ASTON UNIVERSITY

Advanced Decoding Technics For Low Density Parity Check Codes Decoding

YOUSSEF ALLAOUI

MSc by Research in Pattern Analysis and Neural Networks, 2004

Thesis Summary

Error-correcting codes based on very sparse matrices provide the best performance to date. We will focus on improving the decoding of Low Density Parity Checks Codes by examining a new variant of belief propagation for this problem. This new variant is based on techniques such as time averaging and prior biasing. Results obtained by extensive simulations show that, well combined, these techniques bring a modest improvement to BP decoding performance. We also studied the possibility of parallelism, which consists of breaking the algorithm in independent pieces that can be solved simultaneously on different computer nodes. But the time spent in inter-communication between the different machines slows the process and prevent the parallelisation to be as fast as expected.

Keywords: Low Density Parity Checks Codes, Belief Propagation, Message Passing, Time Averaging, Parallelisation

Acknowledgements

First , I would like to thank my supervisors, Professor David Saad and Doctor Jort van Mourik for their general advice and attention during this project.

I am generally grateful to all my lecturers in the MSc PANN course, Professor David Lowe, Prof Ian T Nabney, Doctor Manfred Opper and Professor David Lowe . And more generally, I would like to thank all the NCRG researchers for their constant enthusiasm and the familiar ambiance they keep within the department. Many thanks to Ms Vicky Bond for her care and attention.

I would like to thank all my fellow MSc students from the Neural Computing Research Group: Christophe, Thomas, Youenn, Ben, Kiriko, Dharmesh, Pierre, Dan, for their constant support and their friendship through all this year.

I also want to thank my friends from Aston, Mohamed, Kleopatra, David for their friendly presence through all this year.

And in general my friends from Birmingham Marie-Hélène, Claire, Elwafi, Sabri, Moha, and my friends from l'Almont for their friendship and the good time we spent together.

Last but certainly not least, I would like to express my infinite gratitude to my parents, my sisters Maryame, Saliha, Latifa and Malika, my brother Omar and all my family for their love and encouragement.

Contents

1 Introduction **7**

1.1 Noisy Channel Communication 7

1.1.1 Noisy Channels 8

1.1.2 Error-Correcting Codes 8

1.1.3 Low Density Parity Check Codes 11

1.2 Decoding Process 11

1.2.1 Message Passing Techniques 11

2 Current Decoding Methods **13**

2.1 Belief Propagation 13

2.2 Limits of Optimal Decoding 18

2.2.1 Shannon's Bound 18

2.2.2 Dynamical Transition 19

3 Improvement to Message Passing **23**

3.1 Free Energy Minimisation 23

3.2 Improvements 25

3.2.1 Prior Biasing 25

3.2.2 Time Averaging 26

3.2.3 Expectations 26

4 Experiments and Results **28**

4.1 Implementation 28

4.1.1 Random Graph Generation 28

4.1.2 Belief Propagation Implementation 28

4.2 Simulations 29
4.2.1 Prior Biasing 29
4.2.2 Time Averaging 31
4.2.3 Mixing Methods 31
4.3 Interpretations of the results 31

5 Parallelisation on a Cluster **36**
5.1 What is a Cluster 36
5.1.1 Description 36
5.1.2 NCRG Cluster Configuration 37
5.2 Belief Propagation Parallelisation 37
5.2.1 Implementation 37
5.2.2 Speed Performance 38

6 Conclusions **41**

A Notations Used **46**

B Belief Propagation Algorithms **48**
B.1 Message-Passing 48
B.1.1 Message-Passing Algorithm : Random Method 48
B.1.2 Message-Passing Algorithm: Iterative Method 49
B.2 Functions Detail Algorithm 49
B.2.1 Initialisation 49
B.2.2 Checks Update: Update-R() 50
B.2.3 Nodes Update: Update-Q() 50
B.2.4 Test Found Solution: Test-Result() 51
B.2.5 Time Averaging 51
B.2.6 Sequential Fixing 52

C Graph Generation Detailed Algorithm **53**
C.1 Random Graph Generation 53

List of Figures

1.1 Gaussian channel 9
1.2 pictorial illustration of a BSC 9
1.3 Geometric representation of error correcting code 10

2.1 Tanner Graph 14
2.2 Loop in the bipartite graph 15
2.3 Message Passing characteristic variables 16
2.4 BP pictorial illustration 18
2.5 Geometric representation of the dynamical and critical transition . . . 20
2.6 Standard BP decoding performance 21
2.7 BP performance close to the critical transition 22
2.8 Suboptimal solution probability distribution 22

3.1 Factor graph 24
3.2 Effect of correct fixing of nodes values on BP decoding 27

4.1 Prior biasing effect on BP 30
4.2 Prior biasing effect on BP 30
4.3 Histograms for different values of α and $f = 0.17$ 32
4.4 Histograms for different values of α and $f = 0.2$ 33
4.5 Time averaging effect on BP 34
4.6 Both methods (time averaging + prior biasing) effect on BP 34

5.1 Distribution of tasks on a cluster 38
5.2 BP decoding speed: sequential vs. parallel 39
5.3 Sending data time on the cluster 40

Chapter 1

Introduction

During any information transmission, there is some probability for the received message to be different to the transmitted one due to corruption in the transmission medium.

Error-less transmission can be achieved by properly encoding the message, building in structured redundancy. The encoded message is termed codeword. Even if errors occur during the transmission due to some noise, the correct message can be restored by exploiting this redundancy. Retrieving the original message by decoding the received, corrupted codeword, at the receiving which is a crucial element in the process.

The aim of this project is to apply and explore the capability of a new variant of belief propagation to this particular decoding problem. A secondary objective is studying the efficiency of parallelisation in this context. The following sections are a presentation of the model of transmission used along the project and an introduction to belief propagation.

1.1 Noisy Channel Communication

Two common features of digital modern communication are the redundancy of the message to be transmitted and the corruption that may occur in the medium of transmission. Shannon was one of the first to study these issues [12]. He proved general results on the natural limits of compression or source coding and error-correction (channel coding) and set up the framework of what is now known as information theory.

Shannon's channel coding theorem states that error-free communication is possible if some redundancy is added to the original message in the encoding process.

1.1.1 Noisy Channels

Different models exist to represent a noisy channel. The most commonly used ones are the Gaussian and the Binary Symmetric Channel (BSC). The former represents the transmitted message bits as real values which are being corrupted by white Gaussian noise whereas the latter represents the noise by flipping the transmitted bits with a certain probability f.

For the Gaussian channel(also known as the Additive White Gaussian Noise (AWGN) channel), transmitted bits $t \in 0, 1$ are mapped to transmitted signals $x \in -x_0, +x_0$ and the output is $y = x + v$ where v is a zero mean normally distributed random variable with variance σ^2. We set $\sigma = 1$ and vary the signal amplitude x_0 to control the signal to noise ratio (SNR). This is illustrated in Figure 1.1. We declare the received bit $r = 1$ if $y > 0$ and $r = 0$ otherwise. The likelihood of this bit being in error though this mapping is:

$$f^1 = P(n = 1|y) = (1 + \exp^{2x_0 y})^{-1}, \tag{1.1}$$

where $r = t+n$ (mod 2). f^0, probability for each bit keeping its correct state, is defined as $1 - f$. For a code of rate R, the signal to noise ratio is equal to: $E_b/N_0 = x_0^2/2R\sigma^2$. To report this in decibels, one should compute $10log_{1O}(E_b/N_0)$.

We will focus during our work on the BSC as the treatment is simpler in binary model. Indeed the noise corruption can be represented by a noise vector such that the received corrupted codeword takes the form $r = t+n$. This is illustrated in Figure 1.2

1.1.2 Error-Correcting Codes

Error-correcting codes are based on mapping the original space of messages onto a higher dimensional space in such a way that the typical distance between encoded words (codewords) increases (as shown in Figure 1.3). This makes the corruption caused by the noise to the transmitted word to be recognisable. In the case of Low Density Parity Check (LDPC) Codes , this transformation is represented by the generator matrix G,

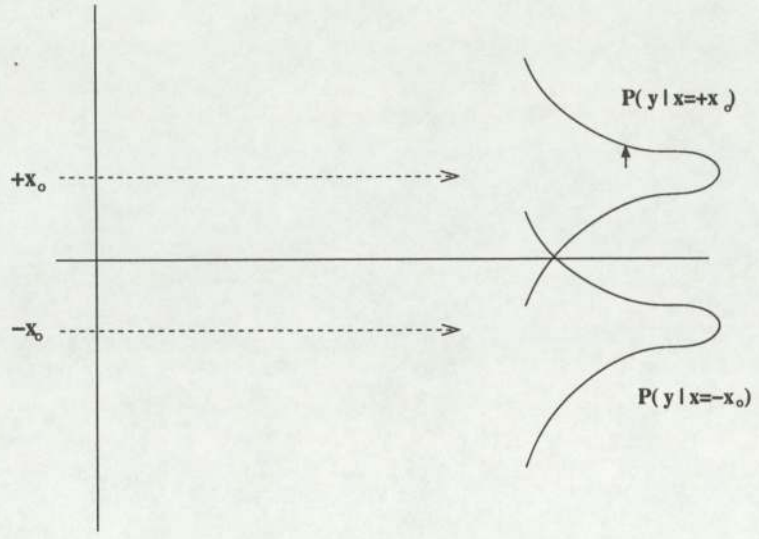
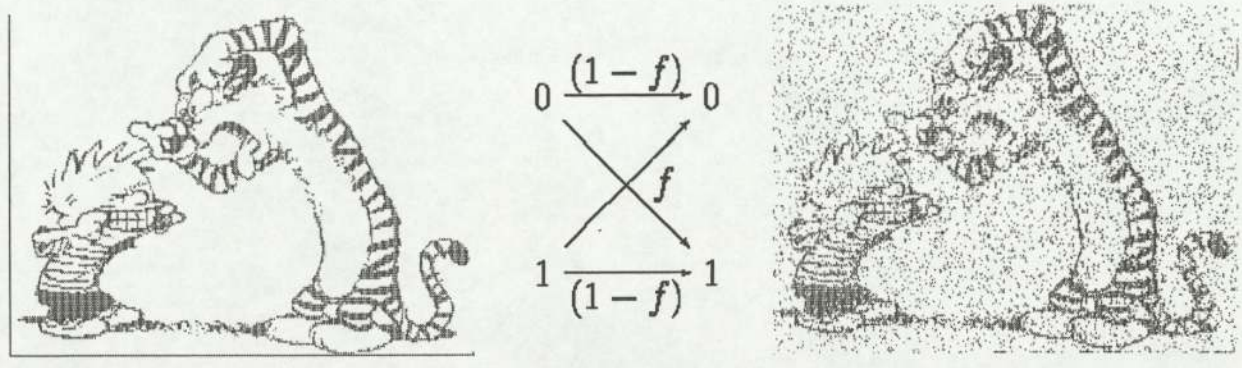


Figure 1.1: Gaussian channel.



