Algorithms For Solving Optimization Problems On Large Random Graphs

YOUENN FORTUNE

MSc by Research in Pattern Analysis and Neural Networks

ASTON UNIVERSITY

September 2004

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.

ASTON UNIVERSITY

Algorithms For Solving Optimization Problems On Large Random Graphs

YOUENN FORTUNE

MSc by Research in Pattern Analysis and Neural Networks, 2004

Thesis Summary

The problem of vertex colouring in large random graphs is unlikely to be an easy task. Given a number of available colours (in our study we used three colours), there are two relevant values of the graph connectivity delimiting three different states: when it is quite easy to find a solution, when it is very hard and when it is almost impossible. We adapt an existing algorithm using message-passing techniques to graphs and hypergraphs (here hypergraph means that the graph contains edges linking three vertices) and study its behaviour in the different states with large random graphs. We then compare it with an exact enumeration algorithm on small systems.

Keywords: Graph colouring, belief propagation, exact enumeration, statistical physics, message-passing technique

Acknowledgements

First, I would like to thank my supervisor, Dr Jort van Mourik, for his patience and availability all along this project. I would also like to thank the Neural Computing Research Group, especially my course lecturers Prof David Lowe, Prof Ian Nabney, Prof David Saad and Dr Manfred Opper. Special thanks to Dr Laura Rebollo-Neira for her night company and to Dr Dan Cornford for his enthusiasm to play football. I wish to thank Ms Vicky Bond for her care and particular efficiency to treat the procedures involved.

My course fellows: Christophe, Youssef, Thomas, Kiriko, Dharmesh and Pierre, and the PhD students Ben, Boremi and Dan are thanked for the good time shared in the Wolfson lab or on the football pitch. Thanks to Stephane for his stimulating discussions and to Miroslav for his LaTeX support.

I also thank Fabrice and Thomas, two close friends, for sending me so few emails all the year long.

I would like to gratefully acknowledge all kind of supports of my parents during the whole year.

Obvious thanks go to Vincent, a special friend for his remote moral support, even from the United States.

Finally, I would like to thank Kleopatra for her understanding, her support and her love.

Contents

1 Introduction **8**
- 1.1 Graph colouring . . . 8
  - 1.1.1 Basic definitions . . . 8
  - 1.1.2 How to colour a graph ? . . . 8
- 1.2 Applications . . . 10
  - 1.2.1 Production scheduling and timetabling . . . 10
  - 1.2.2 Frequency assignment in wireless communication . . . 10
  - 1.2.3 Distributed storage . . . 10
  - 1.2.4 Partitioning variables onto multiple register files . . . 11
  - 1.2.5 Design of multifiber Wavelength Division Multiplexing networks . . . 11
  - 1.2.6 Matrix partitioning . . . 12
- 1.3 The important parameters . . . 12
  - 1.3.1 The size of the graph: N_v . . . 12
  - 1.3.2 The mixture coefficient: α . . . 12
  - 1.3.3 The connectivity: λ . . . 13
- 1.4 Existing methods . . . 14
- 1.5 Message-passing technique . . . 15
- 1.6 Goals of the project . . . 16

2 Pre-requisites **17**
- 2.1 The implementation . . . 17
- 2.2 Random graph generator . . . 17
- 2.3 Random colourable graph generator . . . 18
- 2.4 Pruning . . . 19
- 2.5 Sequential fixing . . . 20
- 2.6 Parallelization . . . 21
  - 2.6.1 Description of the problem . . . 21
  - 2.6.2 What is existing? . . . 22
  - 2.6.3 What is new? . . . 22

3 Belief propagation on large graphs **24**
- 3.1 A probabilistic approach: the Sum-Product Algorithm . . . 24
- 3.2 An application: the graph colouring algorithm . . . 26
- 3.3 The cavity rule . . . 26
- 3.4 A brief description of statistical physics . . . 27
- 3.5 Results . . . 29
- 3.6 The paramagnetic state . . . 31
- 3.7 Time averaging . . . 34
  - 3.7.1 Reasons for non convergence . . . 34
  - 3.7.2 Time averaging as a solution . . . 35
- 3.8 Results . . . 35

4 Exact enumeration on small graphs **41**
4.1 Backtrack algorithm . . . 41
4.1.1 Aims . . . 41
4.1.2 Some catalysts for exact enumeration . . . 42
4.1.3 Results . . . 42
4.2 Belief propagation versus exact enumeration . . . 44

5 Conclusion **47**
5.1 Achievements . . . 47
5.2 Further work . . . 48

A Pseudo-code of Belief Propagation **49**
A.1 Message-passing technique . . . 49
A.1.1 Iterate simply . . . 49
A.1.2 Iterate and calculate the magnetisation . . . 50
A.1.3 Fix . . . 50
A.1.4 Iterate and fix . . . 51
A.2 Belief propagation updates . . . 52
A.2.1 Vertex update . . . 52
A.2.2 Edge update . . . 52
A.3 Survey propagation . . . 53
A.4 Presentation . . . 53
A.5 The formulae . . . 53
A.6 The algorithm . . . 54

B Parallelization algorithm **55**
B.1 Master machine . . . 55
B.2 Slave machine . . . 56

C Backtrack algorithm **57**
C.1 Backtrack algorithm for perfect colourings . . . 57
C.2 Backtrack algorithm for optimal solutions . . . 58
C.3 Graphical diagram of the algorithm . . . 59

D Statistical physics notations **61**

E Energy calculations **62**

F Additional simulations **64**

List of Figures

1.1 First example of graph colouring 9
1.2 Message-passing example 16

2.1 Random graph generation tests 19
2.2 Example of pruning 20
2.3 Embarrassingly parallel 23
2.4 Efficient parallelization 23

3.1 Example of the sum-product algorithm 24
3.2 Ground state energy and entropy, $\alpha = 0$ 29
3.3 Ground state energies and entropies, $\alpha > 0$ ($\alpha \in \{0.25, 0.5, 0.75, 1\}$) 30
3.4 Cavity distribution, $\alpha = 0.5$ 32
3.5 Phase diagram, $\alpha = 0$ 32
3.6 Phase diagrams, $\alpha > 0$ 33
3.7 Belief propagation efficiency on random graphs, $\alpha = 0$ 36
3.8 Belief propagation efficiency on random graphs, $\alpha > 0$ 37
3.9 Belief propagation duration on random graphs, $\alpha \geq 0$ 38
3.10 Belief propagation efficiency on random colourable graphs, $\alpha = 0$ 39
3.11 Belief propagation efficiency on random colourable graphs, $\alpha > 0$ 40

4.1 Exact enumeration efficiency on random graphs, $\alpha = 0$ 42
4.2 Exact enumeration efficiency on random graphs, $\alpha > 0$ 43
4.3 Exact enumeration vs Belief Propagation, $\alpha = 0$ 45
4.4 Exact enumeration vs Belief Propagation, $\alpha > 0$ 46

C.1 Backtrack algorithm graphical representation 60

F.1 Exact enumeration vs Belief Propagation, $\alpha = 0$ and $N_v = 20$ 64
F.2 Exact enumeration vs Belief Propagation, $\alpha > 0$ and $N_v = 20$ 65
F.3 Exact enumeration vs Belief Propagation, $\alpha = 0$ and $N_v = 40$ 66
F.4 Exact enumeration vs Belief Propagation, $\alpha > 0$ and $N_v = 40$ 67

List of Tables

3.1 Upper bounds of λ_c, $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$ 31
3.2 Estimation of λ_d, $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$ 34
3.3 Estimated λ_c, $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$ 36

4.1 Estimated boundaries for λ_c and $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$ 44

Chapter 1

Introduction

Before studying the algorithms in detail, we need to describe the problem accurately, in order to introduce the necessary notations and the mathematical notions.

1.1 Graph colouring

1.1.1 Basic definitions

We define a graph $\mathcal{G}(N_v, \alpha, \lambda)$ as a set of N_v vertices, also called nodes, linked to each other by edges or hyperedges . The variable α represents the mixture coefficient between two types of edges: those connecting exactly two vertices (2 body-interaction edges) and those connecting exactly three vertices (3 body-interaction edges or hyperedges). One should then keep in mind that α belongs to the interval $[0, 1]$. A hypergraph is a graph containing hyperedges. The parameter λ, called the connectivity of the graph, is the average number of edges per vertex.

1.1.2 How to colour a graph ?

Let us now assume that for each vertex v of this graph, we have a set of available colours $\{\mu_{v_1}, \ldots, \mu_{v_c}\}$, where $\{v_1, \ldots, v_c\}$ depend on v. The graph colouring problem is to assign a colour to each vertex so that all (or the maximum number) of the constraints on the edges are satisfied. The constraint here is that not all the vertices connected to the same edge have the same colour, i.e. an edge cannot be monochromatic. If an edge connects two vertices, the condition becomes: the vertices must have different colours. It is obvious that the constraint on that kind of edge is stronger than the constraint on a three body-interaction edge, since there are q times more ways to satisfy the second than the first (if q is the number of colours, there are $q(q-1)$ ways to satisfy the edge and $q(q^2-1)$ ways to satisfy the hyperedge), whereas in both cases there are q ways not to satisfy it.

Although it is possible to define a different set of available colours for each vertex, we will deal with the case where it is the same set for all the vertices. We will denote this set $\{1, \ldots, q\}$. An example of a perfect colouring is shown in Figure 1.1.

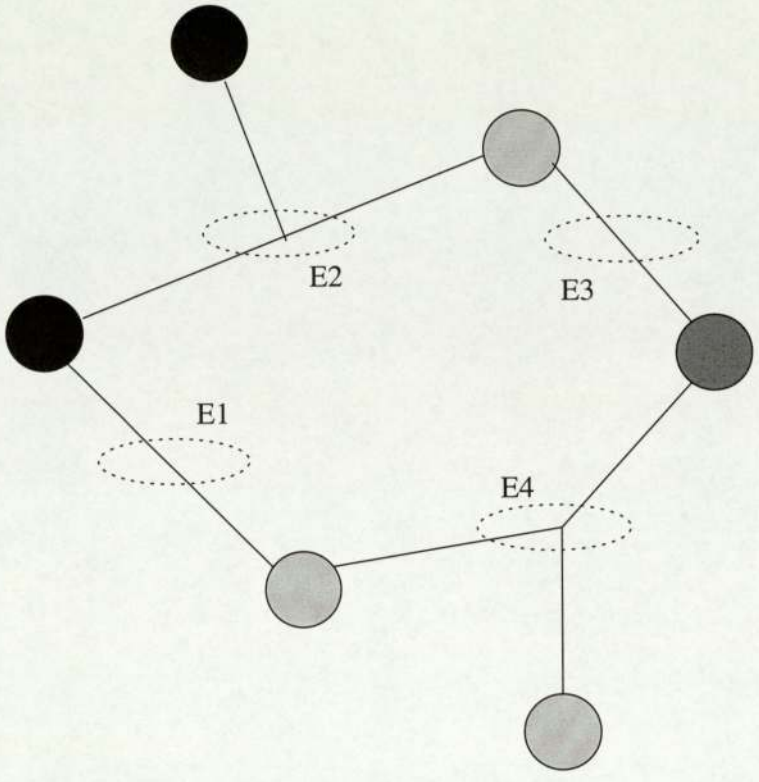


Figure 1.1: A graph, $\mathcal{G}(6, 0.5, 5/3)$. E1 and E3 are two body-edges while E2 and E4 are three body-edges. Three colors are enough since λ is very small. Note that this is one among many possible colourings.

In the example, three colours have been used, although the graph could have been coloured with only two colours. Actually, the graph colouring problem is divided into two main issues : the q-colouring problem, i.e. trying to find if a graph can be coloured with q given colours and when that is not possible to try to reach the best solution (with the smallest number of unsatisfied edges), and the optimal colouring problem, i.e. trying to find the minimum number of colours required to colour a graph (called the chromatic number, $\chi(\mathcal{G})$). In this thesis, we focus on the first problem.

The q-colouring problem of random graphs is a field on which a considerable amount of research has already been done, including, for example, the famous 4-colours theorem [2, 3], stating that every planar graph can be 4-coloured, but there are still many remaining tasks to explore. It is the natural evolution of the percolation theory initiated by Erdös and Rènyi in the middle of the past century [17]. Indeed, it is a very important issue not only in the field of discrete mathematics, but also for

real world applications.

1.2 Applications

Although, for many settings, no perfect algorithms to solve the graph colouring problem are known, any algorithm being able to come close to a solution can be useful for many real world problems that can be mapped onto graph colouring problem. The first three examples deal with two body-edge graphs while the others deal with many body-edge (hyper)graphs.

