1^{(k-n+1 \to k)}\right)$
- $\mathbb{P}\left(T_1^{(k-n \to k-1)}\right)$

We now derive the probability mass function of $B^{(k+1)}$, $k \geq n$.

Again we start with Equation (37),

$P\left(B^{(k+1)} = b\right) = \sum_{(b',x)|b=\Psi_B\left(b',x\right)} P\left(B^{(k)} = b', X^{(k+1)} = x\right)$, and then proceed to calculating the joint distribution $\mathbb{P}\left(B^{(k)}, X^{(k+1)}\right)$.

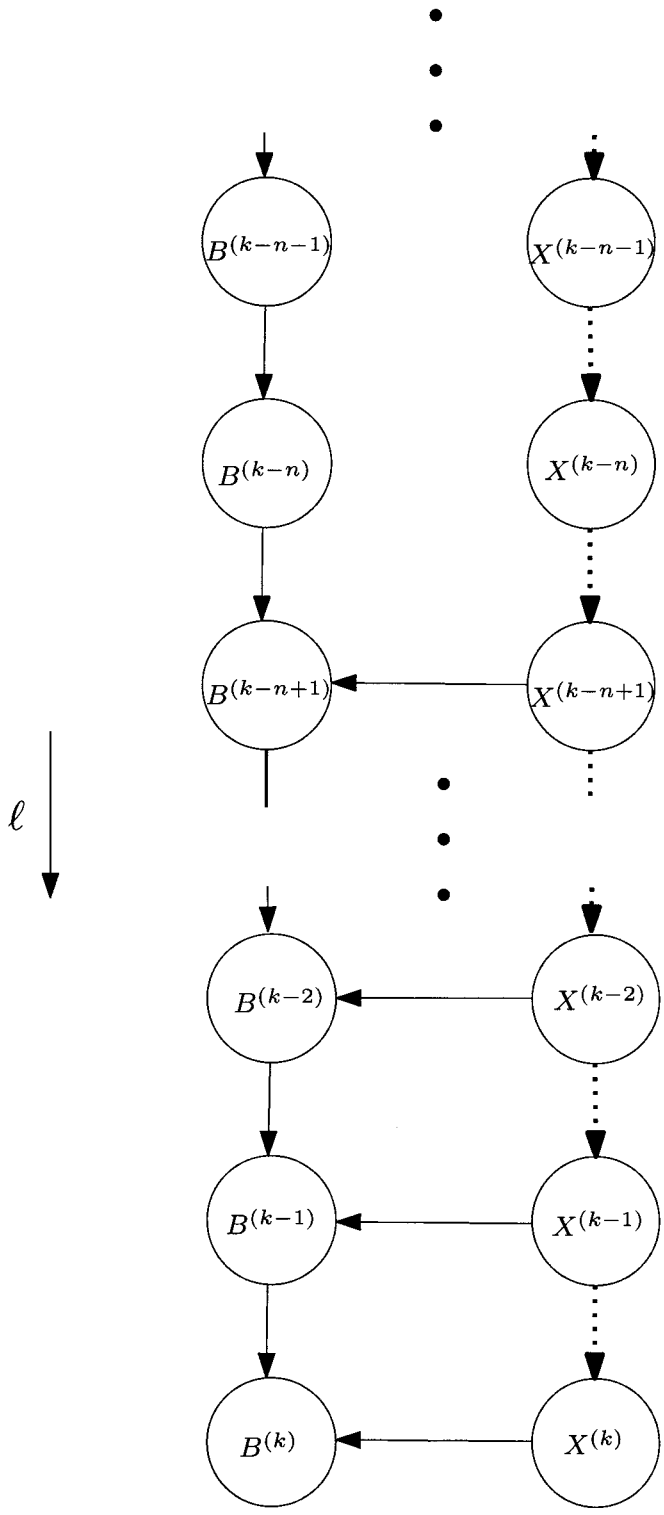


Figure 22: Removing links for Memory Truncation for the general case

5.3.1 Calculating $\mathbb{P}\left(B^{(k)}, X^{(k+1)}\right)$ for $\mathrm{MT}^{(n)}$

$$\begin{aligned}
&\mathbb{P}\left(B^{(k)}, X^{(k+1)}\right) \\
&= \sum_{x^{(k-n+2\to k)}} \mathbb{P}\left(B^{(k)}|X^{(k-n+2\to k)}\right)\mathbb{P}\left(X^{(k+1)}|X^{(k-n+2\to k)}\right)\mathbb{P}\left(X^{(k-n+2\to k)}\right) \\
&= \sum_{x^{(k-n+2\to k)}} \mathbb{P}\left(B^{(k)}|X^{(k-n+2\to k)}\right)\mathbb{P}\left(X^{(k-n+2\to k+1)}\right).
\end{aligned} \tag{69}$$

5.3.2 Calculating $\mathbb{P}\left(B^{(k)}|X^{(k-n+2\to k)}\right)$

$$\mathbb{P}\left(B^{(k)}|X^{(k-n+2\to k)}\right) = \sum_{b^{(k-1)}} \mathbb{P}\left(B^{(k)}|X^{(k)}, B^{(k-1)}\right)\mathbb{P}\left(B^{(k-1)}|X^{(k-n+2\to k)}\right). \tag{70}$$

5.3.3 Calculating $\mathbb{P}\left(B^{(k-1)}|X^{(k-n+2\to k)}\right)$

$$\begin{aligned}
\mathbb{P}\left(B^{(k-1)}|X^{(k-n+2\to k)}\right) &= \sum_{x^{(k-n+1)}} \mathbb{P}\left(B^{(k-1)}|X^{(k-n+1\to k)}\right)\mathbb{P}\left(X^{(k-n+1)}|X^{(k-n+2\to k)}\right) \\
&= \frac{1}{\mathbb{P}\left(X^{(k-n+2\to k)}\right)} \sum_{x^{(k-n+1)}} \mathbb{P}\left(B^{(k-1)}|X^{(k-n+1\to k-1)}\right)\mathbb{P}\left(X^{(k-n+1\to k)}\right).
\end{aligned} \tag{71}$$

$\mathbb{P}\left(B^{(k-1)}|X^{(k-n+1\to k-1)}\right)$ and $\mathbb{P}\left(X^{(k-n+1\to k)}\right)$ are obtained from the previous iteration and $\mathbb{P}\left(X^{(k-n+2\to k)}\right) = \sum_{x^{(k-n+1)}} \mathbb{P}\left(X^{(k-n+1\to k)}\right)$.

Notice here that we are using the fact that $B^{(k-1)} \perp\!\!\!\perp X^{(k)}|X^{(k-n+1\to k-1)}$. This is another consequence of memory truncation, and thus is considered to be an approximation.

5.3.4 Calculating $\mathbb{P}\left(X^{(k-n+2\to k+1)}\right)$

Calculating $\mathbb{P}\left(X^{(k-n+2\to k+1)}\right)$ can be performed the exact same as in section 4.4.2 given we have $\mathbb{P}\left(U_1^{(k-n+2\to k+1)}\right)$. Similarily, $\mathbb{P}\left(U_1^{(k-n+2\to k+1)}\right)$ can be found using

the same methods as in section 4.4.3 given that we have $\mathbb{P}\left(T_1^{(k-n+1\to k)}\right)$.