(a) before (b) channel (c) after

Figure 1.2: Left: Original information to be transmitted. Middle: Graphical representation of the BSC, f being the probability for a bit to flip . Right: Corrupted received message.

encoding consists in multiplying the original signal $\mathbf{s}$ by G^T, giving rise to codeword $\mathbf{t}$.

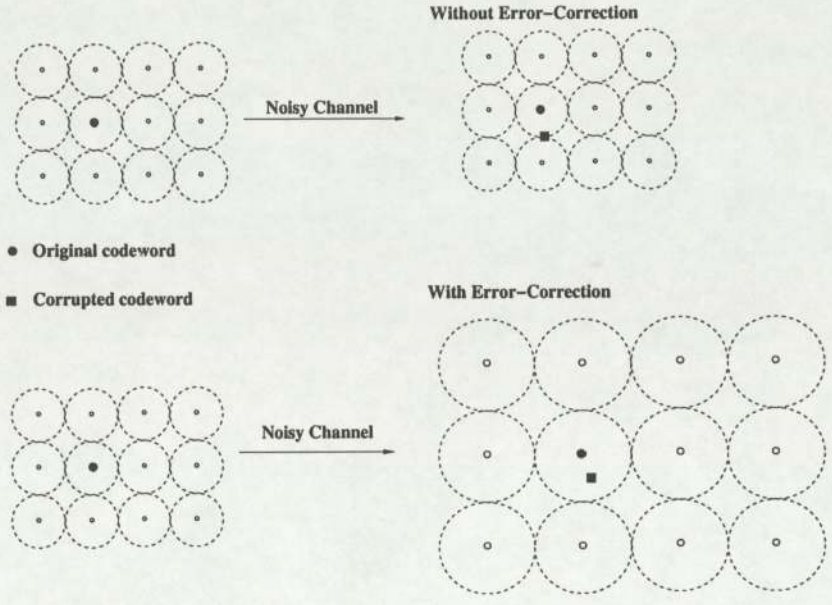


Figure 1.3: Small white circles represent possible positions of the codeword vector in a geometric space. The black one represent the actual codeword to be sent. Channel noise causes corruption which is represented by a drift. The black squares represented the received corrupted codeword . The dashed circles represent decision boundaries in the receiver. In the bottom figure, we show qualitatively the error-correction mechanism. The redundancy introduced in the transmitted information changes the space geometry, mapping it onto a higher dimensionality space and, by this way, increasing the distance between words. The same drift as in the top figure does not result here in a transmission error.

Modern information transmission (from simple hard-disks to satellite transmission) makes extensive use of error-correcting codes to compensate corruption by transmission noise.

Basically, the decoding problem of codes consists in finding the most likely source vector s in the equation $G^T\mathbf{s} + \mathbf{n} = \mathbf{r} \mod 2$ where G is a generator matrix, $\mathbf{n}$ an unknown noise vector and $\mathbf{r}$ the received vector.

In decoding the received vector $\mathbf{r}$, one makes use of H a sparse matrix (a matrix is said to be sparse if the density goes to zero when the matrix size goes to infinity) such that $G^T H = 0$, to obtain: $H\mathbf{r} = H(G^T\mathbf{s} + \mathbf{n}) = H\mathbf{n} = \mathbf{z}$ where $\mathbf{z}$ is termed the syndrome. The decoding problem consists in finding the most likely solution $\mathbf{n}$ to the

binary equation $H\mathbf{n} = \mathbf{z}$ given $\mathbf{z}$, H and the characteristics of the channel used.

1.1.3 Low Density Parity Check Codes

Low Density Parity Checks codes (LDPC codes) are specified by a parity check matrix containing mostly 0's and only small number of 1's. They belong to the family of linear codes which encode the transmitted message through a linear transformation.

A regular binary (M, J, K) LDPC code has codeword length M and a parity-check matrix with exactly J 1's in each column and K 1's in each row, assuming $J \geq 2$ and $K > J$. Accordingly, in a binary LDPC regular code, every code bit is checked by precisely J parity checks, and every parity checks involves precisely K code bits.

For the Gallager code, the parity-check matrix is a concatenation $H = [C_1|C_2]$ of two very sparse matrices, with C_2 a square matrix (size $(M - N)$) being invertible and C_1 an $(M - N, N)$ matrix. The generator matrix $G = [I|C_2^{-1}C_1](\text{mod } 2)$, where I is the identity matrix (size N).

For the MN code, the generator matrix has the form $G^T = C_n^{-1}C_s(\text{mod}2)$ where C_n is an invertible square matrix (size M) and C_s an (M, N) matrix.

As a linear block code, an LDPC code can be represented by a Tanner bipartite graph in which one set of nodes, the variable nodes, correspond to the codeword vector, and the other set of nodes, the check nodes, correspond to the set of parity-check constraints. An edge exists between a variable node n and a check node c if and only if n appears in the parity-check equation corresponding to c; in other words, if a 1 appear in the matrix H in the position (n, c). In the case of Gallager code the vector to be estimate is the noise word whereas in MN code, it is the concatenation of signal and noise.

1.2 Decoding Process

1.2.1 Message Passing Techniques

Invented in statistical physics and later in the area of computer sciences and communication, message passing techniques rely on neighbourhood influences to find a

correct assignment of states to variables under given constraints.

Here, given a graphical model, probability propagation can be used to compute the conditional probability of a message symbol given the observed syndrome. Information is exchanged between neighbouring nodes in the graph by passing messages along the edges.

Starting from a random assignment, the messages sent through each edge are updated recursively with respect to messages received in the previous iterations. Each message is associated with a variable node linked to the edge carrying the message. Invariably, the messages can be interpreted as conveying an estimate of the node's value along with some reliability information for that estimate. Associated with such a message is a hard decision: one can consider the bit's most likely value implied by the message. We will say that a message is "correct/incorrect" if its associated hard decision is "correct/incorrect", i.e., "does/doesn't" agree with the true value of the original noise bit. In addition of its good decoding performance, this algorithm has the considerable advantage of converging to a solution in polynomial time.

Chapter 2

Current Decoding Methods

In the first section,we present an overview of the belief propagation and its application for decoding corrupted codewords encoded using sparse parity-check error correcting codes. In the second section, we observe the actual performance reached by one method compared with existing decoding processes.

2.1 Belief Propagation

Our aim here is to infer the source message $\mathbf{s}$, given the received message $\mathbf{r}$ with the relation $\mathbf{r} = \mathbf{t} + \mathbf{n} = G^T\mathbf{s} + \mathbf{n}$. Which is, in the case of LDPC codes, finding the most probable solution to the equation $H\mathbf{n} = \mathbf{z}$ as explained in Section1.1.2 which will give us directly the most probable codeword $\mathbf{t}$.

Finding the most probable estimate is an NP-hard problem because of the form of the posterior $P(\mathbf{n}|\mathbf{z}) = \mathrm{P}(\mathbf{z}|\mathbf{n})\mathrm{P}(\mathbf{n})/\mathrm{P}(\mathbf{z})$ where $P(\mathbf{z}|\mathbf{n})$ equal 1 if $H\mathbf{n} = \mathbf{z}$ and 0 otherwise. So, to find the most probable $\mathbf{n}$ a posteriori (MPM), we should go over all the $\mathbf{n}$ possible. It is computationally difficult to carry out the exact calculation as it requires a sum over $O(2^N)$ terms.

Belief propagation can be efficiently used to obtain an approximate estimate. We actually assume that the prior probability distribution for $\mathbf{n}$ is factorisable : $(P(\mathbf{n}) = \prod_{\mathrm{j}} \mathrm{P}(\mathrm{n_j}))$.

The decoding process relies on estimating the marginal posterior probability $P(n_j|\mathbf{z})$ for each of the M message bits given the syndrome $\mathbf{r}$.

Given the observed syndrome, we map the problem onto a Bayesian network, which in the case of LDPC code system is a bipartite graph (Figure 2.1). The probabilistic dependencies present in the code are then represented by the graph connections.

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

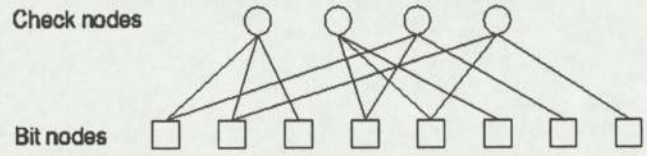


Figure 2.1: Tanner graph representation (right) of a sparse matrix (left)

We use belief propagation, an iterative algorithm based on message passing proposed by Pearl [10] to infer the approximate solution. This algorithm is based on local updates of a set of marginal probabilities and the propagation of beliefs (conditional probabilities) within the network.

The convergence of these iterations requires a tree-like structure network (which obviously means the absence of loops in undirected graphs). Indeed, on directed graphs a node will send to its children a message related to information received from its parents. In undirected graphs, a node sends to its neighbours a message directly related to what it receives from these neighbours. This direct feedback might cause a problem because a node would update its state probability on the basis of its own previous state.

To avoid this problem, we apply the *cavity rule*; the message passed by the node n to the check m is updated regarding all the checks m' other than m and vice versa for the checks. However the problem is just partially avoided because of the existence of small loops. Typically, the bipartite graph suffers from a significant number of loops, but an acceptable assumption is that the network is locally tree-like if there is no short loops (size: 4 edges) as shown in Figure 2.1.

In the particular case of the absence of short loops, Pearl's algorithm provides very good approximation. The negative effect of loops is negligible due to the network size.

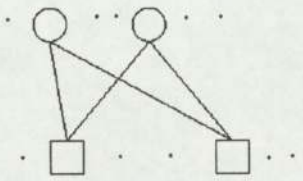


Figure 2.2: Representation of a loop in the bipartite graph (illustrated in Figure 2.1). Two bit nodes tied with the same two check nodes.