1.2.1 Production scheduling and timetabling

Let us consider, for example, a set of lectures (vertices) and a set of time slots (colours) available for the head of the timetables department. To build the graph representation, we need to put an edge between two lectures when they have students in common, and hence they cannot take place in the same time slot. The problem consists in assigning the time slots in such a way that these restrictions are satisfied. Determining the minimum number of time slots needed is equivalent to finding the chromatic number of the associated graph. We could also add additional constraints, such as the preference of a lecturer not to teach on Friday afternoons.

1.2.2 Frequency assignment in wireless communication

In this type of application, studied by Gamst [21], the vertices represent transmitters, the list of colours allowed for a vertex contains the frequencies locally available, and adjacency means that interference may occur between the corresponding two stations and hence they should not use the same frequency. In this interpretation, selecting large subsets of the colour lists ensures that the stations can operate on a fairly wide range of frequencies without the danger of interfering with each other.

1.2.3 Distributed storage

In this quite new issue, coming from private discussions, there is a network of electronic units (for example mobile phones or computers) spread over space. In order to save memory space, the data is split into several parts, the colours. The goal here is to store one part in each unit, the vertex, of the network so that one unit can have access to the missing parts of the data in its direct neighbours. Note that here the constraint is not the same as the one described above: each vertex must be able to see the maximum number of colours.

1.2.4 Partitioning variables onto multiple register files

Current multiple-functional-unit architectures allow the boosting of performance by executing many operations, but register file technology can provide only a limited input/output (I/O) data-bandwidth, and can only support a few, fast, pipelined functional units.

Register allocation, i.e. the process of allocating each instance of a variable to a register, has been historically performed by using Chaitin' algorithm [13, 8].This model, though, cannot be efficiently extended to deal with the additional constraints imposed by a non-homogeneous register space, such as multiple register banks.

In [10, 11], Capitanio *et al.* assume, as a model of execution, a horizontally microcoded, multi-functional-unit architecture, a Very Long Instruction World (VLIW) architecture. The model is parameterized by the number of register files (R) and the number of functional units (F), and it provides 2F read ports and F write ports evenly distributed among the register files, i.e. each register file has $\frac{2F}{R}$ read-ports and $\frac{F}{R}$ write-ports.

Given a program compiled for a single-register file VLIW, its variables must be partitioned into R sets, each to be allocated on a register file. This can be done with the help of a hypergraph whose nodes are the variable registers and whose hyperedges model the competition among variable register concurrently accessed within an instruction. Note that there are two types of edges: read and write edges. One has to colour the nodes of the graphs, the colours represent the register files to which a variable will be assigned, so as to satisfy two types of constraints depending on the type of the hyperedge. A read-hyperedge must not contain more than $\frac{2F}{R}$ nodes of the same colour, since it is representative of an instruction which accesses no more than $\frac{2F}{R}$ operands stored in the same register file and that can therefore can be legally executed and a write-hyperedge must not contain more than $\frac{F}{R}$ for similar reasons.

1.2.5 Design of multifiber Wavelength Division Multiplexing networks

In [19], Ferreira et al. consider the design of multifiber Wavelength Division Multiplexing networks. Given a network with a set, N, of nodes, a set, L, of links each with k fibers and a set, P, of lightpaths (end-to-end connections), the (k, w)-wavelength assignment problem is to assign each path with one out of w wavelengths so that for every link, no more than k paths using the link receive the same wavelength. If the set of vertices is P and the set of hyperedges is $L = \{l \in L, \exists e \in E, e = \{p \in$

P, p involves l}}, the problem is to assign a wavelength among w (the colours) such that

$$\forall e \in E, \forall q \in \{1, \ldots, w\}, \quad \Big|\{v \in E, \phi(v) = k\}\Big| \leq k,$$

where ϕ is the mapping function. Here, there are two optimization problems : given k, minimize w or given w, minimize k.

1.2.6 Matrix partitioning

Another problem is matrix partitioning (see [23]). Given a $m \times n$ matrix with zeros and ones, try to partition the columns of this matrix such that any two columns in the same group do not have a nonzero at the same row position. This can be treated both with graphs and with hypergraphs.

1.3 The important parameters

It is important to define the main parameters properly, since we will do a lot of simulations with several combinations between them.

1.3.1 The size of the graph: N_v

This is the number of vertices of the graph. Our work will focus on two types of graphs: large graphs, $N_v = 1000$, and small graphs, $N_v \in \{20, 30, 40, 50\}$. We notice that the size of the graph is an important parameter because some algorithms can only be applied to relatively small graphs. Large graphs are required for the statistical physics part.

1.3.2 The mixture coefficient: α

As it was previously defined, α represents the parameter that will decide whether we deal with graphs or hypergraphs. α will take the following values $\{0, 0.25, 0.5, 0.75, 1\}$, so as to depict the interval $[0, 1]$ the best we can. If we note N_{e_2} the number of 2 body-edges, N_{e_3} the number of 3 body-edges and N_e the total number of edges, we have the following relation:

$$N_e = (1 - \alpha) \times N_{e_2} + \alpha \times N_{e_3}. \tag{1.1}$$

We also define K as the number of vertices an edge is connected to and then $K \in \{2, 3\}$. It is

convenient to introduce K_{avg} as the mean of all the K. Hence, from Equation (1.1), we have:

$$\begin{aligned} K_{avg} &= 2 \times (1 - \alpha) + 3 \times \alpha \\ &= 2 + \alpha. \end{aligned} \tag{1.2}$$

So far, studies have been mainly on graphs with α equal to zero, with two and three colours [40, 18], and with α equal to one, with two colours [12].

1.3.3 The connectivity: λ

This is a crucial parameter. One can easily understand that for a sparsely connected graph and many colours, colouring is straightforward. On the contrary, with a high connectivity and few colours, the graph will be surely uncolourable. Graph colouring is a very old and difficult problem belonging to the category of NP-hard problems [22, 29]. It seems unlikely that a deterministic polynomial time algorithm will be found, especially in the area of the parameter space where it is very hard to find a solution, around the critical connectivity, λ_c.

Actually, the critical connectivity represents the limit, called the q-COL/UNCOL transition, between two kinds of graphs. Beyond this limit, graphs are uncolourable with a probability tending to one as the size of the graph goes to infinity and below this limit, graphs are colourable with a probability tending to one as the size of the graph goes to infinity. This is a point of contact between computer science and graph theory.

Moreover, [4] showed that there is a connectivity, λ_d, close to the critical connectivity, $\lambda_d < \lambda_c$, defining a clustering phase for $\lambda \in [\lambda_d, \lambda_c]$ in which ground states spontaneously divide into an exponential number of clusters. Indeed, far below λ_d the solutions form one large cluster and a small step from one solution will still provide a solution, whereas for values of $\lambda \in [\lambda_d, \lambda_c]$, the solutions form several small clusters and then it is very difficult to go from one cluster to another. In this region, current, complete and stochastic algorithms are known to fail already for moderate system sizes. We will try to determine λ_d using the phase diagrams.

The best lower bound for λ_c found so far for the 3-COL/UNCOL transition on graphs with two body edges is 4.03 [1], the best upper bound is 4.99 [28], and whereas numerical results predict a threshold of about 4.7 [16], the best value theoretically predicted is 4.69 [36] and is believed to be exact. The clustering transition is, in this case, close to 4.42 [4].

Defined as the average number of edges per vertex, λ is then the key parameter in defining the

probability of an edge connecting K vertices :

$$P_e(v_{i_1}, \ldots, v_{i_k}) = \frac{\lambda}{N_{pe/v}}, \quad \text{with } N_{pe/v} = \begin{pmatrix} N_v - 1 \\ K - 1 \end{pmatrix}, \tag{1.3}$$

where $N_{pe/v}$ represents the number of possible edges per vertex.

In this thesis, we focus on sparse graphs. A sparse graph has a low connectivity with respect to the total number of possible edges, i.e. $P_e \ll 1$.

Given these three parameters, the total number of edges is calculated from the fundamental relation:

$$\lambda \times N_v = K_{avg} \times N_e, \tag{1.4}$$

and we get:

$$N_e = \frac{\lambda N_v}{2 + \alpha}. \tag{1.5}$$

This will be very useful in the forthcoming calculations.

1.4 Existing methods

The naive approach to solve this problem is to test all the possibilities. Of course, since the complexity grows exponentially with the size of the graph, it soon becomes intractable. This technique, known as exact enumeration or implicit enumeration [9, 15, 31], even with tricks to avoid unnecessary counting cannot be applied to very large graphs. Nevertheless, it is the only algorithm which can state with absolute certainty whether a graph is colourable or not. Therefore we will use it to perform some tests on small graphs.

A compromise between running time and quality of the solution, has been found with non-deterministic methods, based on heuristics. One can divide these algorithms into two groups, depending on how they are initialized:

- Algorithms starting from a full initial randomized colouring, such as tabu search [24] and simulated annealing [14, 26]. They consist of a big loop inside of which a random vertex is picked to change its colour, then if the number of unsatisfied edges is improved, the vertex will keep its new colour, otherwise, the colour will be accepted with a certain probability. The main difference between these two algorithms is the way the new vertex is chosen,

- Algorithms starting from a partial colouring of the graph, also called successive augmentation. Then at each step of these algorithms, few vertices are coloured, until complete colouring of

the graph is achieved. Among these algorithms, are belief propagation [40, 5] and survey propagation [6, 5, 4].

Other heuristics, based on Brown's implicit enumeration algorithm [9], have been implemented by Brelaz [7] and by Matula [34], among the most efficient, to find the chromatic number of a graph.

A major part of these algorithms has been tested on dense graphs, i.e. a vertex is, on average, connected to more than half of all the remaining vertices, whose size vary between 200 and 1000 vertices, to find the chromatic number of the graph. Actually, there is a group of graphs, called Leighton graphs [32], for which the chromatic number can be very well approximated by calculations.

Furthermore, issues related to finding an appropriate neighbourhood structure for colouring graphs are described in [35].

1.5 Message-passing technique

Iterative message passing algorithms have found applications in a wide range of data detection problems because they can provide near optimal performance and significant complexity reduction. The variables and the constraints are arranged into a *belief network* (also known as a Bayesian network) and the probabilities are iteratively updated by message-passing along the edges. The specific algorithm we will study, is adapted from the sum-product algorithm and was first used by Pearl [37]. Message-passing techniques rely on neighbourhood influences to find a correct assignment of states to the variables given the constraints, like the satisfiability problem [39] or the decoding problem for error correcting codes [20], where it has been successfully applied.

Here, the initial graph is transformed into an undirected bipartite graph (see Figure 1.2). The vertex set is composed of the original edges and vertices, and the links are the connections between these original edges and vertices. Through these links, vertices and edges communicate. The message sent by an edge e to a vertex v will represent a warning message saying to the vertex that it should not take the colour μ, denoted $\eta^{\mu}_{e\to v}$. On the other side, messages sent by a vertex v to an edge e is interpreted as the probability that the vertex takes the colour μ in the absence of e, denoted $\eta^{\mu}_{v\to e}$. Note that for each vertex and each edge connected to this vertex, $\sum_{\mu=1}^{q} \eta^{\mu}_{v\to e} = 1$. We will later give the probabilistic interpretation of these quantities.

Note that the *magnetisation* of a vertex v for a colour μ, i.e. the probability that a vertex will opt for the colour μ, can be calculated from these messages.

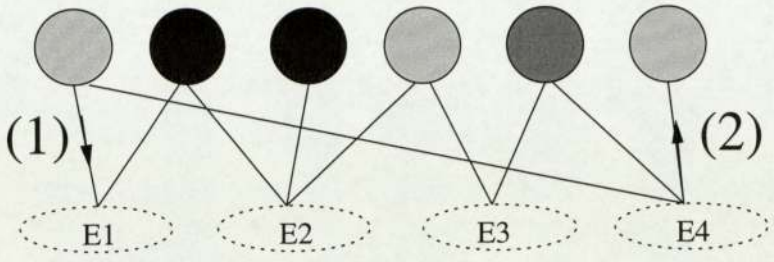


Figure 1.2: A bipartite graph representation of the graph of Figure 1.1. (1) represents $\eta^{\mu}_{v\to e}$ and (2) represents $\eta^{\mu}_{e\to v}$.

1.6 Goals of the project

Now that all the basic definitions have been introduced, the aims of this project can be underlined. First, we will study the analogy between statistical physics and Belief Propagation on large random graphs and then determine bounds for the dynamical connectivity, $\lambda_d(\alpha)$, $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$. Then using an exact enumeration algorithm, called backtrack algorithm, we will find bounds for the critical connectivity, $\lambda_c(\alpha)$, $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$. Finally, we will compare these two algorithms on small graphs.