5.3.5 Calculating $\mathbb{P}\left(T_1^{(k-n+1\to k)}\right)$

We can calculate $\mathbb{P}\left(T_1^{(k-n+1\to k)}\right)$ if we have $\mathbb{P}\left(B^{(k-n+1\to k)}\right)$ as shown in Equation (54). We will follow a similar approach to that shown in Equation (55), and we will start with the second last line from (55).

$$\mathbb{P}\left(B^{(k-n+1\to k)}\right) = \prod_{i=k-n+2}^{k} \sum_{x^{(i)}} \mathbb{P}\left(B^{(i)}|B^{(i-1)}, X^{(i)}\right)\mathbb{P}\left(B^{(k-n+1)}, X^{(k-n+2\to k)}\right). \quad (72)$$

However, we cannot split up the joint probability distribution of $\mathbb{P}\left(B^{(k-n+1)}, X^{(k-n+2\to k)}\right)$ into the product of their respective marginals because they are not independent. In order to compute this joint distribution, we use the fact that under our memory truncation approximation of order n, $X^{(k-n+2\to k)} \perp\!\!\!\perp B^{(k-n)}|X^{(k-n+1)}$. We then have the following:

$$\begin{aligned}
&\mathbb{P}\left(B^{(k-n+1)}, X^{(k-n+2\to k)}\right) \\
&= \sum_{b^{(k-n)}} \sum_{x^{(k-n+1)}} \mathbb{P}\left(B^{(k-n+1)}|B^{(k-n)}, X^{(k-n+1)}\right)\mathbb{P}\left(X^{(k-n+2\to k)}|B^{(k-n)}, X^{(k-n+1)}\right)\times \\
&\mathbb{P}\left(B^{(k-n)}, X^{(k-n+1)}\right) \\
&= \sum_{b^{(k-n)}} \sum_{x^{(k-n+1)}} \mathbb{P}\left(B^{(k-n+1)}|B^{(k-n)}, X^{(k-n+1)}\right)\mathbb{P}\left(X^{(k-n+2\to k)}|X^{(k-n+1)}\right)\mathbb{P}\left(B^{(k-n)}, X^{(k-n+1)}\right) \\
&= \sum_{b^{(k-n)}} \sum_{x^{(k-n+1)}} \mathbb{P}\left(B^{(k-n+1)}|B^{(k-n)}, X^{(k-n+1)}\right)\mathbb{P}\left(X^{(k-n+1\to k)}\right)\mathbb{P}\left(B^{(k-n)}|X^{(k-n+1)}\right).
\end{aligned} \quad (73)$$

$\mathbb{P}\left(X^{(k-n+1\to k)}\right)$ is obtained from the previous iteration where $\mathbb{P}\left(B^{(k-n)}|X^{(k-n+1)}\right)$ is obtained from iteration $k-n+1$.

5.4 Validating Memory Truncation

We would expect the accuracy of our calculation of $\mathbb{P}\left(B^{(\ell)}\right)$ to increase with increasing memory truncation order n. However, it is also expected that after a particular order n, increasing to any order $k > n$ would have negligible improvement in terms of accuracy. Therefore the goal is to find the lowest memory truncation order as to reduce complexity and still preserve accuracy. Our results obtained in this section are based on the DD-BMP algorithm.

First we look at how the probability of error (P_e), defined by $P_e^{(\ell)} = P\left(B^{(\ell)} < 0\right) + \frac{1}{2}P\left(B^{(\ell)} = 0\right)$, is changing with respect to iteration ℓ. We show this for 3 different code ensembles in figures 23, 24 and 25. For each code, the optimal quantization parameter[1] Δ was used and the number of quantization bits is $q = 8$.

The first thing that we notice is the different convergence properties for different degree distributions. For example, in Fig. 25 for the $(4, 6)$ regular LDPC code, the probability of error is almost the same regardless of the memory truncation order. In general though, we have almost complete convergence using memory truncation of order 4. We also see that the probability of error is increasing with increasing memory truncation order. This is an indication of a lower bound on the probability of error although the proof for this does not yet exist. If this were to hold true for all degree distributions, this would imply that our threshold results as given in Chapter 6 are also a lower bound on the actual threshold.

We can also look at how the probability of error is changing with respect to the iteration number for a particular code ensemble and fixed E_b/N_0. Consider the

[1]The methods to finding this optimal parameter is explained in Chapter 6

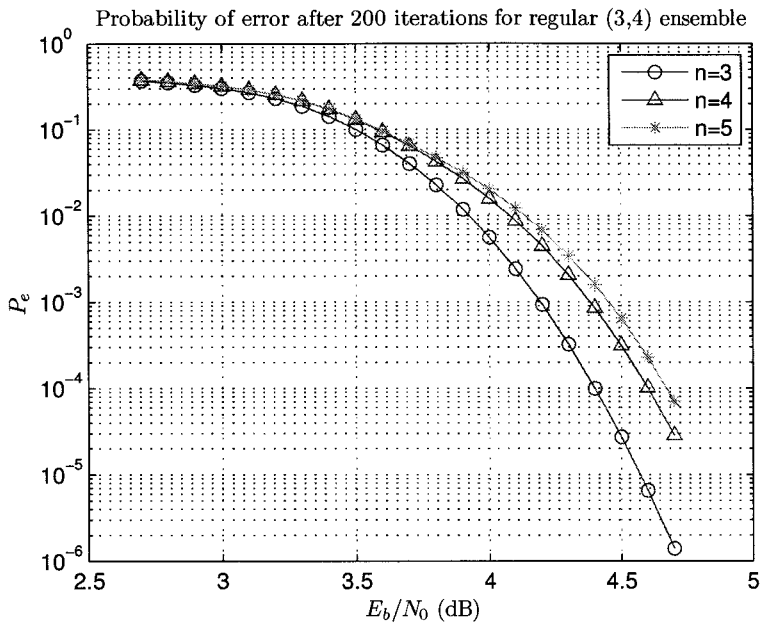


Figure 23: P_e after 200 iterations for regular (3,4) ensemble at memory truncation orders 3, 4, and 5 for varying E_b/N_0

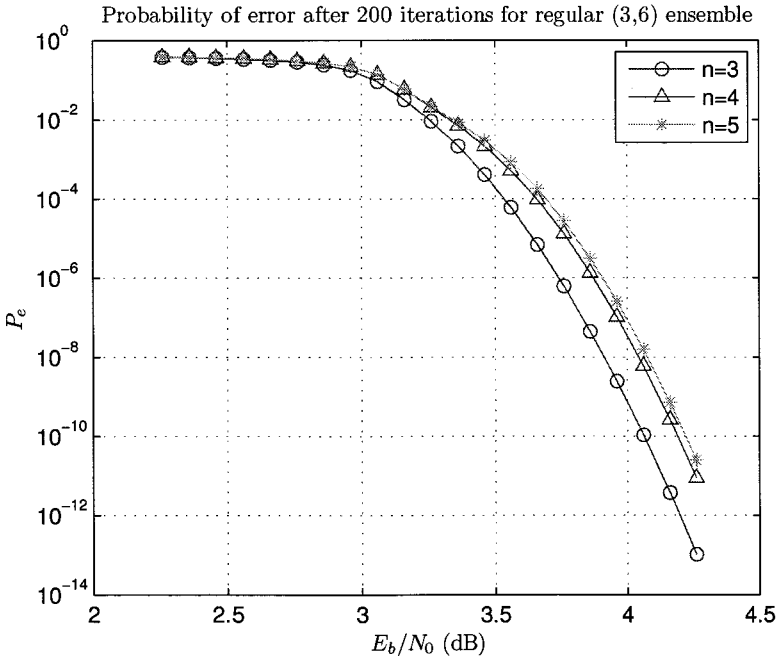


Figure 24: P_e after 200 iterations for regular (3,6) ensemble at memory truncation orders 3, 4, and 5 for varying E_b/N_0