Indeed, when very sparse matrices are used, the probability for a loop in the related graph in a finite number of generations decays as γ/N, where $\gamma \sim O(1)$ [11]. When applying $N \to \infty$, the topology actually converges to a tree-like structure and BP decoding becomes exact. Moreover, for a finite systems, one can expect that a limited neighborhood of a node is tree-like.

Belief Propagation Algorithm

At each step of the algorithm we estimate the new posterior probability $P(n_i|\mathbf{z})$ of a codeword's bit to be at a certain state, given the syndrome vector $\mathbf{z}$.

To simplify things, we call $M(n)$ the set of J checks m in which bit n participates and reciprocally $N(m)$ the set of K bits n that participate in check m.

For each edge between a node n and a check m we define the following conditional probabilities(Figure 2.1) :

$$q_{mn}^{x} = P(X_n = x \mid r_{l \neq m}), \tag{2.1}$$

and

$$r_{mn}^{x} = P(Z_m \mid X_n = x \bigcap \{q_{mn'} \ : \ n' \in N(m) \setminus n\}). \tag{2.2}$$

The message q_{mn}^{x} defined in Equation 2.1 provides the probability that $X_n = x$ given all the check nodes of $M(n)$ apart from m. The message r_{mn}^{x} defined in Equation 2.2 provides the probability that the check node z_m is in his actual value given $X_n = x$. We define as well the posterior probability:

$$\hat{q_n^x} = P(X_n = x \mid r_{m \in M(n)}) \tag{2.3}$$

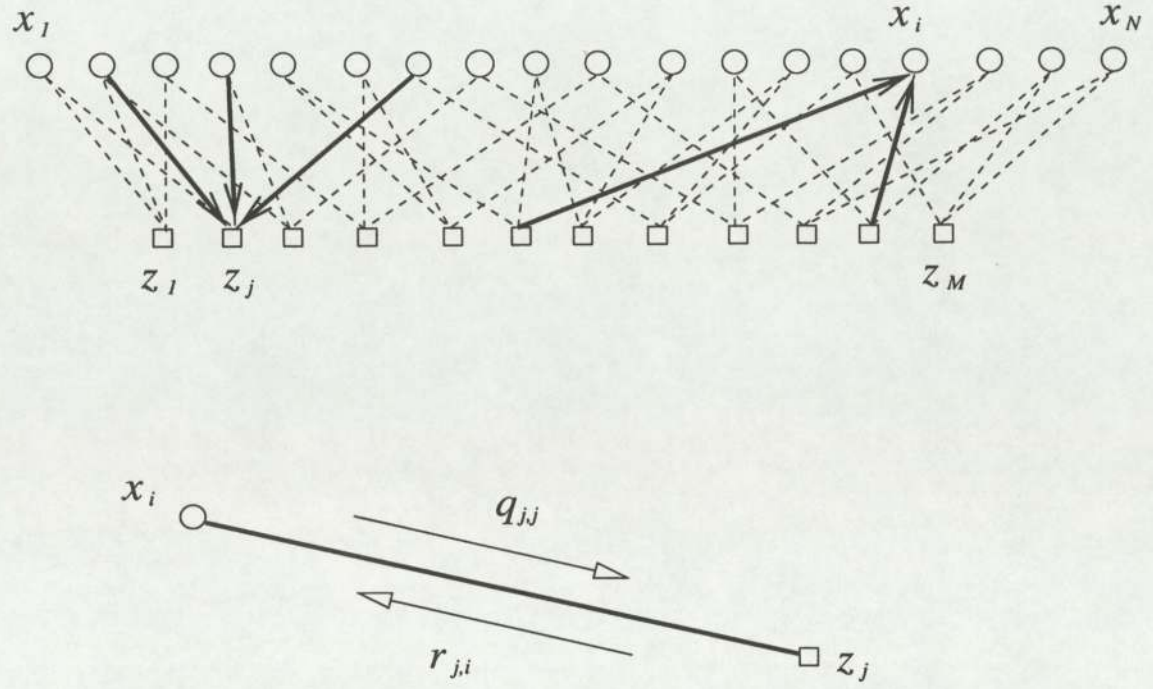


Figure 2.3: Up: The message Passing process. The arrows show the causal relationships : the state of the check z_j is determine by the sum of incoming source/noise node x_i. Down: The message going from a bit node x_i to a check node z_j is denoted $r_{j,i}$ and the opposite is denoted $q_{i,j}$

This probability define the state of the pseudo-decoded word after some iterations of the algorithm.

We will now describe in detail one iteration of the algorithm:

Initialisation Step

This sets the initial values of the q_n's to the prior value f_n (in our case the noise level f), although this initialisation has no consequence on the algorithm itself. We run this step just once at first of the algorithm.

Horizontal Step

For the bit node n, we update the associated probabilities that bit n is equal to x(x is 0 or 1 in our binary case), given the information obtained via checks other than m.

$$q_{mn}^x = \alpha_{mn} f_n^x \prod_{m' \in M(n) \backslash m} r_{m'n}^x \quad \text{(node update rule)}, \tag{2.4}$$

where α_{mn} is a normalization constant (such that $q_{mn}^0 + q_{mn}^1 = 1$) and f_n the prior.

Vertical Step

In this step, we update the associated probabilities that the check m is satisfied if bit n is equal to x and the other bits have a separable distribution given by the probabilities $q_{mn'}$: $n' \in N(m) \setminus n$.

A particularly convenient implementation of this method uses forward and backward passes in which products of the differences $\delta q_{mn} = q^1_{mn} - q^0_{mn}$ are computed. We obtain $\delta r_{mn} = r^1_{mn} - r^0_{mn}$ and the update rule becomes :

$$\delta r^x_{mn} = (-1)^{z_m} \prod_{n' \in N(m) \setminus n} \delta q^x_{mn} \quad \text{(check update rule)} \tag{2.5}$$

Finally given that $r^0_{mn} + r^1_{mn} = 1$, we obtain the updated probability of the given check equal to : $r^x_{mn} = 0.5 * (1 + (-1)^x \delta r_{mn})$

Validation Step

For each bit, we update the pseudo-posterior probability

$$\hat{q^x_n} = \alpha_n f^x_n \prod_{m \in M(n)} r^x_{mn}, \tag{2.6}$$

where α_n (such that $q^0_n + q^1_n = 1$) is a normalisation constant.

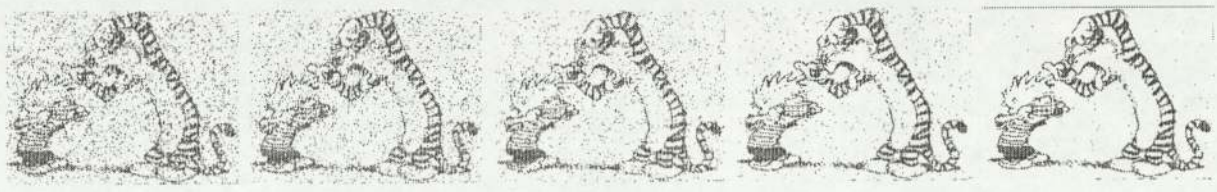
Compute then $\hat{\mathbf{x}}$ such that $\hat{x_n} = 1$ if $q^1_n > 0.5$ and 0 else. The algorithm repeats horizontal and vertical step until it converges to a stable codeword. Which means that all the parity checks are satisfied; $H\hat{\mathbf{x}} = \mathbf{z}$ then the decoding algorithm halts, and $\hat{\mathbf{x}}$ is considered as a valid codeword.

There exist two methods for running one complete loop of the algorithm. The iterative method consists in updating first all the checks and then all the nodes. The random method consists in randomly updating one node or one check among all the nodes not yet updated until all of them are updated. The random method appears to lead to a faster convergence than the iterative one because within the same iteration, updates are already propagated. This is why we make exclusive use of this method for running our set of simulations.

2.2 Limits of Optimal Decoding

Belief propagation appears to be the best decoding process in term of convergence speed (polynomial time) and performance, it is illustrated in Figure 2.4.

In this picture, the image pixels are considered as the encoded message and the received vector is corrupted over a binary symmetric flip noise density of 15%; then at each iteration the best estimate produced by an iterative probabilistic decoder is presented up to 14 iterations. At the 15th iteration, the guess violates no parity checks, and the decoder algorithm halts. The decoding is error-free.



(a) Iteration 1 (b) ...2... (c) ...7... (d) ...11... (e) ...14...

Figure 2.4: Belief propagation applied to a codeword transmitted through a BSC with a noise level $f = 0.15$. After a few iterations we observe a convergence to the correct solution (the original picture).

The algorithm would produce the exact posterior probabilities of all bits if the bipartite graph defined by H contained no cycles which is atypical in reality. The performance of such a decoding scheme is bounded in term of noise level as presented and studied in further sections.

2.2.1 Shannon's Bound

Shannon proved a remarkable result [12], that for any channel, there exists a certain code rate C (the so-called *Shannon capacity* of the channel) below which codes capable of achieving perfect decoding exist and above which such codes can not be found. As the proof was not constructive, we know that codes exist but we're unable to construct them. A message encoded at rate R (message information content/codeword length) up to the channel capacity C can be decoded with a probability of error that decays exponentially with the message length.

This theoretically achievable rate is equal, in the case of the BSC channel, to:

$$C = 1 - H_2(f), \tag{2.7}$$

where

$$H_2(p) = -p \cdot log_2(p) - (1-p) \cdot log_2(1-p) \tag{2.8}$$

In term of noise, for any channel and a given error correcting code rate R, there exists a limit noise level above which it is impossible to achieve perfect decoding. And a code which saturates Shannon's bound represents the most efficient way of transmitting data as the length of the transmitted message is the smallest possible for retrieving perfectly the original data up to the given noise level f_S. In the more general case of biased messages ($P(n_i = 1) = f_s, \forall i$) and allowing a decoding bit error probability p_b, the maximal code rate R_c, for a given flip rate noise f, is given by:

$$R_c = \frac{H_2(f_s) \cdot (1 - H_2(f))}{(1 - H_2(p_b))} \tag{2.9}$$

The challenge is to create error-correcting codes that get closer as possible to what Shannon proved is possible.