A study of the parallelization methods on a cluster of computers will be an important part, since it is a novel technique. The goal is to exploit the power of the 'divide and conquer' principle.

An interesting work to continue, is the comparison between Belief Propagation and Survey Propagation for a mixture of 2- and 3-body interaction graphs, since these are two new algorithms using message-passing techniques.

Chapter 2

Pre-requisites

2.1 The implementation

All the programs have been written in the C language, since the structure is quite simplified, using the Numerical Recipes implementation of vectors, matrices and tensors [38]. Actually, this structure is used for data storage, as the calculations require fast access to this data in order to perform a lot of simple operations such as additions, subtractions, multiplications and divisions. That way, a lot of time can be saved and we can take advantage of the main power of the C language : its speed to do basic calculations.

It should be noted that the belief propagation algorithm was already implemented by Fabre in [18] in the C++ language. The object implementation slows down the program so much that the coding flexibility is outweighed. This has reinforced the motivation for choosing the C language.

Before coding the main algorithm, it is necessary to code several tools.

2.2 Random graph generator

The study of graph colouring will require many graphs on which to perform the simulations. These graphs of course must be randomly generated given three variables: the total number of vertices N_v, the average number of vertices per edge K_{avg} and the connectivity λ. The average number of vertices per edge, K_{avg}, is given by :

$$K_{avg} = 2 \times (1 - \alpha) + 3 \times \alpha, \quad K_{avg} \in [2, 3], \quad K \in \{2, 3\}.$$

It is obvious that K_{avg} is not always an integer, but this will not affect the results. The probability, P_e, of an edge connecting K given vertices is :

$$P_e(v_1, \ldots, v_k) = \frac{\lambda}{N_{pe/v}}, \quad \text{with } N_{pe/v} = \begin{pmatrix} N_v - 1 \\ K - 1 \end{pmatrix}. \tag{2.1}$$

To give an interpretation of the variables used, let us note precisely that $N_{pe/v}$ is the number of possible edges per vertex. One could easily generate a graph by spanning all the K-tuples and accept the creation of the edge with a probability P_e, but its computational cost is of order $\mathcal{O}(N_v^K)$.

The best method to generate the graph is to fix the final number of edges to $N_e = \frac{\lambda N_v}{K_{avg}}$, such that the final connectivity comes very close to the desired one. Then for each edge, a number between K_{avg} and $K_{avg} + 1$ will be randomly chosen, so that the rounding to an integer will give either 2 or 3 and will keep the proportional coefficient α constant. Finally, two or three different vertices will be chosen randomly and thus define the new edge, on the condition that it does not already exist. This algorithm is $\mathcal{O}(N_v)$.

To validate this procedure, we can test the distribution of the number of edges per vertex ($n_{e/v}$) and compare it with its theoretical value. As the size of the system goes to infinity ($N_v \to \infty$), the graph should become Poissonian, i.e. the random variable $n_{e/v}$ becomes Poisson distributed, since it follows a binomial rule, $b(k; N_{pe/v}, P_e(v_1, \ldots, v_k))$:

$$\forall k \in \mathbb{N}, P(n_{e/v} = k) = \begin{pmatrix} N_{pe/v} \\ k \end{pmatrix} P_e(v_1, \ldots, v_k)^k \left(1 - P_e(v_1, \ldots, v_k)\right)^{N_{pe/v}-k} \tag{2.2}$$

$$= \begin{pmatrix} N_{pe/v} \\ k \end{pmatrix} \left(\frac{\lambda}{N_{pe/v}}\right)^k \left(1 - \frac{\lambda}{N_{pe/v}}\right)^{N_{pe/v}-k} \tag{2.3}$$

$$\lim_{N_v \to \infty} P(n_{e/v} = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \quad k \in \mathbb{N}. \tag{2.4}$$

In Figure 2.1, on the left, one can observe the efficiency of the graph generator.

2.3 Random colourable graph generator

Despite the fact that beyond the critical connectivity, λ_c, it is very rare to find a graph which is colourable, we can nevertheless generate colourable graphs. We can achieve that by grouping the vertices in q blocks and allowing the creation of an edge only between different blocks, i.e. no connection

inside a block. It will be useful to test the strengths and weaknesses of the algorithms on that kind of graphs.

Again, the validation tests provide good results (see Figure 2.1, on the right).

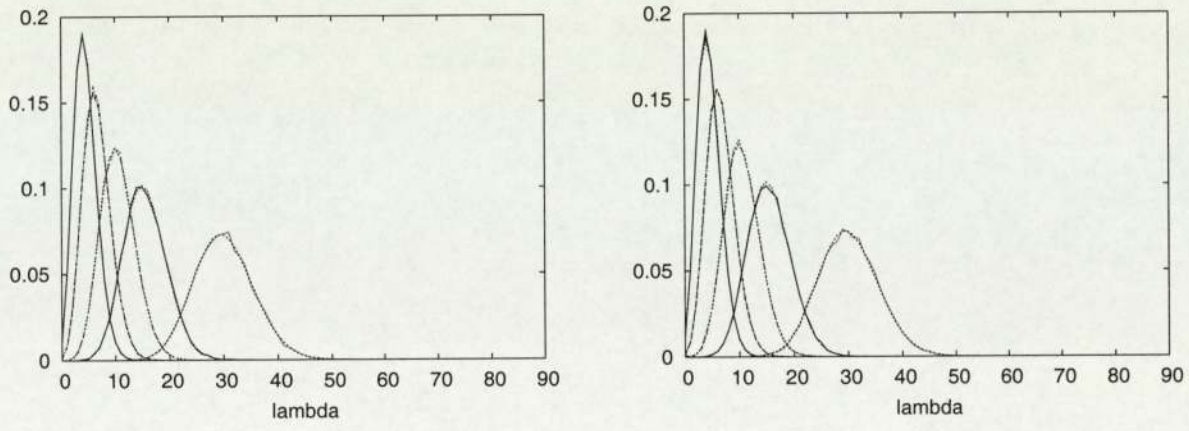


Figure 2.1: Validation tests: random graph generator on the left, random colourable graph generator on the right. Sample of 30 graphs with $N_v = 1000$ for $(\alpha, \lambda) \in \{(0,4); (0.25,6); (0.5,10); (0.75,15); (1,30)\}$.

2.4 Pruning

Since the graph has to be coloured with q colours, the vertices having less than q neighbours can be ignored during the colouring process. Indeed, they will be colourable at the end, depending on their remaining neighbours' colours. The pruning process is recursively applied to the rest of the vertices linked to the pruned vertex.

To reduce the size of the graph, we prune the vertices that have less than q neighbours. A neighbour, here, means the number of edges connected to the vertex and not the number of vertices because we also deal with hypergraphs. Actually, this operation, which is of course reversible, allows a significant reduction of the size of the graph for small connectivities. Up to a certain connectivity, depending on α, the graph is almost always prunable, so that it is not necessary to run any other colouring algorithm.

In the end, when the graph is coloured, we can colour the pruned vertices in the reverse order of their pruning. See Figure 2.2 for an example of this tool.

The advantage of running this process first, is a gain in terms of reduction of complexity, hence in speed, and a gain of accuracy, because these pruned vertices will not disturb the message-passing

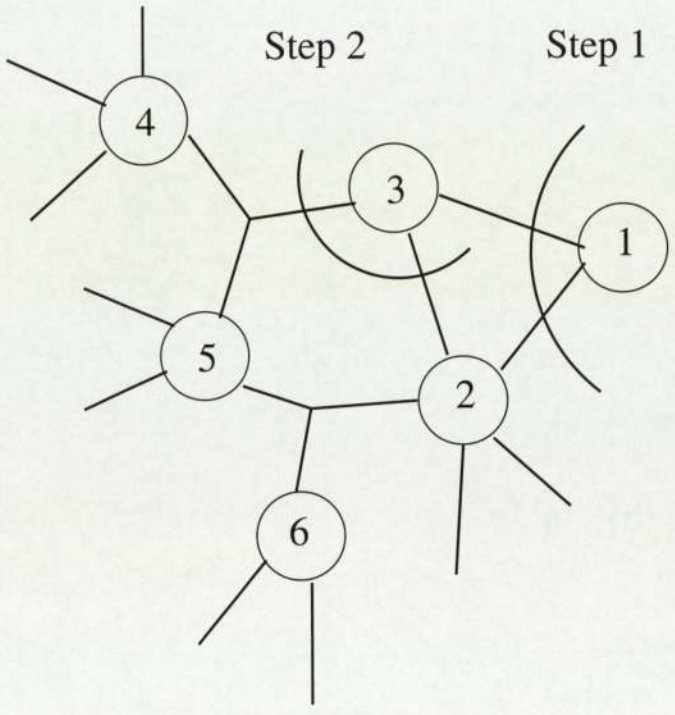


Figure 2.2: With 3 colours, we can eliminate vertex 1 and then vertex 3. After the colouring process, imagine that vertex 2 has colour c_3 and that vertices 4, 5 and 6 have the same colour c_1. Then vertex 3 will take the colour c_2 and vertex 1 will take the colour c_1.

algorithm by sending useless messages to their neighbourhood.

2.5 Sequential fixing

This function concerns the belief propagation algorithm only.

When convergence of the message-passing technique is reached, one has to assign to each vertex that colour that has the highest magnetisation in order to get the final colouring.

However, for graphs or parts of the graph where the connectivity is low or locally low, the area is underconstrained. The consequence is that no vertex will opt for one colour but for all the colours indifferently. This is the so-called paramagnetic state. All (or a part of the vertices) have the same probability $(1/q)$ to choose any colour, and thus the problem arises of assigning the proper colour to a node. One must note that the paramagnetic state is always a stable *solution* of the Belief Propagation equations below λ_d.

To avoid the paramagnetic state, we perform sequential fixing. It consists in repeating the following

two steps:

1. run the algorithm with a fixed number of iterations (if convergence is reached before, skip the remaining iterations and go to next step),

2. fix all the vertices whose magnetisation is above a given threshold,

while there are unfixed vertices remaining. Thus the paramagnetic state is avoided and the final colouring may be extracted properly.

An important issue is to consider the value of the threshold. If it is too low, the vertices will be fixed too early and then this will lead to poor performance, whereas if it is too high, too few vertices will be fixed at each iteration, increasing the computational cost significantly without giving much better results.

From [18], the threshold has been empirically found to be optimal at 0.8, for $\alpha = 0$, providing a very good compromise. Even for $\alpha > 0$, it remains a very good choice. We have to keep in mind that a threshold of 0.8 means that a vertex has to be 80% sure before it will take the given colour.

2.6 Parallelization

2.6.1 Description of the problem

In May, we received in our department twenty-five two-processors computers, the so-called cluster of computers. This ensemble of computers is integrated by an interconnection network and is operating within a single administrative domain. There is one master computer, using two processors, driving the forty-eight others. The main characteristics of this configuration are that all the processors are independent and that there is no shared memory, hence the need to make them communicate efficiently.

In our study, we parallelized the test procedures but not the colouring algorithm in itself. Since there is a huge amount of tests to run, parallelization is a time saver.

Actually, as seen in Section 1.3, there are three parameters to vary. For each combination, we have to generate many different graphs (25, 150 or 1000 graphs for example) and to run the colouring algorithm for each graph. Depending on the combination, applying the algorithm on the sample test can take few minutes up to one week.

2.6.2 What is existing?

The main way of using the cluster so far is called embarrassingly parallel. This consists in sending several independent jobs to the cluster. The master machine will then manage the queue and send the first job to the first free machine and so on. Here there is no communication between the jobs. But this is not optimized at all for the kind of jobs we want to do.

Another implementation has been done, called TACO, but this relies on shell scripts, which is not as fast as C code.

That is why we need to find a way such that we do not loose that much time.

2.6.3 What is new?

We used the MPI [25] package within the C code.

The idea is to define a master node (or processor) which manage the samples and sends to each of the free nodes one single graph to be coloured at the time. An implementation of this algorithm can be found in Appendix B.

Unlike the embarrassingly parallel method which takes a parameter combination and performs a whole sample on one node, this way of processing allows us to cut the sample into its smallest entity, a graph. Then the sharing out of the jobs is much more balanced among the nodes and thus efficient.

In this way, all the processors are used close to a hundred percent of the time. An example is shown in Figures 2.3 and 2.4. As long as one node is busy, all the others will also be blocked, even if they are not doing anything. Thus, the efficiency of well distributing the jobs is obvious.