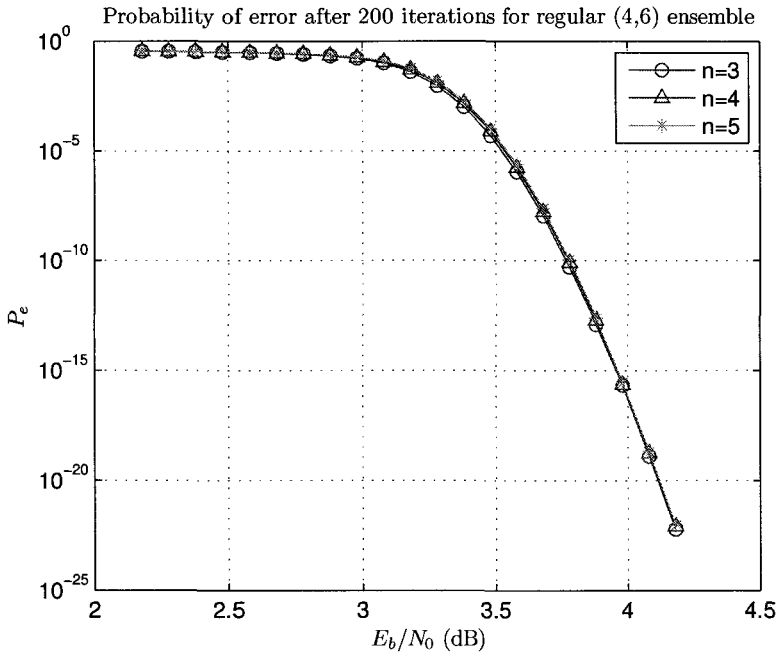


Figure 25: P_e after 200 iterations for regular (4,6) ensemble at memory truncation orders 3, 4, and 5 for varying E_b/N_0

regular (4,5) code ensemble. Setting $q = 8$, the optimal threshold[2] with respect to Δ is computed to be 4.24 dB. Figures 26 and 27 show how the probability of error is changing when we are ± 0.1 dB away from the threshold for memory truncation orders 3, 4, and 5.

In Fig. 26, all curves are converging toward 0 as expected. In Fig. 27, we are below the threshold and hence, P_e converges away from 0 as expected.

Lastly we show an example for the regular (3,7) ensemble in Fig. 28 setting $E_b/N_0 = 3.34$ dB which happens to be the calculated *threshold.* Once again, the curves for memory truncation 4 and 5 are very close together, and are shown to be converging to 0, albeit very slowly since we are at the threshold.

[2]Threshold is calculated using memory truncation of order 4.

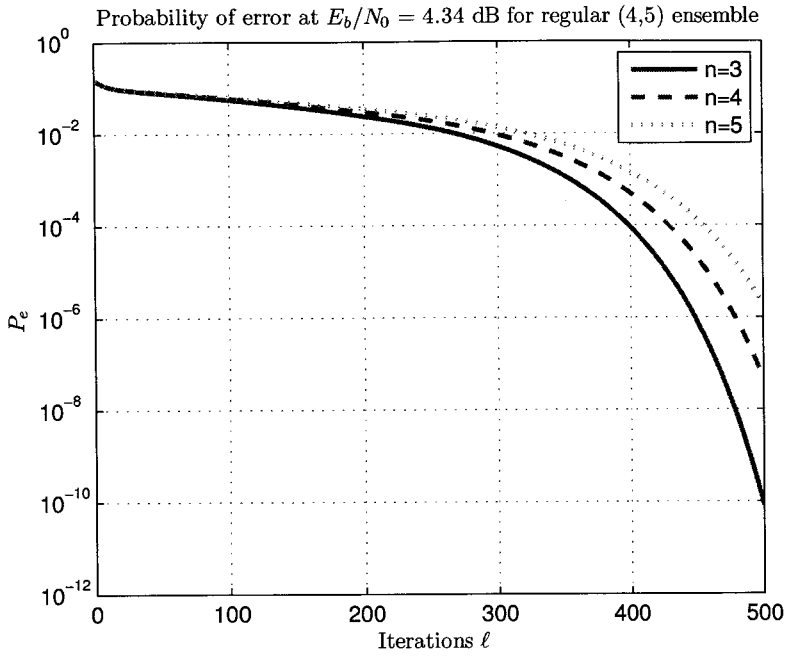


Figure 26: P_e 0.1 dB above threshold for regular (4,5) ensemble at memory truncation orders 3, 4, and 5

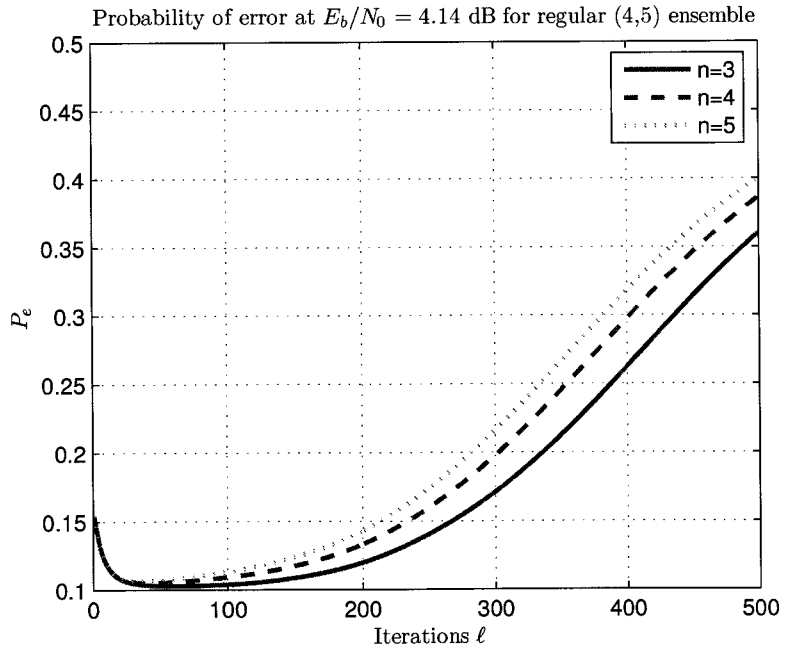


Figure 27: P_e 0.1 dB below threshold for regular (4,5) ensemble at memory truncation orders 3, 4, and 5

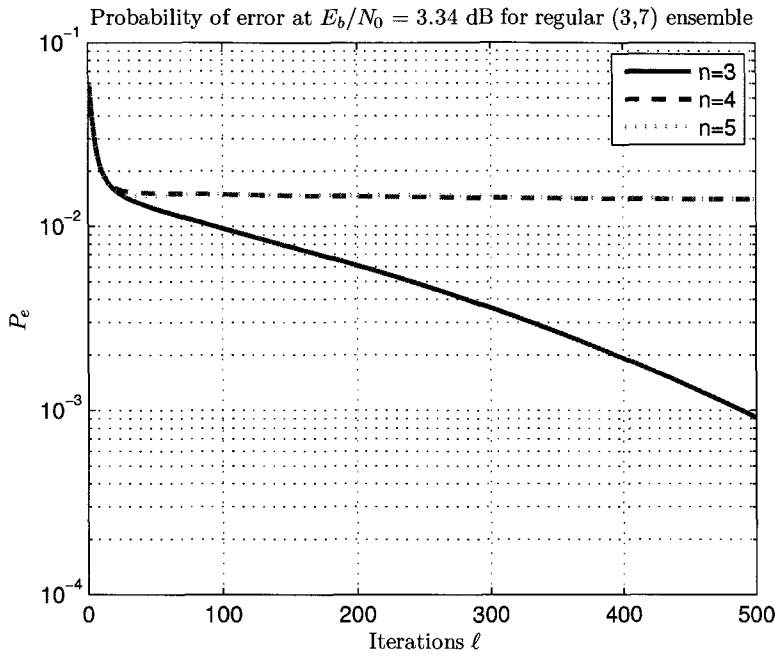


Figure 28: P_e at the threshold for regular (3,7) ensemble at memory truncation orders 3, 4, and 5

In the last example, most importantly, we show how the thresholds are changing for different memory truncation orders. We provide an example of this for the regular $(3, 6)$ LDPC code in Fig. 29. Once again we see convergence at $n = 4$ and conclude that this is a good choice for our simulations in Chapter 6.