2.2.2 Dynamical Transition

So far, some of the most efficient existing error-correcting codes, among which are LDPC codes, manage to achieve excellent results up to a certain noise limit (obviously below the Shannon's bound). This limit is called the dynamical transition in statistical physics. A second transition is termed thermo-dynamical or critical transition point f_C, defining the theoretically noise limit that still allows perfect retrieval for a given code. In the case of code of rate $R = 1/4$ this dynamical transition is situated at around $f = 0.16$ when the critical limit which i s close to Shannon limit $f_c = 0.21$. Reach the dynamical transition is then already a very good result with respect of other error-correcting codes. The challenge of this project is to push practical decoding scheme beyond the dynamical transition towards Shannon's limit.

In order to understand what is happening, we describe the problem schematically (Figure 2.5). One can notice that, up to the dynamical transition, there exists only one cluster of solutions where the BP algorithm aims to find the single correct solution

noise vector, whereas between the dynamical transition and the critical transition there exists many clusters of potential solutions (all satisfying the parity check equations) among which the BP aims to find the correct one. The various clusters are disconnected what makes it difficult to move between clusters by local modifications in search of the true solution among many suboptimal not stable solutions. As the graph is very large, it is highly probable that in one part of the graph, BP is induced to converge to one solution while in other parts it converges to another solution. These different parts being more or less related to each other, each one will try to "convince" the other to opt for its solution. This explains why the algorithm generally doesn't converge in this area. Beyond the critical transition point, one cannot find the true solution even theoretically.

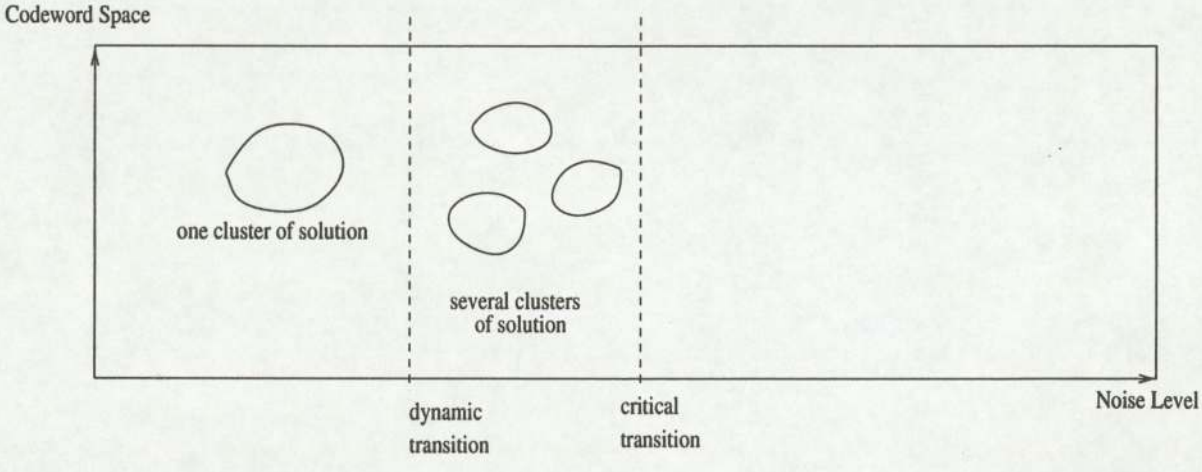


Figure 2.5: Representation of the solutions to the equations $H \cdot \mathbf{n} = \mathbf{z}$, given the prior. Up to the dynamical transition: only one cluster of solutions. Between the dynamical and the critical transition: many cluster of solutions among witch the correct and several suboptimal incorrect ones. Above the critical transition: no solution to the problem.

Performance

The simulation results in Figures 2.6 and 2.7 are based on decoding of biased codewords length $M = 1000$. Each point represent the average error on 1000 trials with belief propagation as a function of the noise level. Error is represented as the overlap between the original noise vector and the decoded one (overlap of 1 means perfectly correct decoding). We can see on Figure 2.6 that error-free decoding is obtained for

100% cases up to a certain noise level (dynamical transition) and from then, error starts increasing. On Figure 2.7, we focus on the area between the dynamical and critical transition.

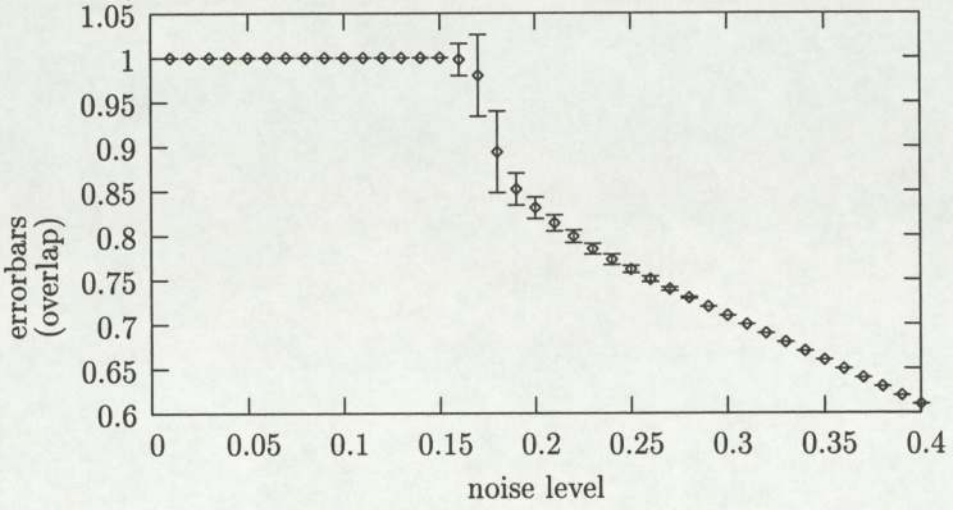


Figure 2.6: Overlap between the actual noise vector n and the decoded one $\hat{n}$ for an LDPC code with $K = 4$ and $C = 3$ (therefore $R = 1/4$). Averages of 200 simulations of BP for a code word length M=1000. Vertical rows represent standard deviations.

A suboptimal solution obtained for one of these codes is represented as a histogram in Figure 2.8 of the cavity-magnetisations distribution (also called ferromagnetic solution) gauged by the initial code word we aim to find out. The points with negative cavity field values represent the wrong magnetisations with respect to the original codeword and positive points represent the correct ones. As we can see the proportion of wrongly decoded bits is much lower than the correct ones.

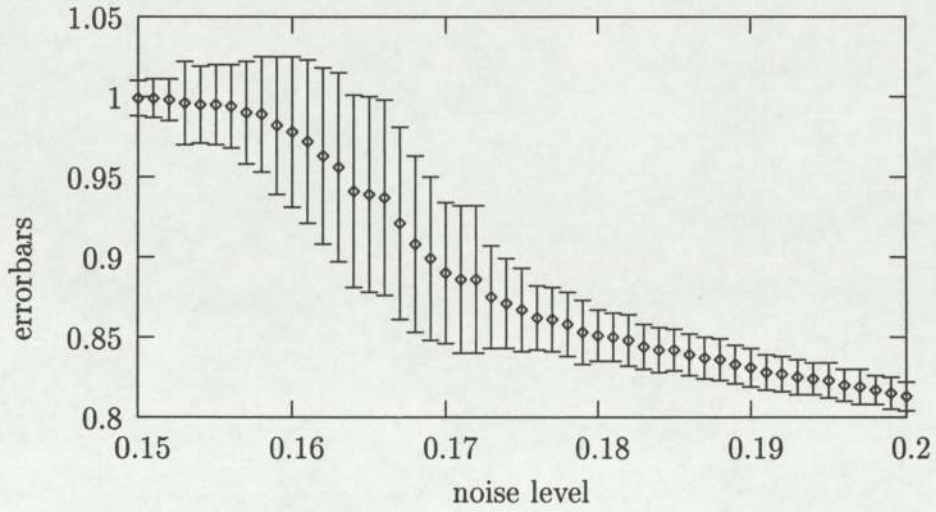


Figure 2.7: Zoom in the previous figure on the area of non convergence between the dynamical and critical transition

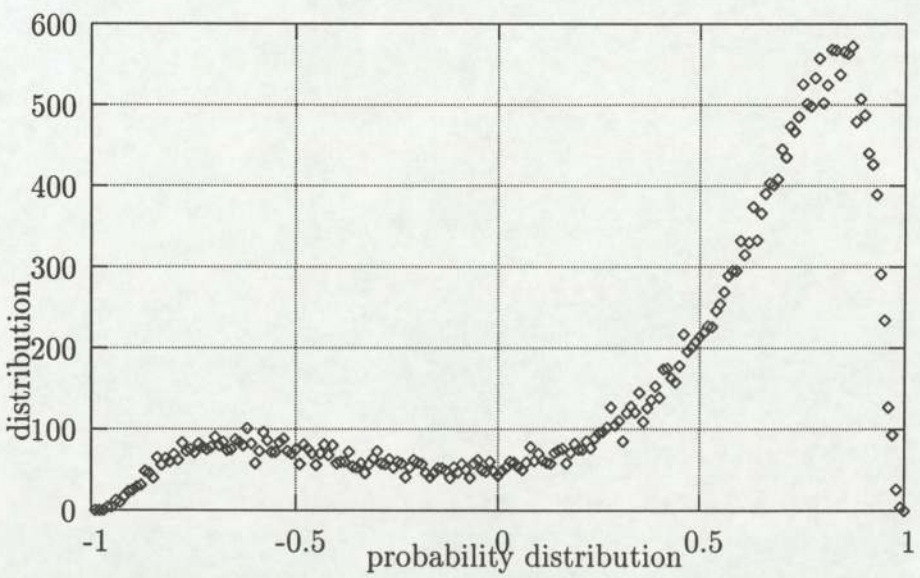


Figure 2.8: Probability distribution histogram for suboptimal solution obtained for $f = 0.2$. Parameters are N=1000 R=1/4. Circles correspond to an experimental histogram obtained by decoding with BP over 1000 random graph constructions.

Chapter 3

Improvement to Message Passing

3.1 Free Energy Minimisation

As mentioned earlier (Section 2.2.2), a major problem occurring during the BP is that convergence, for some reason, is not always obtained.