Since data transfer between the master and the slave machines is time-consuming, the results of each colouring will be written directly in a log file identified by the number of the slave machine rather than being sent back to the master processor. This will require the creation of a parser program to sort and treat all the collected data.

The results were good and allowed to perform tests on a larger sample, which will be especially useful for the exact enumeration algorithm.

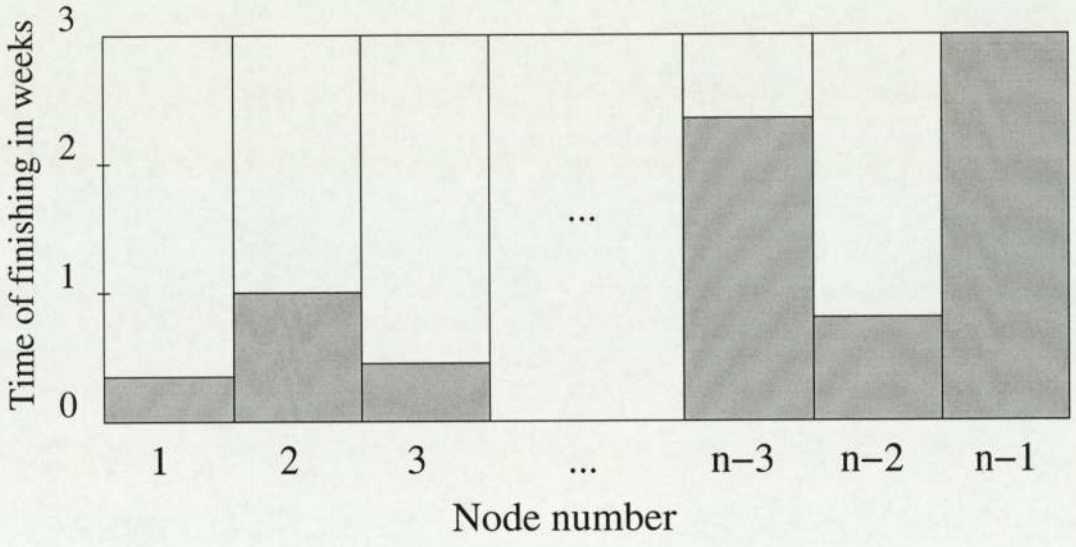


Figure 2.3: In the embarrassingly parallel configuration, the fastest samples have to wait for the slowest. In our case, the longest sample can take up to three weeks.

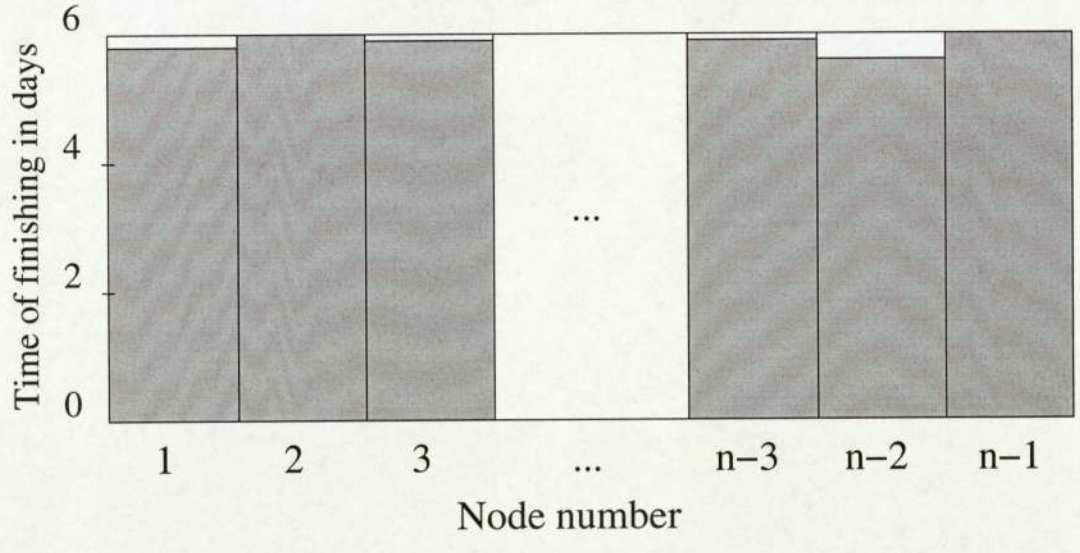


Figure 2.4: In the new configuration, the wastage of time is reduced to a minimum. Parameters are the same as the previous graph.

Chapter 3

Belief propagation on large graphs

Like many algorithms that must deal with complicated global functions of many variables, belief propagation exploits the manner in which the given functions factorise into a product of local functions, on approximation, each of which depends on only a subset of these variables. We now show the origin of belief propagation.

3.1 A probabilistic approach: the Sum-Product Algorithm

Let us consider a bi-partite graph like in Figure 3.1, built with *variables* and *function nodes.*

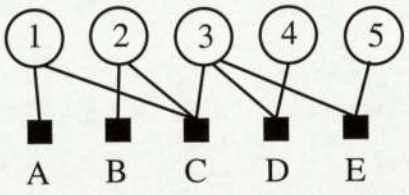


Figure 3.1: Here, a square, called a function node, represents a function of the variables to which it is connected to. The figures represent the variables: $\{x_1, \ldots, x_5\}$.

To each *function node*, e, is associated its set of *variables*, denoted X_e. Moreover, we suppose that all the variables are defined on the same set A, which could be $\{1, \ldots, q\}$. We defined then a constraint function, Z, for each function node, e:

$$Z_e(X_e) = \sum_{t \in A} \left(\prod_{x_i \in X_e} \delta_{x_i,t} \right).$$

Z_e represents the state of the function node e: satisfied, zero, or not, one. The goal of the algorithm

is to solve:

$$\min_e \sum_e Z_e(X_e)$$

That is to say, we want to find the variables assignment such that the maximum number of function nodes are satisfied.

If x is a variable, its neighbours belong to $n(x)$, the set of function nodes in which x is implied. Knowing that $\eta^a_{x\to e}$ is the message sent by a variable to a function node, we then write the equations:

$$\begin{aligned}
\eta^a_{x\to e} &\equiv P(x=a|\overbrace{\{Z_f\}_{f\in n(x)\setminus e}}^{\mathcal{Z}}) \\
&= \frac{P(\mathcal{Z}|x=a)P(x=a)}{P(\mathcal{Z})} \\
&= \frac{P(x=a)}{P(\mathcal{Z})} \prod_{f\in n(x)\setminus e} P(Z_f|x=a),
\end{aligned} \tag{3.1}$$

by applying Bayes' rule and then making the assumption that all the $\{Z_f\}_{f\in n(x)\setminus e}$, denoted $\mathcal{Z}$ for clarity, are independent. This assumption comes from the fact that the graph is locally considered like a tree.

Meanwhile, $\eta^a_{e\to x}$ being the message sent by a function node to a variable,

$$\begin{aligned}
\eta^a_{e\to x} &\equiv P(Z_e|x=a,\mathcal{Z}) \\
&= \sum_{\{y\}_{y\in X_e\setminus x}} P(Z_e,\overbrace{\{y\}_{y\in X_e\setminus x}}^{\mathcal{Y}}|x=a,\mathcal{Z}) \\
&= \sum_{\{y\}_{y\in X_e\setminus x}} P(Z_e|x=a,\mathcal{Y},\mathcal{Z})P(\mathcal{Y}|\mathcal{Z}) \\
&= \sum_{\{y\}_{y\in X_e\setminus x}} P(Z_e|x=a,\mathcal{Y},\mathcal{Z}) \left(\prod_{y\in X_e\setminus x} P(y=a|\mathcal{Z})\right),
\end{aligned} \tag{3.2}$$

by applying the sum-rule, the product-rule and making the assumption that the $\{y\}_{y\in X_e\setminus x}$, denoted $\mathcal{Y}$ for clarity, are independent, with the same approximation as above with the $\{Z_f\}_{f\in n(x)\setminus e}$. By identifying the messages as their probabilities in Equations (3.1) and (3.2), we get:

$$\eta^a_{x\to e} = \frac{P(x=a)}{P(\mathcal{Z})} \prod_{f\in n(x)\setminus e} \eta^a_{f\to x}, \tag{3.3}$$

$$\eta^a_{e\to x} = \sum_{\{y\}_{y\in X_e\setminus x}} P(Z_e|x=a,\mathcal{Y},\mathcal{Z}) \left(\prod_{y\in X_e\setminus x} \eta^a_{y\to e}\right). \tag{3.4}$$

Similar approaches can be found in [30] and [33].

3.2 An application: the graph colouring algorithm

To apply these update rules to our problem, we need to clarify a few things. In the case of graph colouring, one has to replace the definition set of the variables by the set of available colours, here, $\{1, \ldots, q\}$. Let us remind here that the constraint to satisfy is that not all the vertices connected to the same edge have the same colour. Actually, an edge will send to one of its vertices, say v, given a colour, μ, the probability that this edge is unsatisfied if v takes this colour μ. This situation can occur only if all the other vertices except v have this colour, which means that there is only one configuration of the variables for which the factor in the sum is not 0. Hence, we eliminate the sum over all the possible configurations in (3.4). Consequently, the message sent by a vertex to one of its edge, say e, given a colour μ, will be the product of the probabilities that it will take the colour μ, i.e. the product over the other edges of one minus the probability that the other edge will forbid this vertex to take this colour μ. Since for a vertex and one of its edges, the sum over all the colours of the outgoing messages is one, we need to normalize each one of these probabilities. For numerical reasons, we add a Δ factor to the messages sent by an edge. $\Delta = 1 - e^{-\beta}$ where $\beta = 20$ represents the inverse temperature of the system, hence, this factor is very close to 1. According to these transformations, one finally obtains the following formulae:

$$\eta^{\mu}_{e \to v} = \Delta \prod_{w \in \nu(e) \setminus v} \eta^{\mu}_{w \to e}, \tag{3.5}$$

$$\eta^{\mu}_{v \to e} = \frac{\prod_{f \in \epsilon(v) \setminus e} (1 - \eta^{\mu}_{f \to v})}{\sum_{d=1}^{q} \prod_{f \in \epsilon(v) \setminus e} (1 - \eta^{d}_{f \to v})}. \tag{3.6}$$

These are the formulae used in the belief propagation algorithm. One can find an implementation of this algorithm in Appendix A. These messages will also be called cavity magnetisations.

3.3 The cavity rule

As it can be noticed, all the messages sent to a neighbour (whether it is an edge or a vertex) always take into account only the messages received from all its neighbour except from the one which it is sending the message to. That is called the cavity rule.

Apart from the formulae, there is another reason for this. It is due to the fact that message-passing has been first used for directed graphs tree-like and that colouring deals with undirected ones containing loops.

In a directed graph, from the messages it received from its parents, a vertex computes those to

send to its children. But in undirected graphs, it sends to its neighbour a message directly related to the one it received from the same neighbour. In other words, a vertex chooses a state not only regarding to what its neighbours say but also from the state it previously opted for.

To avoid this short and disturbing feedback, the idea is to create a cavity around the vertex we send a message to, when we compute the message. However, the independence between the messages a vertex sends and the state it opted for before is still not total because two vertices can be second or third neighbours. But as the average length of loops grows with the graph size, it is reasonable to think that this feedback can be neglected for large enough graphs.

3.4 A brief description of statistical physics

The fact that there is a growing success in the application of statistical physics techniques to computational complexity problems drives to the perception that these techniques could be accordingly used in a wide range of computational complexity tasks, one of which is the graph colouring problem.

In [40], van Mourik and Saad map the graph colouring, with p-colours, onto the anti-ferromagnetic p-spin Potts model [41]. There, the replica symmetric approximation is used to obtain physical quantities from which important questions about the colourability of a graph can be answered.

The statistical physics approach is based on two main steps. The first is the introduction of a Hamiltonian or cost-function and the second is the calculation of the typical free energy in the large system limit. By employing thermo-dynamic relations, the calculation of the free energy provides the typical ground state energy, which in turn gives the opportunity to predict the graph's colourability[1]. A non-zero ground state energy indicates that, under the given conditions, random graphs are typically not colourable.

Another aspect of the ground state energy, apart from just determining the colourability of a graph, is that it reveals the typical minimal fraction of unsatisfied edges when the graph is non-colourable. On the other hand, the ground state (residual) entropy provides information on the number of different colouring schemes that share the minimum number of unsatisfied edges.