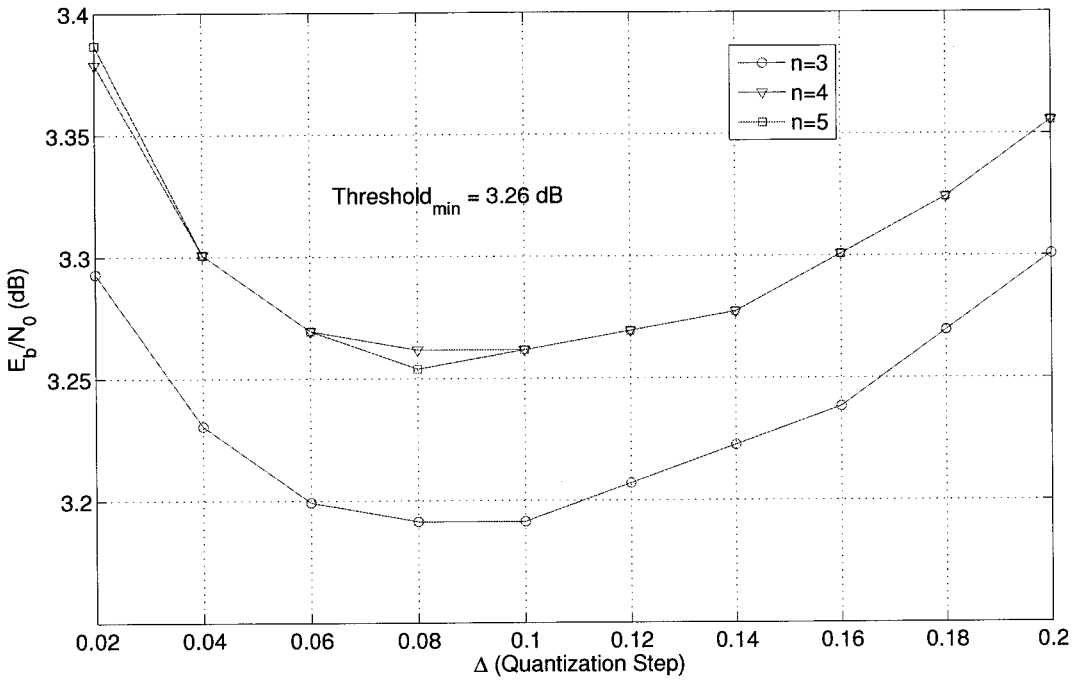


Figure 29: Threshold values for different Δ and memory truncation orders.

Chapter 6

Simulation Results and Discussion

In this chapter we present our simulation methodologies, results, and discussions for the DD-BMP algorithm. First we discuss how the threshold is determined, and in particular, how the *optimal* threshold can be found. For DD-BMP this optimality depends on Δ and q; the quantization step size and number of quantization bits respectively. We make comparisons to finite length codes in order to qualitatively verify the existence of a threshold. All the threshold results are based on using a memory truncation of order 4. Therefore, the accuracy of these thresholds are compared to finite length code simulations. We also observe some more convergent properties of the threshold with respect to the memory truncation order. These thresholds are calculated for a large family of codes, by varying the variable and check degrees. The simulated results are then compared to BP and MS and some interesting observations are made.

6.1 Threshold Determination

Before we describe our algorithm for threshold determination, we need to discuss the monotonic property of the DD-BMP decoder. For a decoder satisfying this property of monotonicity, we can use a simple bisection search to find the threshold which will

be explained later.

Definition 3 (Decoder Monotonicty). *Let W' be a channel which is physically degraded with respect to some other channel W. We define p_W^ℓ to be the expected number of incorrect messages passed in the ℓ^{th} decoding round when applied to the channel W. The decoder satisfies the monotonicity property if $p_W^\ell \leq p_{W'}^\ell$ for $\ell \geq 0$.*

For an AWGN channel, if channel W has noise variance σ_1 and channel W' has noise variance σ_2, then W' is physically degraded with respect to W if $\sigma_2 \geq \sigma_1$. It is natural to suspect that the decoder's performance degrades with increasing noise. Although the proof for monotonicity of the DD-BMP decoder is not provided, it can be assumed to hold true using the BER curves from [16, 17] as supporting evidence.

Assuming the monotonicity of the DD-BMP decoder, we now give the algorithm for threshold determination which is simply a bisection search. Note that the values for variables SNR_{MAX} and SNR_{MIN} are initialized in such a way such that $\text{SNR}_{\text{MAX}} > \text{SNR}_{\text{MIN}} \geq 0$ dB and the threshold lies somewhere in between these two values.

for $i := 1$ **to** I_t **do**
  $\text{SNR} \leftarrow \frac{\text{SNR}_{\text{MAX}} - \text{SNR}_{\text{MIN}}}{2}$
  if $\mathbb{P}\left(\text{sgn}\left(B^{(\ell)}\right) = 0\right) \rightarrow 1$[1] @ SNR **then**
    $\text{SNR}_{\text{MAX}} \leftarrow \text{SNR}$
  else
    $\text{SNR}_{\text{MIN}} \leftarrow \text{SNR}$
  end if
end for
threshold $\leftarrow$ SNR

The number of iterations for the above algorithm, represented by I_t depends on the desired level of precision. In our simulations, our threshold results are calculated

[1]Because we are working with finite precision, we check to see if $\mathbb{P}\left(\text{sgn}\left(B^{(\ell)}\right) = 0\right) > 1 - \epsilon$, for $\epsilon = 10^{-8}$, up to a maximum of 1000 iterations ℓ.

to a precision of 2 decimal points. Therefore, the value of I_t is calculated as follows:

$$\begin{aligned} \frac{\mathrm{SNR_{MAX}} - \mathrm{SNR_{MIN}}}{2^{I_t}} &< 0.005 \\ 2^{I_t} &> \frac{\mathrm{SNR_{MAX}} - \mathrm{SNR_{MIN}}}{0.005} \\ I_t &> \log_2(200) + \log_2(\mathrm{SNR_{MAX}} - \mathrm{SNR_{MIN}}). \end{aligned} \tag{74}$$

Initializing $\mathrm{SNR_{MAX}}$ and $\mathrm{SNR_{MIN}}$ to 8 dB and 0 dB respectively[2], $I_t > 10.64$. Therefore, setting I_t to 11 is sufficient to give the calculated threshold with a precision of two decimal points in dB.

6.2 Optimal Threshold

Finding the *threshold* for a regular LDPC code ensemble under DD-BMP decoding consists of setting the following parameters:

- d_v: variable node degree
- d_c: check node degree
- q: number of quantization bits
- I_t: number of precision iterations
- Δ: quantization step size

Our goal is thus the following: for a particular ensemble of LDPC codes characterized by d_v and d_c, we need to choose a suitable q and Δ in order to find the optimal threshold. For a particular code, the optimal threshold is defined as the lowest E_b/N_0 such that the probability of decoding error converges to zero with iterations.

[2]This is a suitable choice for $\mathrm{SNR_{MAX}}$ and $\mathrm{SNR_{MIN}}$ since the threshold for all the different codes lies in this range.

Raising the number of quantization bits q improves the performance[3] of the code. It has been observed that saturation occurs at $q = 8$. Thus, raising q past 8 bits has negligible performance gain. Therefore, for all simulations we use 8 quantization bits.

Finding the optimal quantization step size Δ was performed through simulations. For a particular code ensemble, we varied the parameter Δ and found the corresponding threshold value. In all cases, the curve E_b/N_0 vs. Δ was a convex function, therefore we simply chose the minimum value to be the optimal threshold. This is seen in Fig. 30 for different regular LDPC ensembles. The circled point refers to the optimal threshold for that particular code ensemble.

6.3 Comparison to a Finite Length Code

In this section, we simulate a regular $(5, 10)$ LDPC code for varying block lengths. For each blocklength, we construct a random parity-check matrix with each row containing 10 ones and each column containing 5 ones. In order to get closer to the best performance of the code, we remove all cycles of length 4 as it is generally beleived that the presence of short cycles degrade the performance of the code [31]. As with our density evolution results, we use the same parameters; $q = 8$ and the optimal Δ; which for the (5,10) regular LDPC ensemble is 0.035. For each SNR point, we transmit the all-zero codeword until 100 block errors occur. The results as well as the computed threshold are shown in Fig. 31.