A bit of Statistical Physics

To understand the concept of free-energy, one can consider a system of N particles, each of which can take one of a discrete number of states, where the states of the ith particle are labelled by x_i. For example, the atoms in a magnetic crystal have their states characterised by the spin "up" or "down". The overall state of the system will be denoted by the state vector $\mathbf{X} = x_1, ..., x_N$. At each state of the system corresponds a certain energy $E(X)$. A fundamental result of statistical mechanics is that, in thermal equilibrium, the probability of a state will be given by Boltzmann Law

$$p(X) = \frac{1}{Z(T)} e^{-E(X)/T}, \tag{3.1}$$

where T is the temperature, and Z(T) is simply a normalisation constant, known as the partition function:

$$Z(T) = \sum_{X \in S} e^{-E(X)/T}, \tag{3.2}$$

where S is the space of all possible states X of the system.

For the case of a factor graph probability (generalisation of the Tanner graph seen in Section 2.1) distribution function (see Figure3.1), we suppose that $p(\mathbf{X})$ factors into

a product of functions: $p(\mathbf{X}) = (1/Z)\prod_{a=1}^{M} f_a(x_a)$. Here a is an index labeling M functions $f_A, f_B, .., f_M$, where $f_a(x_a)$ has arguments x_a that are some subset of $x_1, ..., x_N$. The factor graph is a bipartite graph that expresses the factorisation structure; it has a variable node per variable x_i and a factor node per function f_a, with an edge connecting variable node i and factor node a if and only if x_i is an argument of f_a.

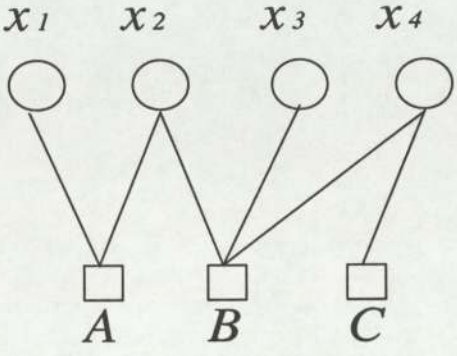


Figure 3.1: A factor graph representing the joint probability distribution $p(x_1, x_2, x_3, x_4) = \frac{1}{Z} f_a(x_1, x_2) f_B(x_1, x_2, x_3) f_C(x_4)$.

We define the energy $E(X)$ of a state $\mathbf{X}$ to be:

$$E(X) = -\sum_{a=1}^{M} ln f_a(x_a). \tag{3.3}$$

The Helmholtz free energy of a system is

$$F_{Helmholtz} = -T \ln Z = -\ln Z \quad \text{(because generally T is set to 1)}. \tag{3.4}$$

This free energy is a fundamentally important quantity in statistical mechanics, because if one calculates the functional dependence of $F_{Herlmotz}$ on quantities like a macroscopic magnetisation field H or temperature T, then it is easy to compute experimentally measurable quantities like the response of the system to a variation of H or T.

So far, we have described all the coding and decoding process via the Boolean (0,1) representation. It is convenient to introduce binary variables ± 1 in order to apply methods of statistical physics.

The connection between spin systems and error correcting codes, first noted by Sourlas [13] in 1989, is based on the existence of a simple isomorphism between the

additive Boolean group$(\{0,1\},\oplus)$ and the multiplicative binary group$(\{+1,-1\},\cdot)$ defined by:

$$S \cdot X = (-1)^{s \oplus x}, \tag{3.5}$$

where $S, X \in \{+1,-1\}$ and $s, x \in \{0,1\}$. A parity check bit in a linear code is formed by a Boolean sum of K bits of the form $\bigoplus_{j=1}^{K} s_j$ that can be mapped onto a K-spin coupling $\prod_{j=1}^{K} S_j$.

This new representation is worth using because of the compactness of the corresponding equations. For example, one can describe the conditional probabilities standing for the transmission through a BSC in a simple manner as

$$P(r|t) = \frac{1+\rho rt}{2} = \frac{\exp[\beta_n rt]}{2\cosh(\beta_n)}, \tag{3.6}$$

where t and r $(\in \{-1,+1\})$ are the transmitted and received message bits respectively, f the flip probability also known as noise level of the channel and $\rho = 1-2f$ and $\beta_n = (\frac{1}{2})ln[\frac{(1-f)}{f}]$. ($\beta_n$ represents the inverse of the temperature in physical representation.)

The second advantage of this binary representation is that it makes the similarity to Ising spin models enabling one to take advantage of the techniques developed in statistical physics. By representing the free-energy of the system and trying to minimise it, one can obtain the update equations for belief propagation.

3.2 Improvements

3.2.1 Prior Biasing

The problem of non-convergence may occur because of a conflict between the prior constraint over the algorithm and the parity checks satisfaction constraint. Up to the dynamic transition, this two constraints contribute jointly to reach the correct solution. The *prior* (assumption on the noise word density), in our case derived from the characteristic of the BSC, represents the constraint of maximising the codeword overlap with the true solution. In statistical physical view of the problem, it represents a *temperature* of our system and appear then to be crucial in the BP processing.

The idea is to find out empirically the influence of this *prior* by biasing it towards a higher magnetisation during BP iterations providing global information may facilitate the choice of a better solution. Decreasing this *prior* will apply a higher constraint over the algorithm whereas increasing it will relax this constraint (increase the *temperature* makes the system more free to variate).

3.2.2 Time Averaging

The time averaging is a very powerful tool proved to reach very good results in many fields (non linear oscillations, stability analysis among many others). The idea here is that, if the algorithm does not converge, to compute the message sent over a certain number of iterations, a *time window*, and then compute the average messages. Choosing the *time window* too long could slow the algorithm without gaining in accuracy, while too narrow *windows* would bias the averaging by the short oscillations influence. We then apply *sequential fixing* to the system. Basically, it consists in fixing the states of the nodes which, we suppose, have reached their final state. To do so we consider the variables which state probability is above a certain *threshold* and fix them for the following of the BP. No need to say that the choice of the *threshold* is essential.

Complete explanation of the algorithm can be found in Appendix B

3.2.3 Expectations

We expect the prior-biasing to give the right direction to belief propagation. This technique should be applied when it doesn't happen to converge. The time averaging combined to sequential fixing should then lead the algorithm to converge to the exact solution although it might slow the convergence speed.

As we can see in the Figure 3.2, fixing a small proportion of bits in the codewords to their exact state gives rise to an attraction of the remainder of the bits to their final exact state.

Using these ideas, we will endeavour in further sections to find empirically the most efficient way to combine them in order to improve the performance of standard BP.

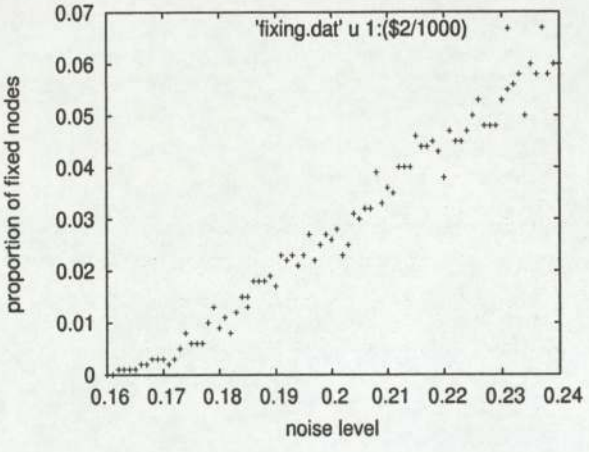


Figure 3.2: Proportion of correctly fixed bits (before running the BP) necessary to reach the correct solutions in 100% cases, with respect of the noise level. These numbers of necessary fixed bits have been computed by averaging the values over 1000 randomly constructed graphs.

Chapter 4

Experiments and Results

In this chapter, we describe experiments carried out as part of this project and the results obtained. In the first section, we present the implementation itself and in the second one, an overview of the different results reached along the simulations, these will be interpreted in the final section.

4.1 Implementation

4.1.1 Random Graph Generation

To get a correct study of LDPC codes, one needs to generate a high number of large and sparse matrices to carry out experiments on. We use generate randomly low density matrices with uniform weight per column and per row, but we need also to avoid matrices with short loops (explained in Section 2.1) which means no two columns have an overlap greater than 1. In term of bipartite graph, if two nodes are tied to the same pair of checks, then one should allocate to them other checks to remove the short loop. Exact explanation of the algorithm used can be found in Appendix C.

4.1.2 Belief Propagation Implementation

All the following simulations results are based on 1000 trials, and the obtained data are processed to obtain statistical values.

The output provided is the overlap between the original and decoded message which

allows to compute the total bit error mean and the error-bar which represents the variance obtained from the simulations results. To determine the convergence speed, we monitor the number of iterations. To know whether the algorithm has converged or not, we fix a maximum number of iterations.

The exact update rules used in the algorithms can be found in Appendix B.

4.2 Simulations

For each experiment, we generate a random noise vector $\mathbf{n}$, and we compute the syndrome vector $\mathbf{z}$ according to the sparse matrix H such that $\mathbf{z} = H\mathbf{n}$.

We generate random noise vectors such that their density corresponds to the prior probability f (probability for the word to have a unit element at a specific bit). If we were to apply the flip probability to each of the bits independently, we would here introduce variance in the number of unit elements per vector. Instead, we compute the exact number of unit elements we need in the noise vector and allocate them randomly. We can then be sure that we get a noise vector with a correct probability, used later in the decoding process and that no variance is introduced in the results. In order to make the experiments more relevant, the generation of noise vectors and sparse matrices use distinct random number generators.

4.2.1 Prior Biasing

The prior used all along the belief propagation process is biased to a lower corruption rate as explained in Section 3.2.1. We assign it $f_\alpha = \alpha f$. The biasing coefficient α has been determined empirically by minimising the error mean during a training part. Two methods have been implemented. The first was to fix α to a certain value during the entire decoding process. The second was to decrease it monotonically during the decoding process starting from a high value and decreasing dynamically at each iteration (for example from $\alpha = 2$ to 1 decreasing of 0.01 after each iteration). The most efficient method in term of results appears to be the first one. This is the one we use in the reported simulations.