The free energy turns out to be a functional of the distribution of the cavity magnetisations. By analogy with equations (3.5) and (3.6), the meaning of $\eta^{\mu}_{e\to v}$ and $\eta^{\mu}_{v\to e}$ in the formulae above are the same. Then the free energy is extremized with respect to the distributions of these cavity

[1] A graph would be colourable if its free energy is zero.

magnetisations in order to obtain the physical free energy. This extremization process gives the update rules for the cavity magnetisations. The details of this calculation are beyond the scope of this thesis. Replacing, in the Replica Symmetric Approximation, the integrals over the distribution $\pi(\vec{\eta}_{v\to e})$ and $\hat{\pi}(\vec{\eta}_{e\to v})$ by sums over the corresponding microscopic cavity magnetisations, we obtain the equations found with the mathematical approach.

From [40], we obtain the Replica Symmetric free energy per edge, $\mathcal{F}_e$:

$$\mathcal{F}_e = \frac{1}{\beta}\frac{1}{N_e}\left(N_v \sum_{\mu\in[\![1,p]\!]} f_\mu \hat{f}_\mu + G_1 - G_2 - G_3\right), \tag{3.7}$$

where :

$$\begin{aligned}
G_1 &= \sum_{v\in V}\sum_{e\in\varepsilon(v)} \log\left(1 - \sum_{\mu\in[\![1,p]\!]} \eta^\mu_{v\to e}\eta^\mu_{e\to v}\right),\\
G_2 &= \sum_{e\in E} \log\left(1 - \Delta \sum_{\mu\in[\![1,p]\!]}\prod_{v\in\nu(e)} \eta^\mu_{v\to e}\right),\\
G_3 &= \sum_{v\in V} \log\left(\sum_{\mu\in[\![1,p]\!]} e^{\hat{f}_\mu}\prod_{e\in\varepsilon(e)} (1-\eta^\mu_{e\to v})\right),
\end{aligned}$$

with E and V are the set of edges and vertices respectively, β the inverse temperature, $\Delta = 1 - e^{-\beta}$ the previous factor, $\nu(e)$ the set of vertices of edge e and $\varepsilon(v)$ the set of edges of vertex v.

Note the presence of f_μ, the actual fraction of vertices of colour μ, and $\hat{f}_\mu$, the corresponding chemical potential, for the constrained fraction of vertices. These can be useful in forcing the use of a colour rather than another, but although implemented, it has not been used (we take $\forall\mu, \hat{f}_\mu = 0$). Then the ground state energy per edge and the entropy per vertex are:

$$E_{0,e} = \lim_{\beta\to\infty}\frac{1}{N_e}\sum_{e\in E}\frac{e^{-\beta}\sum_{\mu=1}^{p}\prod_{v\in\nu(e)}\eta^\mu_{v\to e}}{1-\Delta\sum_{\mu=1}^{p}\prod_{v\in\nu(e)}\eta^\mu_{v\to e}}, \tag{3.8}$$

$$S_{0,v} = \lim_{\beta\to\infty}\beta\frac{N_e}{N_v}(E_{0,e} - \mathcal{F}_e). \tag{3.9}$$

In principle β should be infinite but, for numerical reasons, β will take the value 20.

If we consider, that we have a mixture of $1-\alpha$ edges linked to 2 vertices and α edges linked to 3 vertices, the formulae for the paramagnetic state are :

$$\begin{aligned}
E_{o,e,pm} &= (1-\alpha)\frac{e^{-\beta}}{p^{K_{min}-1}-\Delta} + \alpha\frac{e^{-\beta}}{p^{K_{max}-1}-\Delta},\\
\mathcal{F}_e &= \frac{1}{\beta}\left(\frac{\lambda k_{avg} - \lambda - k_{avg}}{\lambda}\log(p) - (1-\alpha)\log(p^{K_{min}-1}-\Delta) - \alpha\log(p^{K_{max}-1}-\Delta)\right).
\end{aligned}$$

The details of this calculation can be found in Appendix E.

The study of the paramagnetic state is important in finding λ_d. The paramagnetic state is an obvious solution to the Replica Symmetric iterative equations. For small average connectivities, it is the only one. The appearance of a nontrivial solution coincides with a clustering transition of ground states into an exponentially large number of extensively separated clusters [4]. In spin-glass theory, this transition is called *dynamical.* As soon as the paramagnetic state is not the only stable solution, we are above λ_d.

3.5 Results

To validate the implementation of belief propagation, we have plotted the behaviour of these formulae in Figures 3.2 and 3.3.

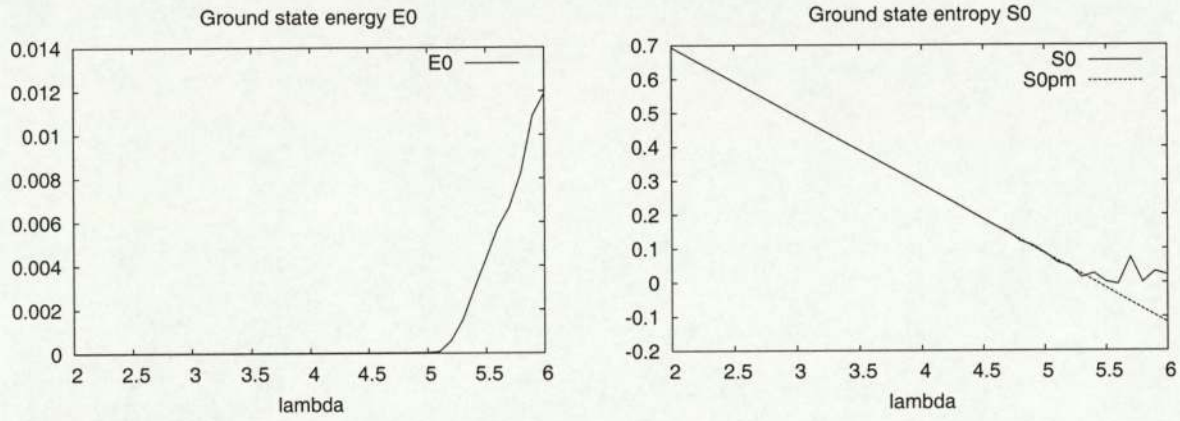


Figure 3.2: $N_v = 1000$, $q = 3$ and $\alpha = 0$. Left : the ground state energy $E_0(\lambda)$ remains 0 until $\lambda \simeq 5.1$, whereas the paramagnetic ground state always predicts 0. Right : the ground state entropy $S_0(\lambda)$ coincides with its paramagnetic value $S_{0,pm}$ up to $\lambda \simeq 4$.

For $\alpha = 0$, one can note the agreement with [40]. The limit of λ above which the ground state energy becomes positive corresponds to an upper bound of the critical connectivity, since $E_{0_{RS}}$ is a lower bound for the true ground state energy and even when E_0 equals zero, it does not guarantee colourability. One can then summarize them in Table 3.1.

An issue appears here: the accuracy of K_{avg} of a graph and its consequences on λ. Actually, the graph generation will create a graph whose K_{avg} comes close to its theoretical value but due to the approximation while calculating the number of edges, there is a small difference. This is true only for α different from 0 and 1. Since N_v, N_e and K_{avg} will stay fixed, the parameter which will undergo

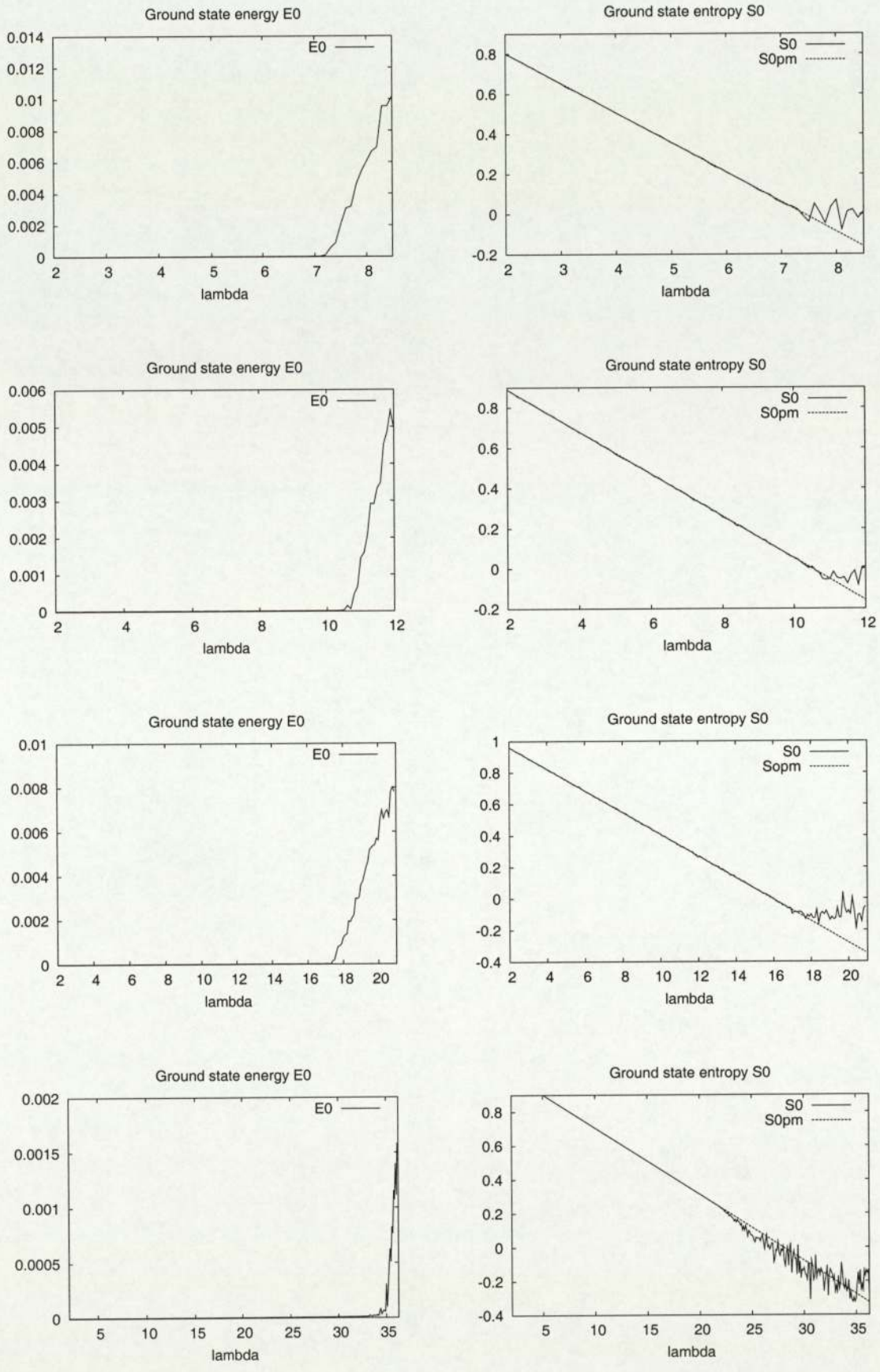


Figure 3.3: The graphs here have the same behaviour as for $\alpha = 0$. The graphs are sorted in the increasing order of $\alpha \in \{0.25, 0.5, 0.75, 1\}$. $N_v = 1000$, $q = 3$.

α	λ_{up}
0	5.1
0.25	7.1
0.5	10.4
0.75	17
1	33

Table 3.1: Upper bound of λ_c for each α.

the approximations is λ. Hence the less than perfect match between the paramagnetic entropy and the entropy is expected, as it can be seen for α equals to 0 and 1 in Figures 3.2 and 3.3.

However, comparing these with known results ([40, 18]) for α equal to zero validates results obtained by the belief propagation algorithm.

Another type of statistics that can be used is the distribution of the messages sent by the vertices. Depending on the value of the connectivity with respect to the critical value, this distribution takes different shapes (see Figure 3.4). Those shapes are similar for all the values of α with adapted connectivities.

3.6 The paramagnetic state

An interesting task is to plot the phase diagrams (λ, T) for different values of α. The goals of this operation is firstly to categorize the phase transition, by focusing on points at which derivatives diverge and secondly to predict in which state the system will be given the two parameters.

All the transitions are second order in the distribution of the cavity magnetisations. This means that the probability distribution goes continuously from a single peak at $1/3$, below the transition point, λ_d, to a probability distribution still containing this peak but lower and with probabilities spread at the base of the peak just after the transition point, $\lambda > \lambda_d$. In Figure 3.4, for $\alpha = 0.5$ and $\lambda = 8.1$, we see something continuously appearing at the base of the peak, hence the second order transition.