From this graph we can make several observations. First, the trend of the curves agrees with our monotonicity property. Secondly, as the block length increases, the

[3]Improvement in performance implies a reduction in the threshold.

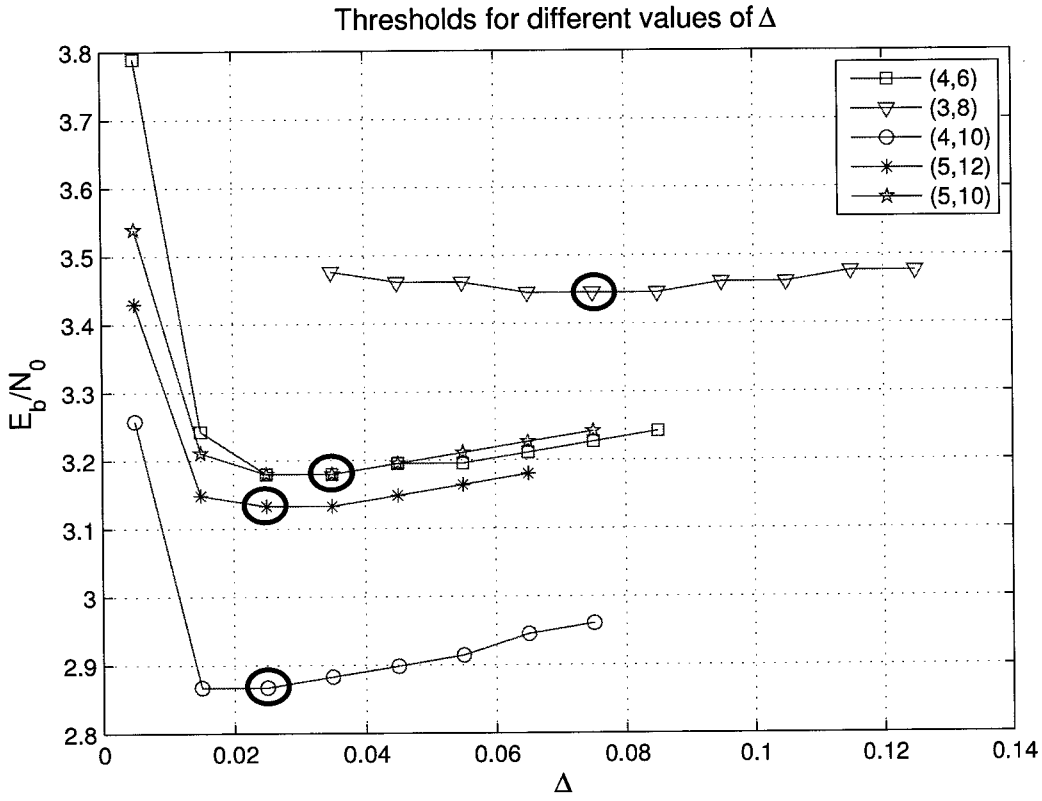


Figure 30: Finding the optimal threshold for different regular LDPC code ensembles

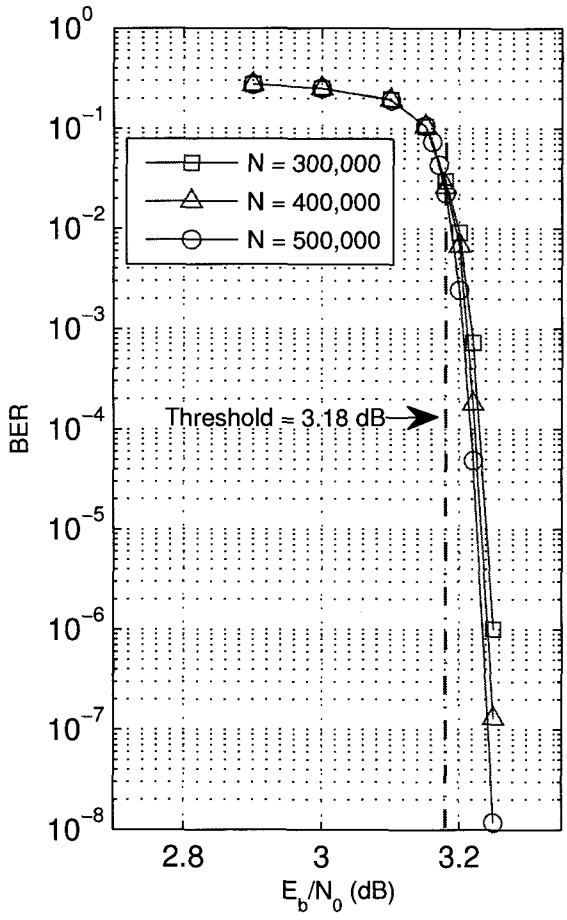


Figure 31: BER curves of $(5, 10)$ regular LDPC codes with block lengths 300,000, 400,000, and 500,000 decoded by DD-BMP, and the corresponding threshold.

slope of the waterfall region grows larger, as expected. As can be seen, the threshold value matches well with the waterfall region of the simulated error rate curves.

6.4 Results and Discussion

The thresholds are obtained for all regular (d_v, d_c) codes with $3 \leq d_v \leq 8$ and $d_v < d_c \leq 18$. All the threshold results are provided in Appendix B. For discussion purposes, in this section we only provide a subset of the results to highlight certain trends and to provide a comparison with the BP and MS decoding algorithms.

For a fixed d_v, the threshold gap between DD-BMP and BP[4] decreases with increasing the rate. The results for $d_v = 4$ and $d_v = 6$ are shown in Tables 1 and 2. As can be seen, at higher rates the performance gap is less than 1 dB. When comparing to MS[5], for codes with larger degrees, DD-BMP outperforms MS. The results for rate-1/2 codes are reported in Table 3. These results show that by increasing the degrees, the performance gap between MS and DD-BMP, which is to the advantage of MS for lower degrees, disappears and then reverses to the advantage of DD-BMP. These performance results for DD-BMP are impressive considering that both the check node operations and the message-passing for DD-BMP are much simpler than those of BP and MS. They also demonstrate the potential of iterative decoding algorithms with memory in achieving better performance/complexity trade offs compared to memoryless algorithms.

[4]The thresholds for BP were found using Gaussian Approximation [22].

[5]For the density evolution analysis of MS, 8-bit messages were used in order to have a fair comparison.

Table 1: Thresholds of BP and DD-BMP for ensembles with $d_v = 4$ and $5 \leq d_c \leq 18$

d_c	*rate*	$\left(\frac{E_b}{N_0}\right)_{BP}$ (dB)	$\left(\frac{E_b}{N_0}\right)_{DD-BMP}$ (dB)	Diff (dB)
5	1/5	2.57	4.24	1.67
6	1/3	1.73	3.18	1.45
7	3/7	1.27	2.88	1.61
8	1/2	1.59	2.80	1.21
9	5/9	1.67	2.82	1.15
10	3/5	1.78	2.87	1.09
11	7/11	1.89	2.93	1.04
12	4/6	1.99	2.99	1.00
13	9/13	2.10	3.06	0.96
14	5/7	2.20	3.13	0.93
15	11/15	2.29	3.20	0.91
16	3/4	2.39	3.27	0.88
17	13/17	2.47	3.33	0.86
18	7/8	2.55	3.40	0.85