As we can notice on the graphs (Figure 4.1 and 4.2), biasing the prior does not

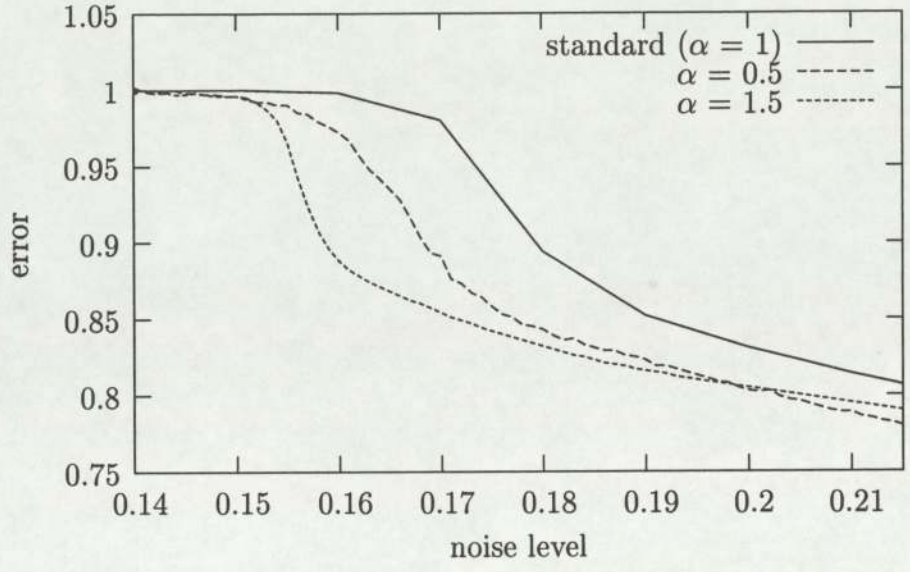


Figure 4.1: Biasing the prior throughout the BP decoding by a coefficient α (here we represent $\alpha = 0.5$ and $\alpha = 1.5$).

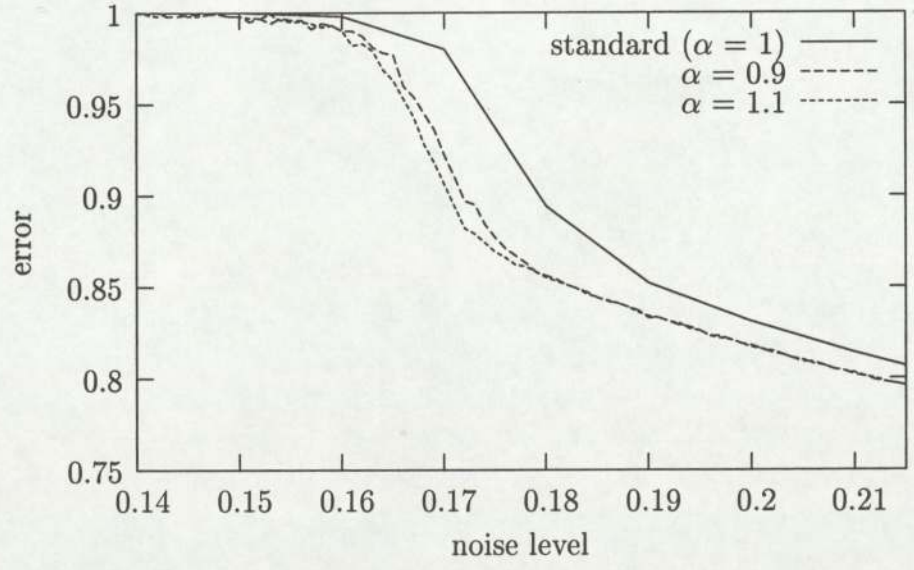


Figure 4.2: Biasing the prior throughout the BP decoding by a coefficient α close to 1 (here we represent $\alpha = 0.9$ and $\alpha = 1.1$).

really over-perform the standard unbiased BP decoding. But another effect is worth studying. To have a better idea about this effect, we represent the cavity magnetisations probability distribution (as explained in Section 2.2) for a noise level close enough to the critical transition to be in the non-convergence area. We took $f = 0.17$ and $f = 0.2$and tried several values for α the prior biasing coefficient for the prior (Figures 4.3 and 4.4). Notice that for $\alpha = 1$ the prior f is equal to the original flip rate for the BSC channel.

On these histograms , probabilities are gauged by the original codeword so that we get probabilities that corresponds to the correct estimate on the positive side and wrong estimates on the negative side of the abscissae axis. One notices that for $\alpha < 1$ the probabilities tend to reach 0.5 whereas for $\alpha < 1$ they tend to go towards 0 or 1. The latter effect is the most interesting one because, as we will see in the next section we need the state probabilities to be high enough to be sure of taking the right decision while fix the nodes value to a certain state.

4.2.2 Time Averaging

Time-averaging, introduced in Section 3.2.2 is a variant of belief propagation which takes place when the algorithm doesn't converge to a solution. The *time window* has been fixed empirically to 40 loops .

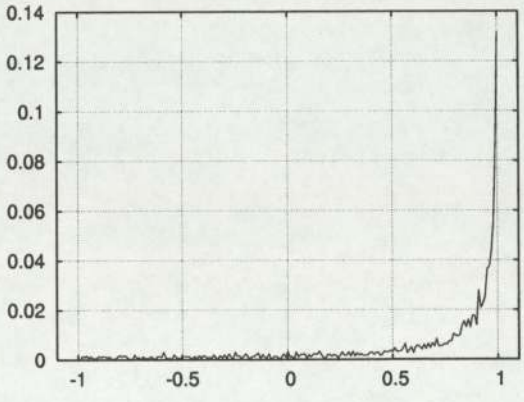
4.2.3 Mixing Methods

The best results have been achieved by using both methods. Biasing the prior throughout the BP iterations combiened with time averaging.

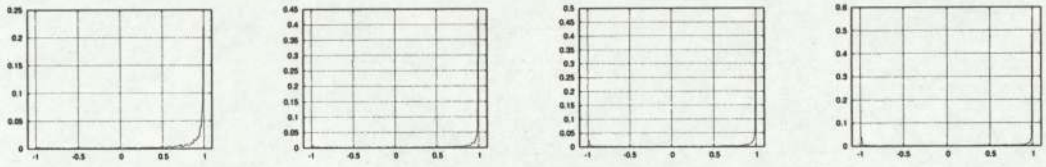
As we can see from this graph (Figure 4.6), the performance of the modified BP over-perform that of standard BP decoding. The transition point is pushed beyond the dynamical transition (in this case 0.16) to a little more 0.17. The critical point in this case is approximately 0.21.

4.3 Interpretations of the results

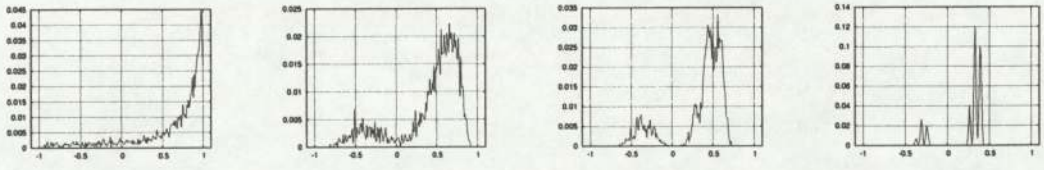
We would expect to get better decoding results, and get closer to the critical transition noise level by applying these methods, given the results illustrated in Figure 3.2,



(a) $\alpha = 1.0$ (unbiased)

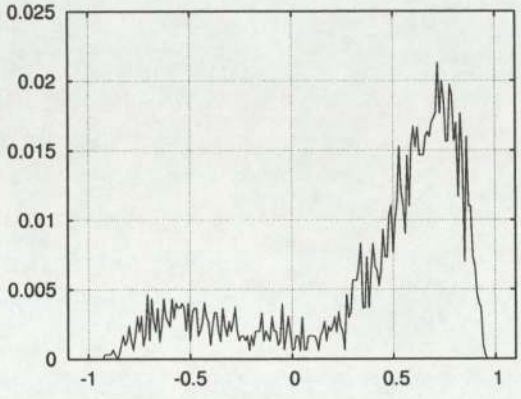


(b) $\alpha = 0.7$ (c) $\alpha = 0.5$ (d) $\alpha = 0.3$ (e) $\alpha = 0.1$

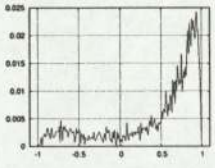


(f) $\alpha = 1.2$ (g) $\alpha = 1.5$ (h) $\alpha = 1.7$ (i) $\alpha = 2.0$

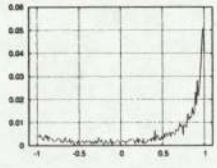
Figure 4.3: Histograms for different values of the prior biasing coefficient α for a noise level $f = 0.17$ (above the dynamical transition). The probability distribution is gauged by the original codeword so that we get probabilities that corresponds to the correct estimate on the positive side and wrong estimates on the negative side of the abscissae axis.



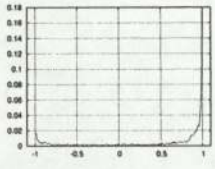
(a) $\alpha = 1.0$ (unbiased)



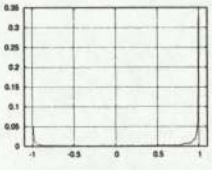
(b) $\alpha = 0.7$



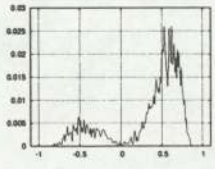
(c) $\alpha = 0.5$



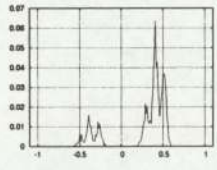
(d) $\alpha = 0.3$



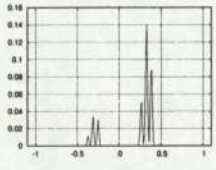
(e) $\alpha = 0.1$



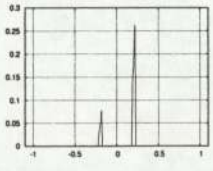
(f) $\alpha = 1.2$



(g) $\alpha = 1.5$



(h) $\alpha = 1.7$



(i) $\alpha = 2.0$

Figure 4.4: Probability distribution histograms for different values of the prior biasing coefficient α for a noise level $f = 0.2$.