The main information phase diagram can give with relation to graph colouring is the lower bound of λ_c, i.e. the dynamical connectivity λ_d. As explained previously in Section 3.4, as soon as the paramagnetic state is not the only stable solution, we are above λ_d. At the transition point, we have to suddenly increase the temperature to help the system to converge to the paramagnetic state, which

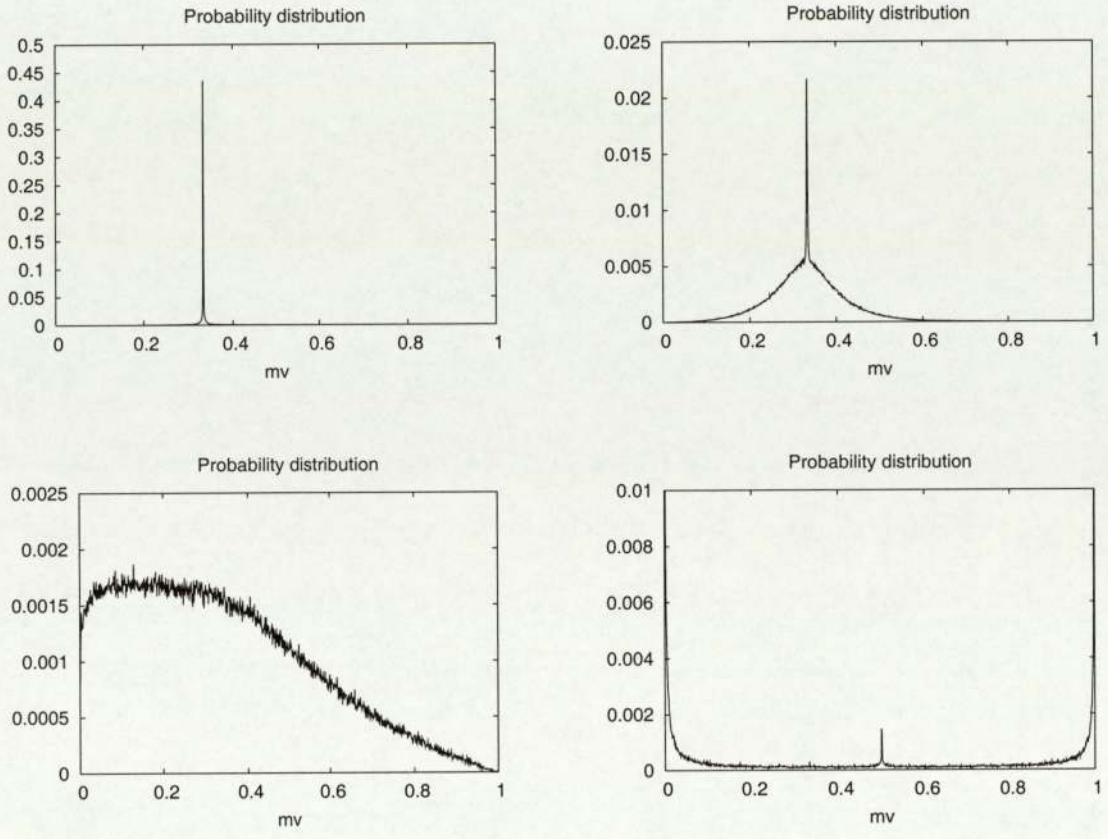


Figure 3.4: $N_v = 1000$, $\alpha = 0.5$ and $q = 3$. On the upper part: the stationary distribution for $\lambda = 8.1$ ($\lambda_d \lesssim \lambda$) on the left and $\lambda = 8.7$ ($\lambda_d < \lambda < \lambda_c$)on the right. The absence of peaks near $mv \simeq 0, 1$ indicates that $E_0(\lambda) = 0$. On the lower part: the stationary distribution for $\lambda = 9.9$ ($\simeq \lambda_c$) on the left and $\lambda = 12$ ($> \lambda_c$) on the right. Note the peak at $mv = 0.5$ and the peaks and non-trivial distribution at $x \simeq 0$ and $x \simeq 1$, indicating that many vertices are forced (not) to take a specific colour, and that $E_0(\lambda) > 0$.

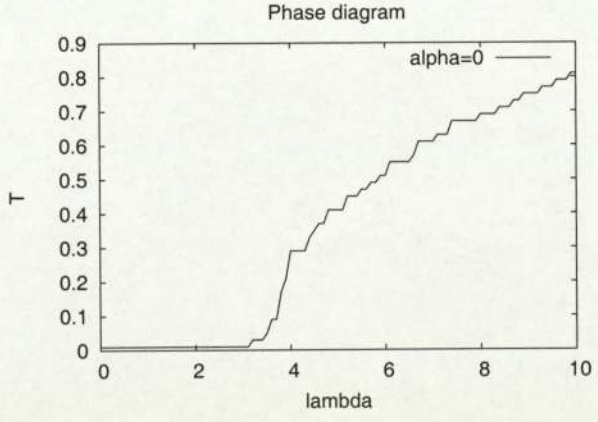


Figure 3.5: $N_v = 1000$, $q = 3$ and $\alpha = 0$. The line separates the paramagnetic state (above the line) from the non-paramagnetic state (below the line).

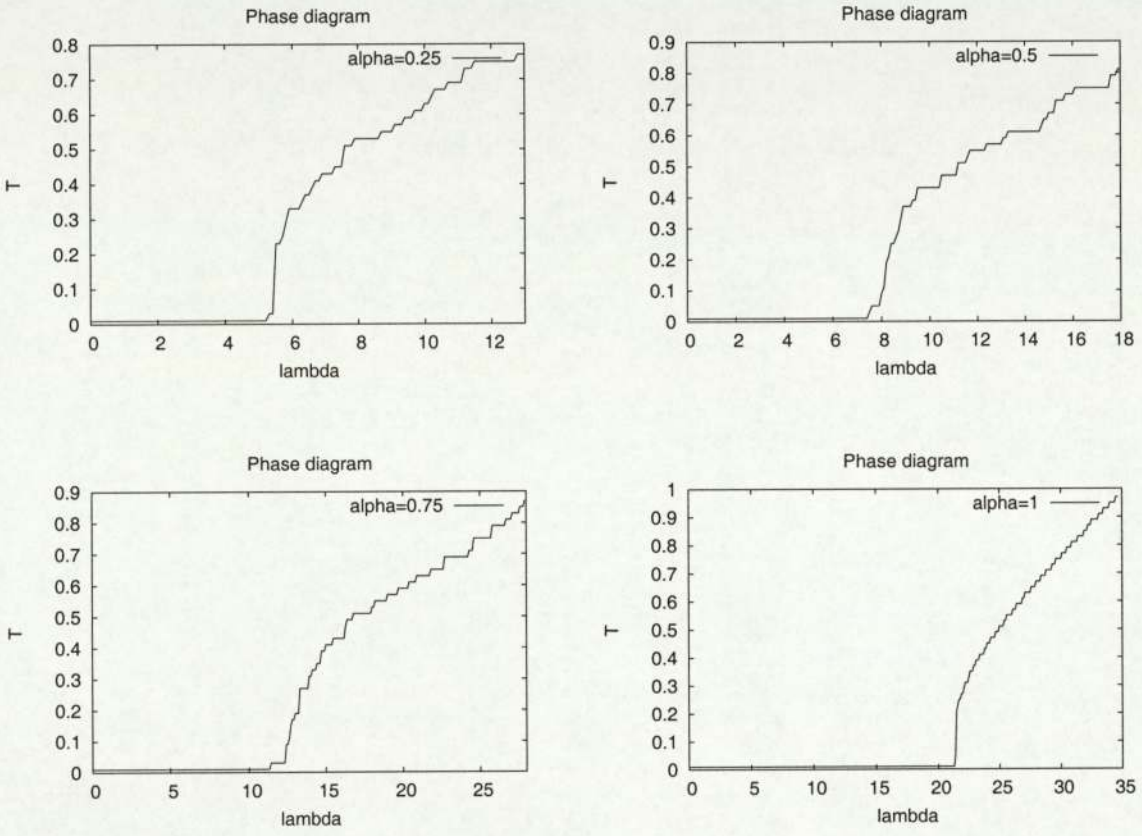


Figure 3.6: $N_v = 1000$ and $q = 3$. The line separates the paramagnetic state (above the line) from the non-paramagnetic state (below the line).

means, therefore, that the paramagnetic state is in competition with other solutions. The big jump of the temperature at this point explains the spontaneous division into an exponentially large number of clusters, i.e. the *complexity* times the size of the graph (see [4]).

α	λ_d
0	4
0.25	5.5
0.5	8
0.75	12.5
1	21.4

Table 3.2: Estimation of λ_d for each α.

3.7 Time averaging

A major drawback of Belief Propagation is that convergence is not always reached. In fact it almost never converges for large graphs and $\lambda > 4$, hence the need for a catalyst, time averaging.

3.7.1 Reasons for non convergence

Generally, a colouring of a graph is not unique. Let us suppose that a solution is found, called C_1. All the permutations among the colours, for example replacing c_i by c_j and vice versa, also gives a new and different colouring which is still a solution. This is illustrated in the previous section when the ground state entropy is positive.

Let us now assume we have a large graph. Since the initialization of the messages is done randomly, it is unlikely that this allocation is close to the solution for the whole graph. Actually, it may happen that in one part of the graph, Belief Propagation will tend to converge to a solution, C_i, and in the other parts it may converges to different colourings, $\{C_j, \ldots, C_n\}$. Among these other colourings, some might be compatible with C_i and some may not. As for connectivities close to the critical connectivity, the graph is a single cluster and then all these parts are linked. Each of these parts will try to convince the others to opt for its colouring and they might all settle for the same one. However, the larger the graph, the more chances there are that this process continues indefinitely, hence the non convergence.

3.7.2 Time averaging as a solution

Time averaging is a very powerful tool used in a wide variety of problems, like noise control, non-linear oscillations or stability analysis.

This tool is employed after few iterations of Belief Propagation when convergence is not reached. Basically, it averages the magnetisation of each vertex over a number of supervised iterations (a time window). Then the vertices are fixed with respect to the average magnetisation instead of the final magnetisation. There is an implementation of this algorithm in Appendix A.1.2.

By averaging the magnetisations over a time window, this algorithm undergoes less of the variations due to non convergence and can find the parts of the graph which have settled for a unique colouring.

The length of the time window is crucial since it has great influence on the performance of the algorithm: a too narrow window is more influenced by short oscillations providing poor results, while a too broad a window results in much higher computational time without a significant gain in accuracy.

From [18], this value is fixed to a value of 30 iterations, which seems to be a good compromise.

3.8 Results

Most of the tests will be performed around the critical connectivity, to measure the efficiency of the time averaging on large random graphs. Three quantities are evaluated: the average percentage of unsatisfied edges per graph, the percentage of perfect colouring and the average running time of the algorithm.

Globally, we note that the behaviour of the graphs are similar for all the values of α (see Figures 3.7, 3.8 and 3.9). For example, for $\alpha = 0.5$ and $\lambda = 10$, the average number of unsatisfied edges is slightly lower than 0.6, which means that, on average, 24 edges among 4000 are not satisfied. Hence, the very good efficiency of Belief Propagation.

An interesting point is to study the q-COL/UNCOL transition. From the graph representing the average number of unsatisfied edges, it is not very clear to see where the transition takes place, since

the curve is quite smooth. This is due to the fact that even if the graph is not colourable, the belief propagation algorithm finds the optimal solution. Thus, we require the use of the second quantity. Let us recall that below the transition, one can find solution with a probability tending to one as the size of the graph goes to infinity and that above the transition, this probability goes to zero. The percentage of perfect colouring can be viewed as this probability. The connectivity for which this percentage becomes zero should be close to the critical connectivity. However, for $\alpha \geq 0.25$, this critical connectivity has not been studied nor stated yet. This is why we dealt with bounds so far. Even so, the values reported in Table 3.3 are compatible with the intervals presented earlier in this chapter.