Table 2: Thresholds of BP and DD-BMP for ensembles with $d_v = 6$ and $7 \leq d_c \leq 18$

d_c	*rate*	$\left(\frac{E_b}{N_0}\right)_{BP}$ (dB)	$\left(\frac{E_b}{N_0}\right)_{DD-BMP}$ (dB)	Diff (dB)
7	1/7	5.14	6.46	1.32
8	1/4	3.52	4.71	1.19
9	1/3	2.90	4.00	1.10
10	2/5	2.61	3.63	1.02
11	5/11	2.47	3.45	0.98
12	1/2	2.41	3.34	0.93
13	7/13	2.39	3.26	0.87
14	6/7	2.40	3.24	0.84
15	3/5	2.43	3.24	0.81
16	5/8	2.46	3.24	0.78
17	11/17	2.50	3.26	0.76
18	2/3	2.54	3.28	0.74

Table 3: Thresholds of MS and DD-BMP for rate-1/2 ensembles

d_v	d_c	$\left(\frac{E_b}{N_0}\right)_{MS}$ (dB)	$\left(\frac{E_b}{N_0}\right)_{DD-BMP}$ (dB)	Diff (dB)
3	6	1.70	3.26	1.56
4	8	2.49	2.80	0.31
5	10	3.09	3.18	0.09
6	12	3.54	3.34	-0.21
7	14	3.91	3.61	-0.30

6.5 Modifying the DD-BMP Algorithm

In this section we propose a modified version of DD-BMP which gives better performance compared to DD-BMP when analysed through density evolution. In light of the results presented, we are optimistic that many decoding algorithms with memory have yet to be discovered which provide great performance and low decoding complexity.

Consider the general memory update equation as given in Section 2.3.3:

$$B^{(\ell)} = K_c\big(\alpha B^{(\ell-1)} + \beta X^{(\ell)}\big). \tag{75}$$

For DD-BMP, the values for α and β are both 1. For the modified version, we still fix α to be 1, however, we modify the value of β in such a way to improve the performance. Our choice of β is inspired by the work done in [32]. In the work of [32], density evolution is performed on two modified BP-based[6] algorithms: the normalized and offset BP-based algorithm. In both cases, the reliabilities of the messages being passed is reduced and the resulting performance of the decoding algorithm was improved over the conventional BP-based algorithm.

In the modified DD-BMP algorithm, β is chosen to be less then 1 in order to reduce the reliability of the variable node output message $X^{(\ell)}$. The choice of β depends on d_v so that it compresses the message space of $X^{(\ell)}$ as follows:

$$\{-(d_v-1), -(d_v-3), ..., d_v-3, d_v-1\} \longrightarrow \left\{-\frac{d_v - d_v \bmod 2}{2}, ..., \frac{d_v - d_v \bmod 2}{2}\right\}. \tag{76}$$

[6]BP-based is analogous to Min-Sum.

Example 6. *Consider* $d_v = 3$. *Therefore* $S_X := \{-2, 0, 2\}$. *In order to compress the messages according to equation 76, we choose* $\beta = 0.5$. *Therefore, our new set of variable node output messages are now* $\{-1, 0, 1\}$.

Example 7. *In this example, we take* $d_v = 4$. *Therefore* $S_X := \{-3, -1, 1, 3\}$. *It is obvious that we cannot find a value of* β *to compress the message space to* $\{-2, -1, 1, 2\}$. *However, because of the clipping function* K_c, *we can do following. Choose* $\beta = 0.7$. *Therefore the message space of the variable node output is now* $\{-2.1, -0.7, 0.7, 2.1\}$ *which is equivalent to a message space of* $\{-2, -1, 1, 2\}$ *since* K_c *always rounds to the nearest integer.*

We now show how the performance of the modified version compares with the original DD-BMP algorithm. The thresholds are calculated[7] for various regular code ensembles and presented in Table 4. It is clear the by compressing the messages at the variable node output, better performance is achieved. Even though the performance gap is small, we have made the observation that better decoding algorithms with memory with low complexity exist.

It is natural to observe some kind of trade-off between DD-BMP and modified DD-BMP. Even though in the asymptotic case, modified DD-BMP gives better results, how does the performance compare in a finite block length scenario. We construct a random regular (4,8) LDPC code of block length 10,000 and set the maximum number of iterations to 100, 200, and lastly to 300. We then simulate the bit error rate curves of DD-BMP ($\beta = 1.0$) and modified DD-BMP ($\beta = 0.7$). This is shown in Fig. 32.

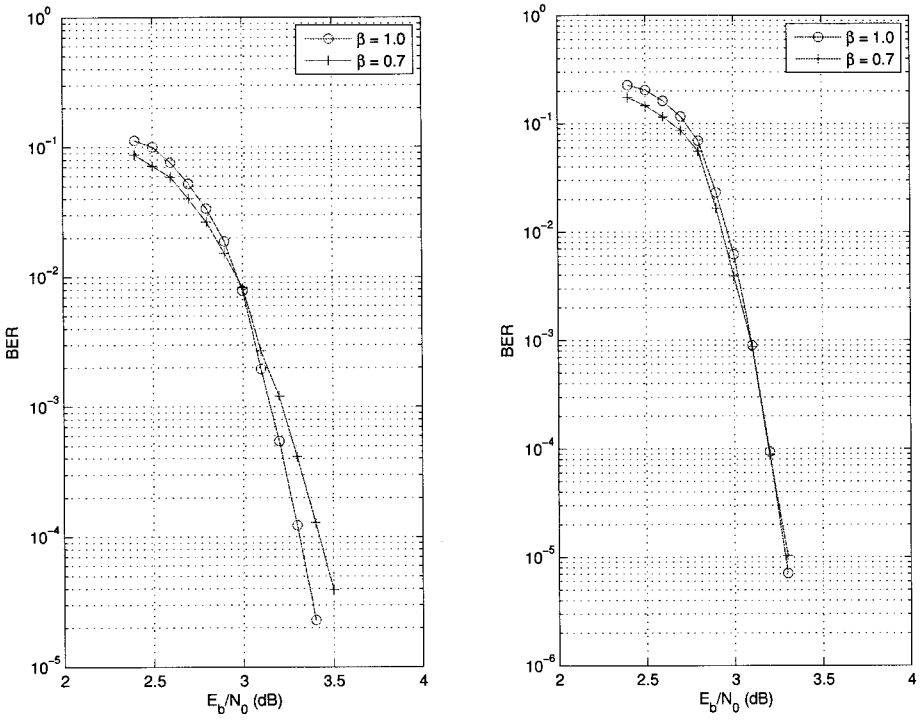
We observe the following results. At 100 maximum iterations as shown in Fig. 32(a), the original DD-BMP algorithms has better performance at higher SNR's.

[7]As with previous threshold calculations, memory truncation of order 4 is used.

Table 4: Thresholds of DD-BMP and Modified DD-BMP for different code ensembles

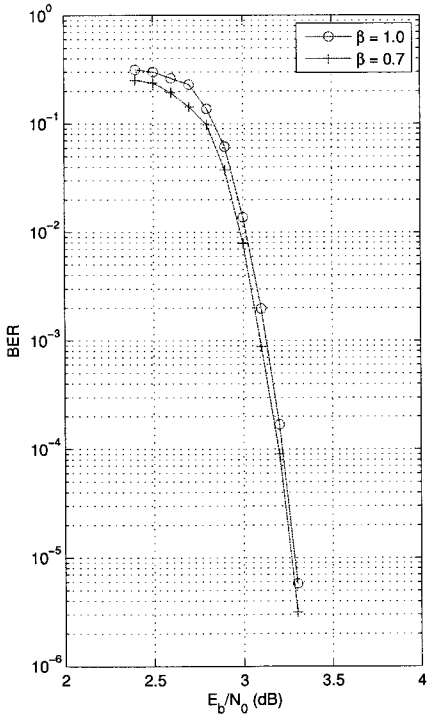
d_v	d_c	$\left(\frac{E_b}{N_0}\right)_{DD-BMP}$ (dB)	$\left(\frac{E_b}{N_0}\right)_{\text{Modified } DD-BMP}$ (dB)	Diff (dB)
3	4	3.71	3.49	0.22
3	5	3.26	3.12	0.14
3	6	3.26	3.11	0.15
3	7	3.34	3.22	0.12
3	8	3.45	3.34	0.11
4	5	4.24	4.15	0.09
4	6	3.18	3.10	0.08
4	7	2.88	2.80	0.08
4	8	2.80	2.72	0.08

However, as the number of maximum iterations is increased, the performance is better for the modified DD-BMP algorithm as expected. A higher number of iterations is required for modified DD-BMP since we are reducing the reliabilities of messages. Hence, we expect the convergence to be slower. The drawback of compressing the reliabilities is having to decode with more iterations in order to show improvements over the original DD-BMP algorithm.