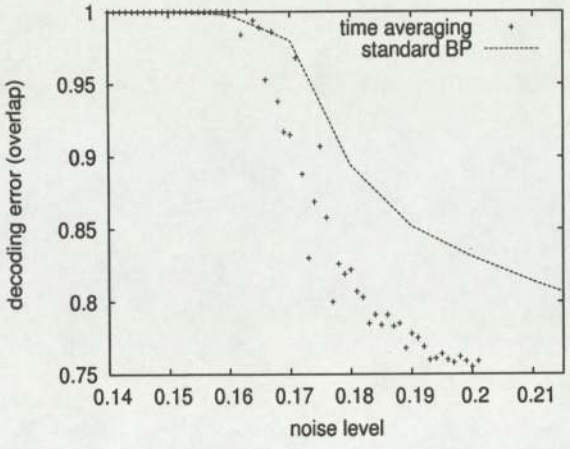


Figure 4.5: BP decoding using time averaging. The *threshold* for fixing nodes values after time averaging has been computed empirically and set to 0.9.

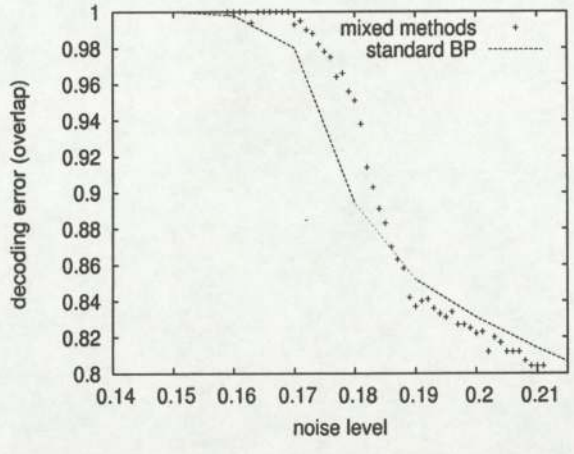


Figure 4.6: Biasing the prior throughout the BP decoding in conjunction with time averaging as well. The biasing coefficient α has been fixed empirically at 0.95 and the *threshold* for *sequential fixing* of nodes values after time averaging is set to 0.9.

where we can notice that the fact of fixing correctly a small proportion of nodes lead the BP decoding to the correct codeword. The results so far show a modest improvement in decoding performance; however, obtaining an improvement on the way between the dynamical and critical transition points is already an encouraging achievement.

Time averaging on this problem didn't prove to be as efficient as for other hard computational problem such as graph colouring [3] where it permitted BP algorithm to nearly saturate the critical transition. One explanation may be that in other problems such as graph colouring, there exist many correct solutions, whereas for error correcting codes, only one true solution exists. This means that during the *sequential fixing*, only one wrong fixed bit lead to a complete failure.

Nevertheless, prior biasing helped in reducing fixing problem, although it might slow the convergence of the algorithm by leaving the system more free to fluctuate.

Chapter 5

Parallelisation on a Cluster

Traditionally, software has been written for serial computation: to be executed by a single Central Processing Unit and through a series of instructions, executed one after the other. In its simplest sense, parallel programming is the simultaneous use of multiple compute resources to solve a computational problem in order to save *wall clock* (computer clock) time or solve larger problems.

By its nature, the BP algorithm, particularly for very large LDPC codes, seems to be very suitable for a parallel implementation. In other words, it could be broken apart into discrete pieces of work that can be solved simultaneously. In the first section, we present the clusters technology in general and the NCRG facilities in particular. In the second section, we study how belief propagation can be parallelised and applied therefore to the cluster computational capacity.

5.1 What is a Cluster

5.1.1 Description

A cluster is an ensemble of off-the-shelf computers integrated by an interconnection network and operating within a single administrative domain. It consists of separating computers with no physical shared memory. Processes on different nodes need to communicate to coordinate computation and transfer data.

5.1.2 NCRG Cluster Configuration

The NCRG department has two clusters , one of 24 2-Pentium nodes and a smaller one of 7 2-AMD processor nodes. We will focus on the former as it is the one reserved to C programming. It is composed of one master node and 23 computing nodes.

We use then the message passing model which is characterised by the fact that each node use its own local memory during computation and then node exchange data through communication by sending and receiving messages. This data transfer usually requires cooperated operations to be performed by the nodes , for example: a send operation must have a matching receive operation.

From a programming perspective, message-passing implementations commonly comprise a library of subroutines that exist as source code and the programmer has the responsibility for determining parallelism operations. We commonly use the MPI (Message Passing Interface) implemented on our cluster, for message passing programming.

5.2 Belief Propagation Parallelisation

BP decoding is an algorithm based on message passing, as introduced in Section 1.2. It is well-known that as the noise level in a channel increases, the decoding time (measured in term of algorithm iterations) also increases. As the noise f approaches the critic transition f_c, this decoding time diverges ($t \propto 1/(f_c - f)$ [5]).

In this section the technical details of our simulations are described.

5.2.1 Implementation

As we can see in the implementation (as described in Section 2.1), this decoding process consists in a serial of nodes state updates. For different nodes (either bit or check) the state probability update operations are independent within a single iteration of the decoding algorithm. The main principle of the parallelisation is the sharing of graph nodes among the cluster processors so that each one deals with a certain number of checks and bits, as illustrated in Figure 5.1.

At each iteration, one single processor will update the checks and bits probabilities

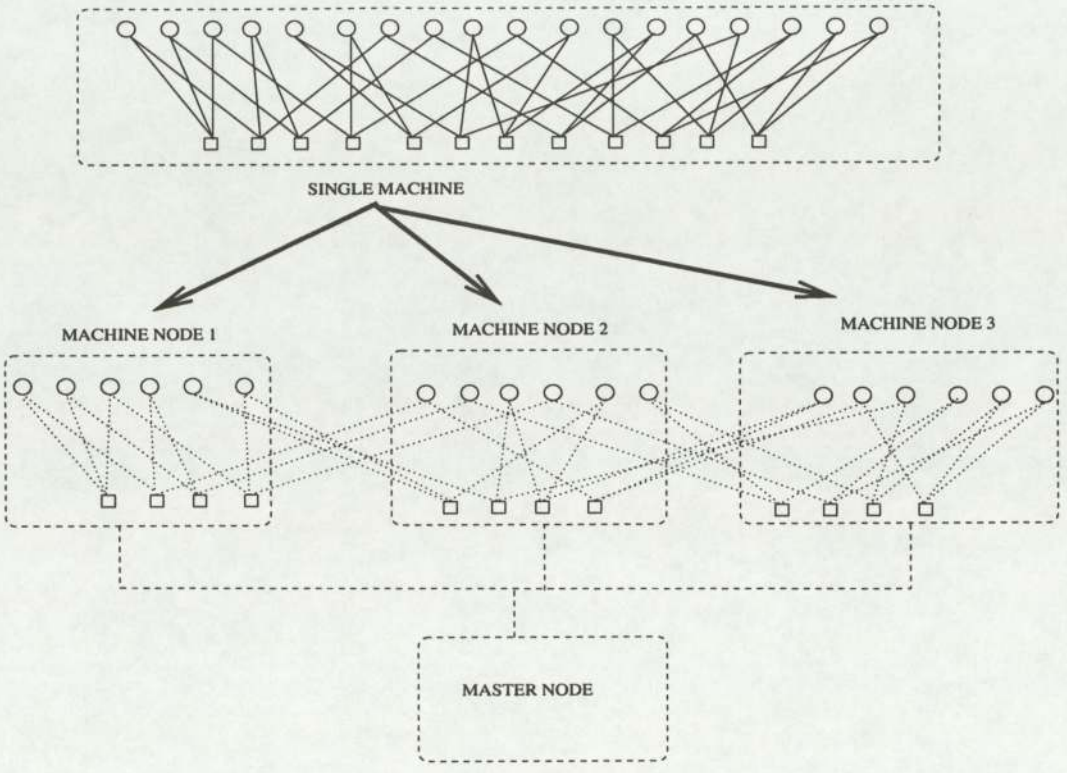


Figure 5.1: Graph nodes divided and distributed among the cluster computing nodes to have a balanced distribution of tasks .

which it is in charge of. After all the processors have completed their iteration, they communicate the values they computed to all processors, so that for the next iteration, they all have the same probabilities for the checks and nodes to be at a certain state. By doing so, since the complexity per iteration is divided by the number of running processors, the computational time should be divided by the number of cluster nodes.

5.2.2 Speed Performance

In both cases (single machine and in parallel implementation) the flip rate f (noise level) was selected as being close enough to the dynamical transition for this code rate such that the decoding is characterised by relatively long convergence times. We ran the simulations over 200 different code instances and computed the average *wall clock* decoding time in seconds.

We made the choice of fixing the code rate in order to have an accurate comparison between different codeword sizes. The noise level f has been chosen for the simulations as 0.18 .

Code Size	Rate	Standard BP decoding time (sec.)	Parallelised BP decoding time
1000	1/4	5.702960e-01	7.668980e+00
2000	1/4	1.015597e+00	1.476667e+01
4000	1/4	2.116086e+00	2.318081e+01
8000	1/4	4.274886e+00	5.882426e+01
16000	1/4	8.532149e+00	1.074536e+02
32000	1/4	1.698316e+01	1.711698e+02
64000	1/4	3.373872e+01	4.329622e+02
128000	1/4	6.843001e+01	7.638526e+02
256000	1/4	1.363311e+02	2.370557e+03
512000	1/4	2.732913e+02	6.884423e+03

Figure 5.2: Values obtained for BP decoding time for a fixed rate and by varying the code size

We demonstrated (Table 5.2) that the parallel implementation, although expected to be faster in theory, is much slower than the single machine running one. This for two reasons. The first reason is simply the high speed of the single machine for decoding. Second is the time loss related to the inter communication required between the cluster node during information transfer. Indeed after each iteration of the algorithm, each process broadcasts all the values it has charged and updated during the iteration. This data transmission are considerable with respect of the decoding computing time. This data broadcasting time has been computed and proved to be proportional to the data size.

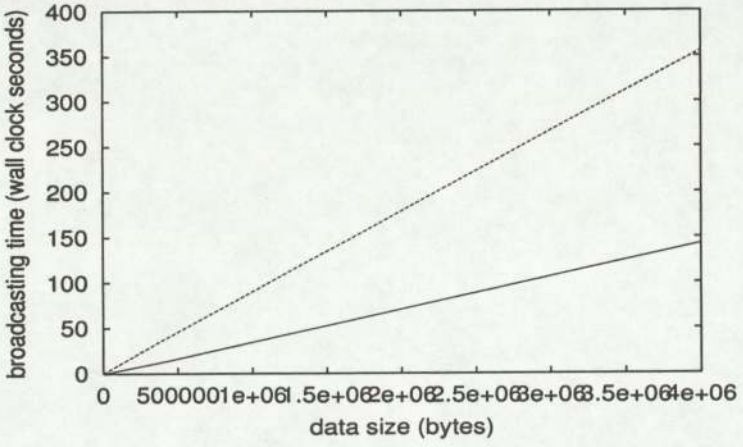


Figure 5.3: Time spent for sending data measured as number of bytes from the master node to all the computing nodes. The upper graph has been computed by sending data to 10 nodes. The lower line is for broadcasting data to 4 nodes.