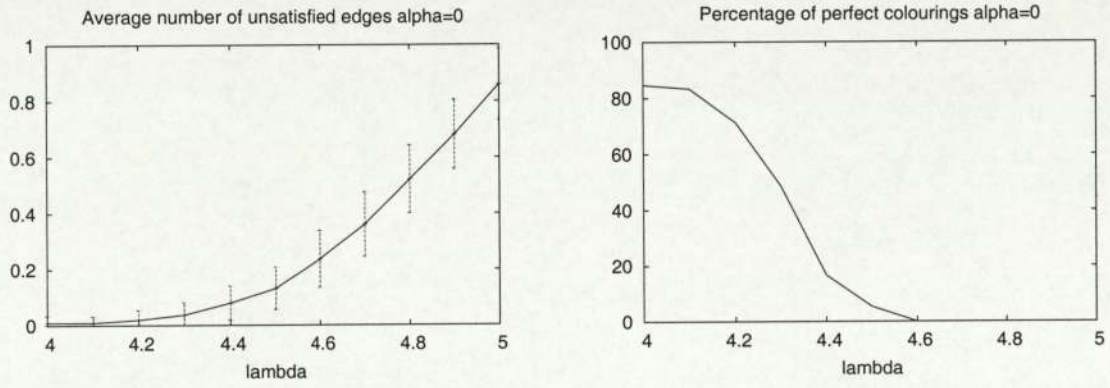


Figure 3.7: Performance of Belief Propagation with time averaging on random graphs. $N_v = 1000$, $q = 3$ and $\alpha = 0$, data averaged over 150 graphs for each connectivity.

α	$\lambda_{c_{est}}$
0	4.6
0.25	6.4
0.5	9.4
0.75	14.7
1	25.6

Table 3.3: Estimated values of λ_c for each α.

As we can see, despite the fact that around the critical connectivity, or the assumed critical connectivity, the problem is known to be NP-hard [22, 29], the computational time does not increase exponentially and actually, stays quite constant. This is due to the low computational cost of the message-passing techniques.

Tests have been performed on random colourable graphs and, as we expected, provide very good

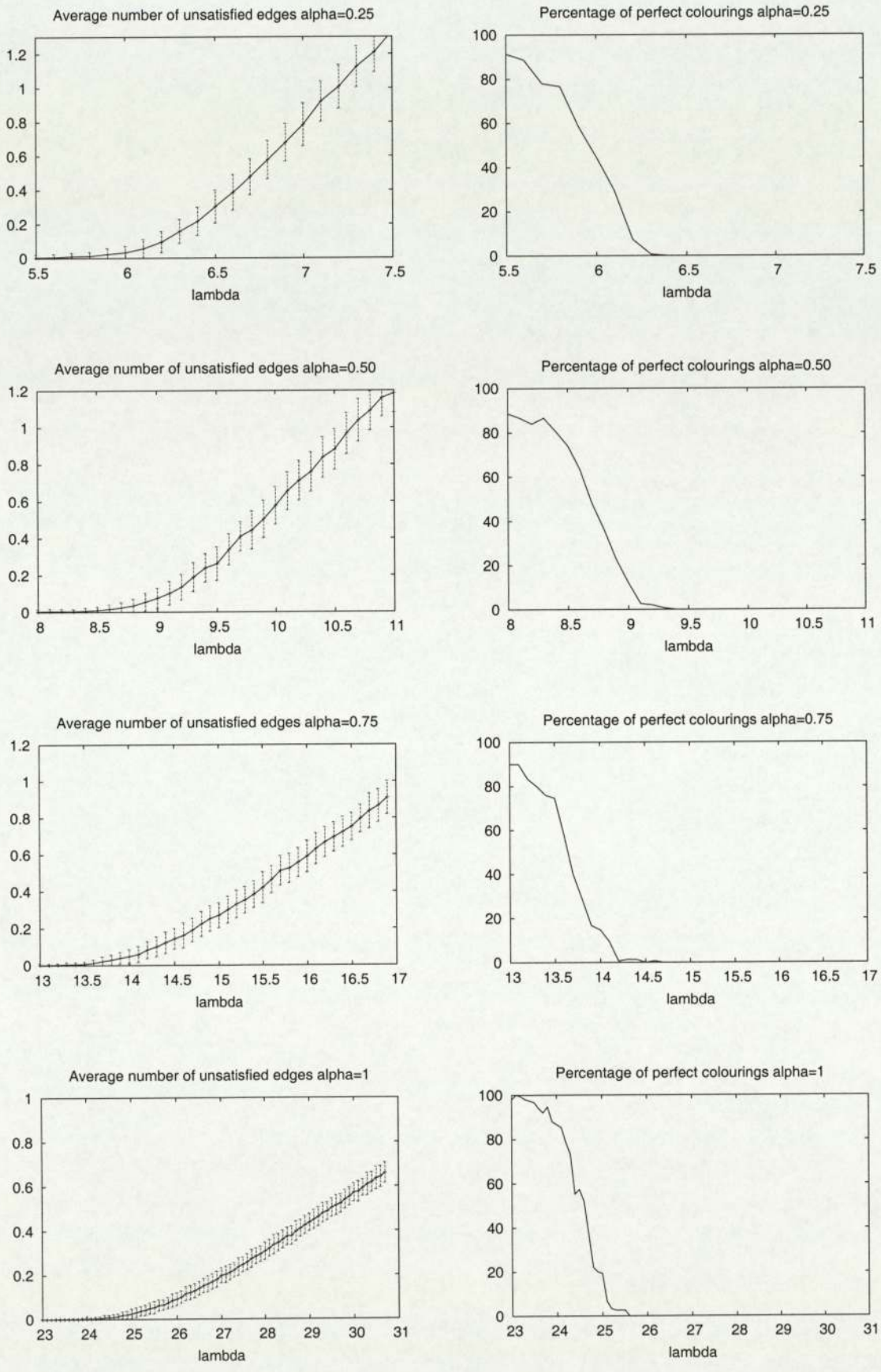


Figure 3.8: Performance of Belief propagation with time averaging on random graphs. $N_v = 1000$, $q = 3$ and $\alpha \in \{0.25, 0.5, 0.75, 1\}$. Data averaged over 150 graphs for each connectivity.

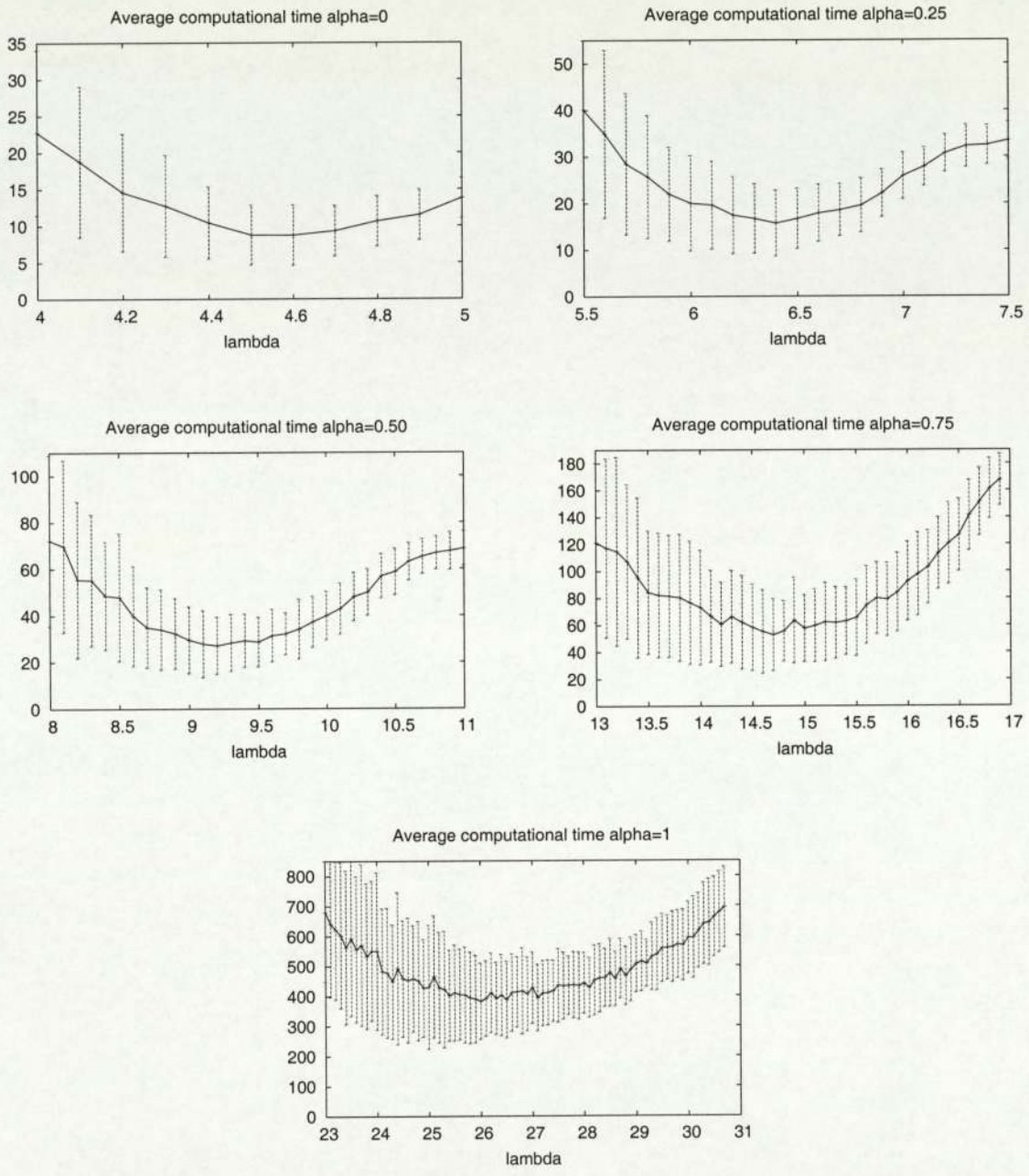


Figure 3.9: Computational Time of Belief propagation. $N_v = 1000$, $q = 3$ for all α. Data averaged over 150 graphs for each connectivity.

results, except for $\alpha = 1$. In that case, it seems that the algorithm behaves with random colourable graphs as it does with random graphs, since the results are very similar (see Figures 3.10 and 3.11). The computational time is less than for the random graphs, except again for the case where $\alpha = 1$. It seems that for α tending to one, the performance of Belief Propagation decreases significantly.

Indeed, tests performed on small graphs show that, for $\alpha = 1$, the percentage of perfect colourings starts at quite high values for a system size of 30 vertices and decreases quickly to go to zero for graphs of 500 vertices, passing through medium values at 50 vertices.

Note the high value of the standard deviation. This is due to the fact that there are very few values other than zero and the fact that the average number of unsatisfied edges can only take predefined values.

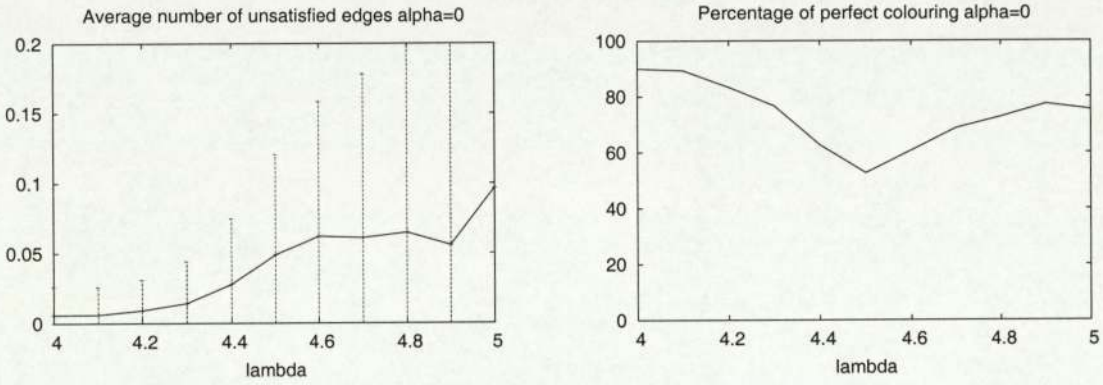


Figure 3.10: Performance of Belief Propagation with time averaging on random colourable graphs. $N_v = 1000$, $q = 3$ and $\alpha = 0$, data averaged over 200 graphs for each connectivity.