(a) 100 Iterations

(b) 200 Iterations



(c) 300 Iterations

Figure 32: BER curves of (4,8) regular LDPC code with block length 10,000 and maximum number of iterations of a) 100, b) 200, and c) 300. Codes are decoded using DD-BMP and modified DD-BMP with $\beta = 0.7$.

Chapter 7

Future Work

We now propose a list of future works in relation to iterative decoding algorithms with memory. Since the technique of density evolution for iterative message passing algorithms with memory is a novel idea, there is still a lot of theoretical work that can be performed. As well, new and improved algorithms may be found which can be verified through density evolution simulations. The following lists some directions for future work:

- Density evolution analysis of irregular LDPC codes and finding the thresholds of irregular ensembles
- Density evolution analysis applied to the case where memory nodes exist in both directions on a particular edge, as well as for instances where memory nodes may exist between check nodes and variable nodes
- Finding optimal degree profiles for irregular LDPC codes decoded by iterative message passing algorithms with memory
- Finding the optimal α, β, and γ for the general algorithm described in Section 2.3.3
- Proving the monotonicity of decoding algorithms with memory

- Finding the relationship between memory truncation order n and the degree distributions
- Validating memory truncation from a graph theoretic point of view, i.e. proving that using memory truncation of order n as a function of the code parameters and the decoding algorithm is suitable to find the threshold within precision ϵ from the exact threshold, for some $\epsilon > 0$
- Proving concentration results (as is done for memoryless algorithms) for iterative message passing algorithms with memory
- Finding a suitable memory truncation analysis for SR algorithms
- Finding thresholds as well as designing irregular codes for SR algorithms
- Analysis of iterative message passing algorithms with memory applied to channels with memory, i.e. fading channels

Chapter 8

Conclusion

In this thesis, we have developed the framework for the density evolution analysis of iterative message passing algorithms with memory over memoryless channels. The general model for iterative decoding algorithms with memory was presented which includes DD-BMP and SR algorithms as special cases. Based on this model, we employed the directed acyclic graph representation, also known as Bayesian network, to capture the dependencies among different messages and memory contents in a space with two dimensions: iteration number ℓ and the depth of the decoding tree d. Using this, we showed that for algorithms with memory, the incoming messages to a node are no longer independent, and that the existing dependencies cause the complexity of density evolution to grow at least exponentially with ℓ. We also derived the density evolution equations and used techniques to make them tractable. In particular, we introduced an approximation by truncating the memory of the decoding process and estimating it with a Markov process of finite order n. As an example, we applied the proposed density evolution technique to DD-BMP for a number of regular LDPC code ensembles and obtained their thresholds. The threshold results converged to the true threshold values with arbitrary precision as the memory truncation length n was increased. Our results showed that for DD-BMP, $n = 4$ is often enough to obtain the threshold with a good accuracy. A careful inspection of the threshold values of

DD-BMP for regular LDPC codes in comparison with BP and MS thresholds revealed a number of interesting phenomena: (a) By increasing the rate, the performance gap between BP and DD-BMP reduced so that for high rate codes the difference is only a fraction of a dB; (b) For a given rate, by increasing the degrees, the performance advantage of MS over DD-BMP decreases at first and then eventually reverses so that for large degrees, DD-BMP outperforms MS by a few tenths of a dB. These performance results for DD-BMP are very impressive since both the message passing and the check node operations in DD-BMP are significantly less complex that those in BP and MS algorithms. In addition to the above results, our work introduced the general framework for presenting and analysing iterative decoding algorithms with memory. This can be used to design irregular codes for existing algorithms or to devise new algorithms with improved performance/complexity tradeoffs.

List of References

[1] R. Gallager, "Low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 8, pp. 21–28, January 1962.

[2] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533–547, Sep 1981.

[3] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, Mar 1974.

[4] J. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inform. Theory*, vol. 24, pp. 76–80, Jan 1978.

[5] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar 1999.

[6] D. MacKay and R. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, p. 1645, Aug 1996.

[7] J. Pearl, *Probabilistic reasoning in intelligent systems; networks of plausible inference*. San Mateo, Ca., Kaufmann., 1988.

[8] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Comm.*, vol. 47, pp. 673–680, May 1999.

[9] E. Eleftheriou, T. Mittelholzer, and A. Dholakia, "Reduced-complexity decoding algorithm for low-density parity-check codes," *Electronics Letters*, vol. 37, pp. 102–104, Jan 2001.

[10] J. Chen and M. Fossorier, "Decoding low-density parity check codes with normalized APP-based algorithm," *Proc. IEEE Global Telecommunications Conference, 2001*, vol. 2, pp. 1026–1030, Nov 2001.

[11] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," *Proc. IEEE Global Telecommunications Conference, 2001*, vol. 2, pp. 1036–1036E, Nov 2001.

[12] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Communications Letters*, vol. 50, pp. 406–414, Mar 2002.

[13] S. Hemati and A. H. Banihashemi, "Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes," *IEEE Trans. Comm.*, vol. 54, pp. 61–70, Jan. 2006.

[14] H. Xiao, S. Tolouei, and A. H. Banihashemi, "Successive relaxation for decoding of LDPC codes," *Proc. IEEE 24th Biennial Symposium on Comm. Queens Univ.*, vol. 5, pp. 107–110, June 2008.

[15] H. Xiao and A. Banihashemi, "Comments on successive relaxation for decoding of ldpc codes," *IEEE Trans. Comm.*, vol. 57, pp. 2846 –2848, october 2009.

[16] N. Mobini, A. H. Banihashemi, and S. Hemati, "A differential binary message-passing LDPC decoder," *IEEE Trans. Comm.*, pp. 2518–2523, Sept 2009.

[17] N. Mobini, A. H. Banihashemi, and S. Hemati, "A differential binary message-passing LDPC decoder," *Proc. IEEE Global Telecommunications Conference*, pp. 1561–1565, Nov. 2007.

[18] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb 2001.

[19] S.-Y. Chung, J. Forney, G.D., T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Comm. Lett.*, vol. 5, pp. 58–60, Feb 2001.

[20] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb 2001.

[21] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," *Proc. IEEE Global Telecommunications Conference, 2001*, vol. 2, pp. 1021–1025, 2001.

[22] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inform. Theory*, vol. 47, pp. 657–670, Feb 2001.

[23] J. Chen and P. Fossorier, "Density evolution for BP-based decoding algorithms of LDPC codes and their quantized versions," *Proc. IEEE Global Telecommunications Conference, 2002*, vol. 2, pp. 1378–1382, Nov. 2002.

[24] A. Eckford, F. Kschischang, and S. Pasupathy, "Analysis of low-density parity-check codes for the gilbert-elliott channel," *IEEE Trans. Inform. Theory*, vol. 51, pp. 3872–3889, Nov. 2005.

[25] J. Hou, P. Siegel, and L. Milstein, "Performance analysis and code optimization of low density parity-check codes on rayleigh fading channels," *IEEE Trans. Comm.*, vol. 19, pp. 924–934, May 2001.

[26] A. P. Worthen and W. E. Stark, "Low-density parity check codes for fading channels with memory," *Proc. of the 36th Annual Allerton Conference on Communication, Control and Computing*, pp. 117–125, 1998.