Chapter 6

Conclusions

In this thesis, we carried out an empirical study on the effect of new techniques inspired by statistical physics principles on belief propagation. We first implemented the standard belief propagation applied to LDPC decoding problem and checked that its performance was bounded in term of noise level given the channel model.

We studied then how to apply a variant of this belief propagation called time averaging, as this variant appeared to achieve very good performance in other graph problems such as graph colouring. This technique on itself didn't prove to be efficient because of the precision needed in error-correcting code problem. We combined this technique with an other variant of BP called prior biasing which consists in biasing the prior by a certain coefficient throughout the decoding process. This combination appeared to reach quite good results and over-perform standard belief propagation.

It would be interesting to study the effect of this new variant of BP decoding over other types of error-correcting codes such as irregular codes or MN codes. In this project, we always considered the BP with its standard update rules. It would be interesting in an other hand to re-think this rules in term of statistical physics by taking in consideration the fact that the *temperature* needs to get lower while derivating these rules.

We have undertaken then to implement a parallelisation of the BP decoding algorithm in term of programming. What did not really appear to be efficient in term of speed because of inter-communication time loss between the computer nodes.

And finally, one can tell that as long as the theoretical bound in term of noise level is not achieved, there is still room for improvement and some new decoding algorithm or even type of error-correcting code are to be found. The fact that we can go above the dynamical transition gives good hope in this sense.

Bibliography

[1] S. Chung, T. J. Richardson, and R. L. Urbanke. Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation. 2001.

[2] M. C. Davey. *Error-correction using Low-Density Parity-Check Codes.* PhD thesis, University of Cambridge, 1999.

[3] B. Fabre. Algorithm for colouring random graphs. Master's thesis, Aston University, 2003.

[4] R. G. Gallager. *Low Density Parity Check Codes.* Number 21 in Research monograph series. MIT Press, Cambridge, Mass., 1963.

[5] I. Kanter and D. Saad. Error-correcting codes that nearly saturate Shannon's bound. *Physics Review Letters*, 83(13):2660–2663, 1999.

[6] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Improved low-density parity-check codes using irregular graphs and belief propagation. Submitted to ISIT98, 1998.

[7] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. In *Proceedings of 1997 IEEE International Symposium on Information Theory. Ulm, Germany.*, page 113, 1997.

[8] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In Colin Boyd, editor, *Cryptography and Coding. 5th IMA Conference*, number 1025 in Lecture Notes in Computer Science, pages 100–111. Springer, Berlin, 1995.

[9] D. J. C. MacKay and R. M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18):1645–1646, August 1996.

[10] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, 1988.

[11] T. Richardson and R. Urbanke. The capacity of low-density parity check codes under message-passing decoding. *IEEE Transactions on Information Theory,* 47(2):599–618, 2001.

[12] C. Shannon. *Mathematical Theory of Communication.* Number 21 in Research monograph series. Bell Sys. Tech., Cambridge, Mass., 1948.

[13] N. Sourlas. Spin-glass models as error-correcting codes. *Nature,* 339:693–695, 1989.

[14] R. Vicente, D. Saad, and Y. Kabashima. *Low Density Parity Check Codes - A statistical physics perspective.* PhD thesis, Aston University, 2002.

[15] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical report, Mitsubishi, 2002. MERL TR-2002-??

Index

Bayesian network, 14
belief propagation, 12, 13, 15, 31, 37, 48
Binary Symmetric Channel, 8
bipartite graph, 11, 14, 24
    Tanner graph, 23

cavity rule, 14
cluster, 36
critical transition, 19

dynamical transition, 19

error-correcting codes, 8

free energy, 23

Gallager code, 11
Gaussian Channel, 8

information theory, 7

Low Density Parity Check Codes, 8, 11
    sparse matrix, 14, 53

message passing, 11, 15, 48
MN code, 11

parallelisation, 36, 38
prior, 29
    prior biasing, 25, 29
prior biasing, *see* prior

sequential fixing, 26, 35, 52
Shannon's limit, 18
short loop, 14, 28
sparse graph, 28, 53
sparse matrix, *see* Low Density Parity Check Codes

Tanner graph, *see* bipartite graph
time averaging, 26, 31, 51
    time window, 31

Appendix A

Notations Used

- **BP** : Belief Propagation algorihtm.
- **BSC** : Binary Symmetric Channel.
- p_b: **bit error** The probability that a code message bit differs from the original transmitted one after decoding. The difference is determined in term of overlap between the retrieved codeword and the transmitted one.
- *BER*: **block error** The probability that the decoded message differs from the original one after decoding.
- **error-bar** Variance among the computed error obtained throughout the process for fixed values
- $M(n)$: set of J checks m in which bit n participates.
- $N(m)$: set of K bits n that participate in check m.
- $M(n) \setminus m$: $M(n)$ apart from n.
- $N(m) \setminus n$: $N(m)$ apart from m.
- $q_n^0 = P(X_n = 0 \mid r_l)$ represents the pseudo-posterior probability that bit X_n is 0, given the information obtained via all the checks.
- $q_{mn}^0 = P(X_n = 0 \mid r_{l \neq m})$ represents the probability that bit X_n is 0, given the information obtained via checks others than check m.

- $r^0_{mn} = P(Z_m \mid X_n = 0 \bigcap \{q_{mn'} : n' \in N(m) \setminus n\})$ represents the probability of check Z_m to be satisfied if bit n is considered fixed at 0 and the other bits have a separable distribution.

Appendix B

Belief Propagation Algorithms

B.1 Message-Passing

B.1.1 Message-Passing Algorithm : Random Method

```
For Nmax iterations

  While not { all the checks have their messages updated and
              all the nodes have their messages updated } Do
    Select randomly a check m among those not yet updated
    For all nodes n ∈ N(m) Do
      Update its messages r_mn ( Update–R() )
    EndFor

    Select randomly a node n among those not yet updated
    For all checks m ∈ M(n) Do
      Update its messages q_mn ( Update–Q() )
    EndFor
  EndWhile

  If (solution to the decoding is reached) Then
    Exit
```

Else

Continue

EndIf

EndFor

B.1.2 Message-Passing Algorithm: Iterative Method

For Nmax iterations **Do**

For all the checks m **Do**

For all nodes $n \in N(m)$ **do**

Update its messages r_{mn} (Update–R())

EndFor

EndFor

For all the nodes n **do**

For all checks $m \in M(n)$ **do**

Update its messages q_{mn} (Update–Q())

EndFor

EndFor

If (solution reached) **Then**

exit

Else

continue

EndIf

EndFor

B.2 Functions Detail Algorithm

B.2.1 Initialisation

For all the nodes n **Do**

For all their checks $m \in M(n)$

$Q_{mn} = f_n$: the prior probability of node n: $P(X_n = 0)$

EndFor

EndFor

B.2.2 Checks Update: Update-R()

This algorithm works for a syndrome vector $\mathbf{z}$ and two matrices; one of cavity magnetisation Q and one of check satisfaction R.

For all nodes n **Do**

For all the checks m **Do**

$\delta q_{mn} = 2 * q_{mn} - 1 \; (= q^0_{mn} - q^1_{mn})$

EndFor

EndFor

Update Check m :

For all the nodes $n \in N(m)$ **Do**

$\delta r_{mn} = (-1)^{z_m} \prod_{n' \in N(m) \backslash n} \delta q_{mn'}$

$r_{mn} = 0.5 * (1 + \delta r_{mn})$

EndFor

B.2.3 Nodes Update: Update-Q()

This algorithm works for a given prior f and two matrices of cavity magnetisation Q and check satisfaction R.

Update node n :

For all the checks $m \in M(n)$ **Do**

$q^x_{mn} = \alpha_{mn} f^x_n \prod_{m' \in M(n) \backslash m} r^x_{m'n}$

where α_{mn} is such that $q^o_{mn} + q^1_{mn} = 1$

$q_n^x = \alpha_n f_n^x \prod_{m \in M(n)} r_{mn}^x$

where α_n is such that $q_n^o + q_n^1 = 1$

EndFor

B.2.4 Test Found Solution: Test-Result()

This algorithm works for a given pseudo-posterior probability vector $\hat{\mathbf{q}}$, and a given ϵ for determining the convergence.

Create found code $\hat{x}$:

For all n **Do**

$\hat{x_n} = 1$ **if** $q_n < 0.5$ and 0 otherwise

EndFor

Test of convergence:

For all n

If q_n is stabilised during 5 loops **Then**

algorithm halts

Else

continue

EndIf

B.2.5 Time Averaging

This algorithm works for a given *Nmax* maximum number of iterations above which the algorithm halts, and a certain *time window Tav* for averaging the magnetisations.

For Nmax iterations

For Tav iterations

```
    Run Belief Propagation
    Add the pseudo-posterior probabilities
  EndFor
  Average the pseudo-posterior probabilities
  Sequential Fixing

EndFor
```

B.2.6 Sequential Fixing

This algorithm works for a given *threshold* and a pseudo-posterior averaged probabilities vector of nodes.

```
Fix all the nodes which pseudo-posterior probability is above the thresho
If none is found then
  fix the highest probability state node value
Endif
```

Appendix C

Graph Generation Detailed Algorithm

C.1 Random Graph Generation

This algorithm works for a given size for the sparse matrix, K the number of unit elements per row (sufficient to determine a rate R).

Make a list containing all the available checks
(represented by their location on the graph)

FIRST ALLOCATION

For each node n
Select randomly K checks among those remaining in the list
Allocate them to the node n
Delete these checks from the list
EndFor

LOOP TREATMENT

For each node n

For each node $n' \neq n$

If n and n' have more than one check in common **Then**

Select randomly a third node n''

Exchange one check with n' looping one

(such that the loop between n and n' no longer exists)

EndIf

EndFor

EndFor