[27] A. Kavcic, X. Ma, and M. Mitzenmacher, "Binary intersymbol interference channels: Gallager codes, density evolution, and code performance bounds," *IEEE Trans. Inform. Theory*, vol. 49, pp. 1636–1652, July 2003.

[28] J. A. T. Thomas M. Cover, *Elements of Information Theory, Second Edition.* John Wiley & Sons, 2006.

[29] F. V. Jensen and T. D. Nielsen, *Bayesian Networks and Decision Graphs.* New York, NY: Springer Science, second ed., 2007.

[30] R. D. Shachter, "Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams)," *Proc. Fourteenth Conference in Uncertainty in Artificial Intelligence*, pp. 480–487, 1998.

[31] Y. Mao and A. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," *Proc. Int. Conf. Communications*, vol. 1, pp. 41 –44 vol.1, June 2001.

[32] J. Chen and M. Fossorier, "Density evolution for two improved bp-based decoding algorithms of ldpc codes," *Communications Letters, IEEE*, vol. 6, pp. 208–210, May 2002.

Appendix A

Representing Elements through Integers

Consider a finite set of elements S which contains $|S|$ elements. We introduce a mapping $\mathcal{N} : S \rightarrow \{0, 1, ..., |S| - 1\}$. Now $\mathcal{N}$ is necessarily a bijection so that each element from S corresponds to a distinct integer between 0 and $|S| - 1$. We will use N_S to denote the set $\{0, 1, ..., |S| - 1\}$. This provides an index which can be used to reference a particular element in S. For example, $S(i)$, $i \in N_S$, corresponds to a particular element in S. As well, $S(A)$, $A \subset N_S$ will correspond to a collection of elements in S. We now show how this mapping is made for different types of sets.

In iterative decoding algorithms with memory, we need to describe the possible incoming and outgoing messages of a node, as well as the content of the memory. In each of these cases, the messages/content are a collection of distinct integers. Thus, $S \subset \mathbb{Z}$. If we order the elements of S in increasing order, then the first element corresponds to 0, the second element corresponds to 1, and so on. For example, if the message space $S_M = \{-3, -1, 1, 3\}$, then $\mathcal{N}(S_M) = \{0, 1, 2, 3\}$.

We will also deal with vector elements that are taken from the set $S^n \subset \mathbb{Z}^n$. Therefore, each $s \in S^n$ can be represented by the sequence $s_1, s_2, ..., s_n$ where $s_i \in S$. We can now define the function $\vec{\mathcal{N}}$ on vector elements:

$$\vec{\mathcal{N}}(s) = \sum_{i=1}^{n} \mathcal{N}(s_i) \times |S|^{i-1} \tag{77}$$

Example 8. *Let* $S = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ *and let* $a = \{-3, 0, 1, -1, -2, 4\}$ *where* $a \in S^6$. *First we change each sub-element of* a *into its corresponding integer value;* $a' = \{1, 4, 5, 3, 2, 8\}$. *Now* $\vec{\mathcal{N}}(a)$ *is simply the base 10 representation of* a' *which is a 6 digit number in base 9 where the right most number is the most significant. Hence,* $\vec{\mathcal{N}}(a) = 488143$. *Conversely,* $S^6(488143) = a$.

Appendix B

Threshold results for DD-BMP and BP

Table 5: Thresholds for BP and DD-BMP

d_v	d_c	$rate$	$\left(\frac{E_b}{N_0}\right)_{BP}$ (dB)	$\left(\frac{E_b}{N_0}\right)_{DD-BMP}$ (dB)	Diff (dB)	Δ_{optimal}
3	4	1/4	1.06	3.71	2.65	0.165
3	5	2/5	0.97	3.26	2.29	0.105
3	6	1/2	1.16	3.26	2.10	0.080
3	7	4/7	1.39	3.34	1.95	0.075
3	8	5/8	1.59	3.45	1.86	0.075
3	9	2/3	1.79	3.57	1.78	0.065
3	10	7/10	1.96	3.66	1.71	0.065
3	11	8/11	2.11	3.77	1.66	0.055
3	12	3/4	2.23	3.87	1.64	0.060
3	13	10/13	2.38	3.93	1.55	0.050
3	14	11/14	2.50	4.01	1.51	0.050
3	15	4/5	2.61	4.09	1.48	0.045
3	16	13/16	2.72	4.17	1.45	0.050
3	17	14/17	2.81	4.24	1.43	0.050
3	18	5/6	2.93	4.31	1.38	0.050

Table 6: Thresholds for BP and DD-BMP (Continued)

d_v	d_c	$rate$	$\left(\frac{E_b}{N_0}\right)_{BP}$ (dB)	$\left(\frac{E_b}{N_0}\right)_{DD-BMP}$ (dB)	Diff (dB)	Δ_{optimal}
4	5	1/5	2.57	4.24	1.67	0.040
4	6	1/3	1.73	3.18	1.45	0.030
4	7	3/7	1.57	2.88	1.31	0.030
4	8	1/2	1.59	2.80	1.21	0.025
4	9	5/9	1.67	2.82	1.15	0.025
4	10	3/5	1.78	2.87	1.09	0.020
4	11	7/11	1.89	2.93	1.04	0.020
4	12	2/3	1.99	2.99	1.00	0.020
4	13	9/13	2.10	3.06	0.96	0.020
4	14	5/7	2.20	3.13	0.93	0.020
4	15	11/15	2.29	3.20	0.91	0.020
4	16	3/4	2.39	3.27	0.88	0.020
4	17	13/17	2.47	3.33	0.86	0.015
4	18	7/9	2.55	3.40	0.85	0.020
5	6	1/6	3.96	5.59	1.63	0.055
5	7	2/7	2.65	4.09	1.44	0.045
5	8	3/8	2.23	3.54	1.32	0.040
5	9	4/9	2.07	3.29	1.22	0.035
5	10	1/2	2.04	3.18	1.14	0.030
5	11	6/11	2.05	3.15	1.10	0.035
5	12	7/12	2.09	3.13	1.04	0.030
5	13	8/13	2.15	3.14	0.99	0.030
5	14	9/14	2.21	3.17	0.96	0.025
5	15	2/3	2.28	3.20	0.92	0.025
5	16	11/16	2.35	3.25	0.90	0.020
5	17	12/17	2.42	3.29	0.87	0.020
5	18	13/18	2.48	3.34	0.86	0.020

Table 7: Thresholds for BP and DD-BMP (Continued)

d_v	d_c	*rate*	$\left(\frac{E_b}{N_0}\right)_{BP}$ (dB)	$\left(\frac{E_b}{N_0}\right)_{DD-BMP}$ (dB)	Diff (dB)	Δ_{optimal}
6	7	1/7	5.14	6.46	1.32	0.025
6	8	1/4	3.52	4.71	1.19	0.025
6	9	1/3	2.90	4.00	1.10	0.020
6	10	2/5	2.61	3.63	1.02	0.015
6	11	5/11	2.47	3.45	0.98	0.020
6	12	1/2	2.51	3.34	0.93	0.015
6	13	7/13	2.39	3.26	0.87	0.015
6	14	4/7	2.40	3.24	0.84	0.015
6	15	3/5	2.43	3.24	0.81	0.015
6	16	5/8	2.46	3.24	0.78	0.015
6	17	11/17	2.50	3.26	0.76	0.015
6	18	2/3	2.54	3.28	0.74	0.015
7	8	1/8	6.15	7.46	1.31	0.030
7	9	2/9	4.31	5.49	1.18	0.025
7	10	3/10	3.54	4.64	1.10	0.025
7	11	4/11	3.14	4.18	1.04	0.025
7	12	5/12	2.92	3.90	0.98	0.020
7	13	6/13	2.79	3.73	0.94	0.020
7	14	1/2	2.72	3.61	0.89	0.015
7	15	8/15	2.69	3.55	0.86	0.020
7	16	9/16	2.68	3.51	0.83	0.015
7	17	10/17	2.68	3.48	0.80	0.015
7	18	11/18	2.69	3.48	0.79	0.015