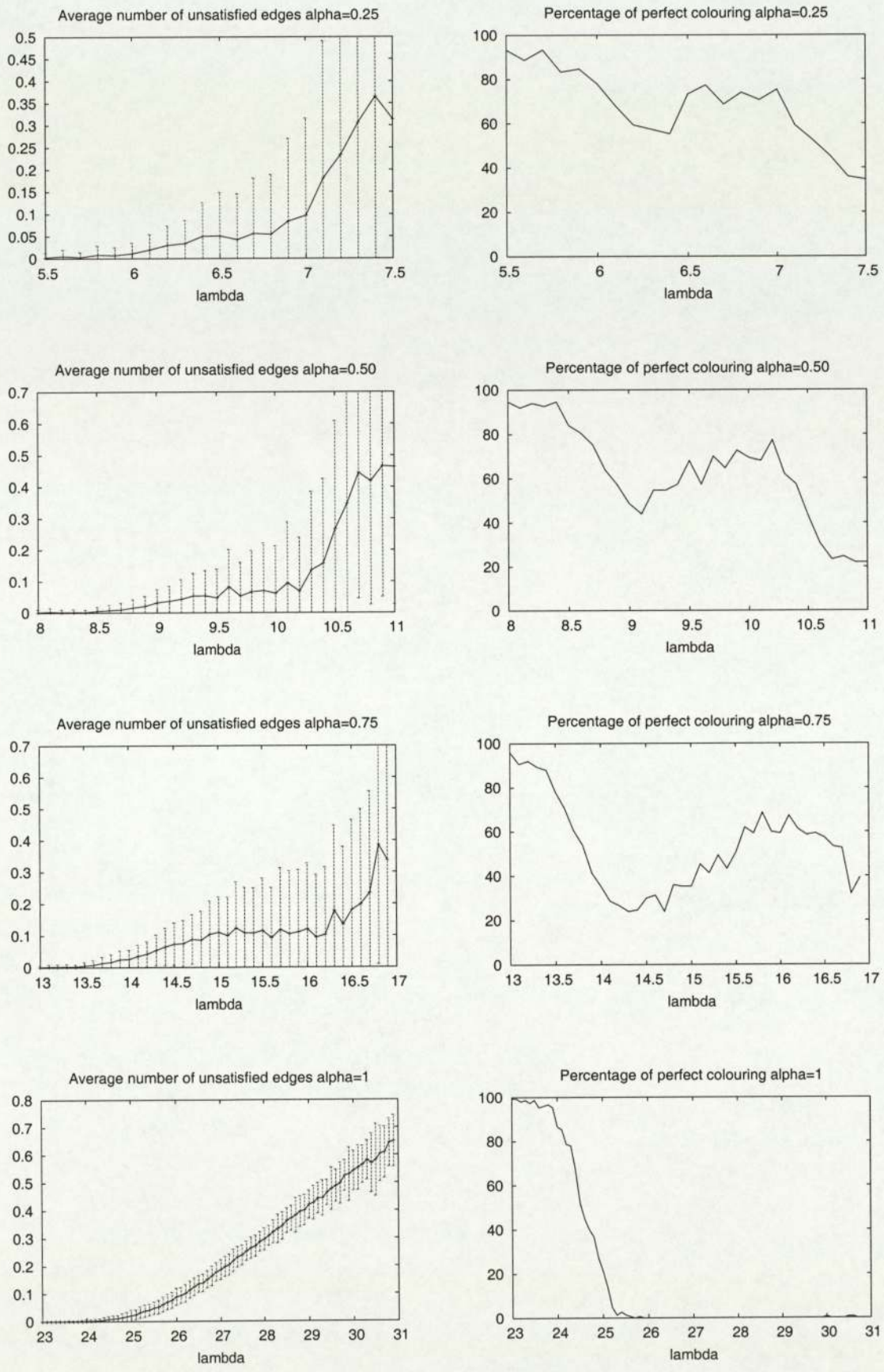


Figure 3.11: Performance of Belief propagation with time averaging on random colourable graphs. $N_v = 1000$, $q = 3$ and $\alpha \in \{0.25, 0.5, 0.75, 1\}$. Data averaged over 200 graphs for each connectivity.

Chapter 4

Exact enumeration on small graphs

4.1 Backtrack algorithm

4.1.1 Aims

In order to find the critical connectivity in the cases where $\alpha \in \{0.25, 0.5, 0.75, 1\}$, exact enumeration is the only algorithm which can state with certainty whether a graph is colourable or not. Basically, this algorithm will enumerate all the possible colourings for a given graph. Since, in general, the number of possible colourings examined grows exponentially with the size of the system, we are limited by the accessible system sizes (i.e. $N_v \simeq \mathcal{O}(50)$) and may expect considerable finite size effects.

The idea is to plot the probability that a graph is colourable with respect to the connectivity. To do so, for a given sample of 10^3 graphs, we calculate the percentage of perfect colourings. From [27], we have two important pieces of information:

1. for a given connectivity λ_1, if the percentage of perfect colouring is better, respectively worse, for a larger system size, that means that $\lambda_1 \leq \lambda_c$, respectively $\lambda_1 \geq \lambda_c$.

2. the convergence time, as a function of the connectivity, diverges around the critical connectivity.

The second point can also be justified by the fact that around the critical connectivity, there are but very few solutions, hence the long time needed to find a colouring.

4.1.2 Some catalysts for exact enumeration

To reduce the time of execution, it is helpful to use a few catalysts.

As with Belief Propagation, it may be useful to prune the graph. Even if it will not prune more than three or four vertices for high connectivities, it will still save time.

A second tool is early stopping, implemented by [40]: one starts colouring the remaining vertices, keeping track of the remaining available colours per vertex for all the uncoloured vertices. One then stops exploring the search tree as soon as a good colouring is reached, since the goal here is to know the colourability of the graph. Otherwise, when the number of available colours for a vertex becomes zero, the colouring so far will lead to a contradiction later on, hence the abandoning of this branch of the search tree. One then backtracks to the point where a colouring was still possible.

These methods greatly reduce the actual number of colourings to explore, however, remaining exponential in the system size, thus greatly limiting the accessible system size. Furthermore, since we stop as soon as we encounter a contradiction, we have no information on the minimum number of unsatisfied edges (i.e. the ground state energy) or the number of schemes that yield the minimum number of unsatisfied edges (i.e. the residual entropy).

4.1.3 Results

Due to the lack of time for the project, the system size is chosen between 30 and 50, since it is better to do more simulations of small graphs rather than few simulations on larger graphs.

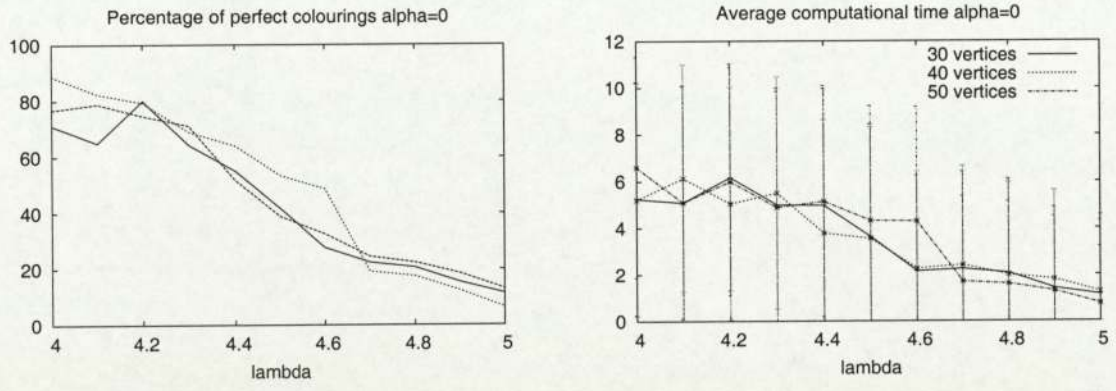


Figure 4.1: Performance of Exact enumeration on random graphs. $N_v \in \{20, 30, 40, 50\}$, $q = 3$ and $\alpha = 0$, data computed over 10^3 graphs for each connectivity.

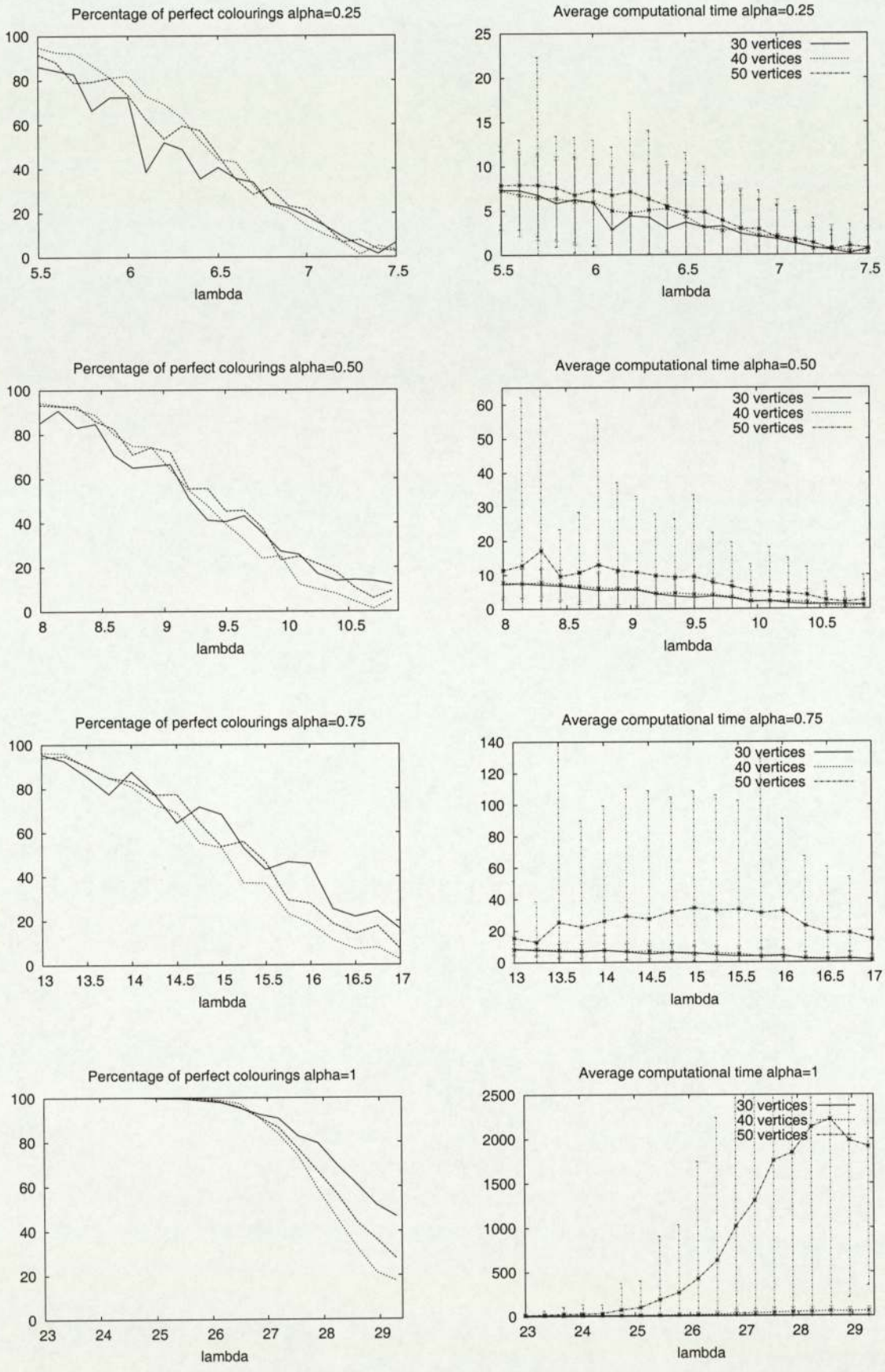


Figure 4.2: Performance of exact enumeration on random graphs. $N_v \in \{20, 30, 40, 50\}$, $q = 3$ and $\alpha \in \{0.25, 0.5, 0.75, 1\}$. Data computed over 10^3 graphs for each connectivity.

Unfortunately, it seems that the number of graphs is not enough to make up for the finite size effects. However, it is possible to determine boundaries for the critical connectivities (see Table 4.1).

α	λ_{lo}	λ_{up}
0	4.5	4.8
0.25	6.1	6.7
0.5	9.3	9.8
0.75	14	14.5
1	26.5	27

Table 4.1: Estimated values of λ_c for each α with exact enumeration.

4.2 Belief propagation versus exact enumeration

By modifying the exact enumeration algorithm, it is possible to compare with Belief Propagation. Of course, Belief Propagation cannot perform better than exact enumeration, but it will give an idea on how efficient Belief Propagation is on small graphs.

For clarity of reading, we show here the results obtained for $N_v = 30$, since the other simulations are similar. Appendix F provides the results for N_v equal to 20 and 40.

Concerning the percentage of unsatisfied edges, it seems that the distance between Belief Propagation and Exact Enumeration curves is a constant with respect to λ, c_α. Furthermore, it seems that this constant is decreasing while α is increasing. Note that the standard deviations are generally larger for the Belief Propagation algorithm than for exact enumeration.

Concerning the percentage of perfect colourings, after the upper bound found previously for the critical connectivity, the two algorithms behave very similarly.

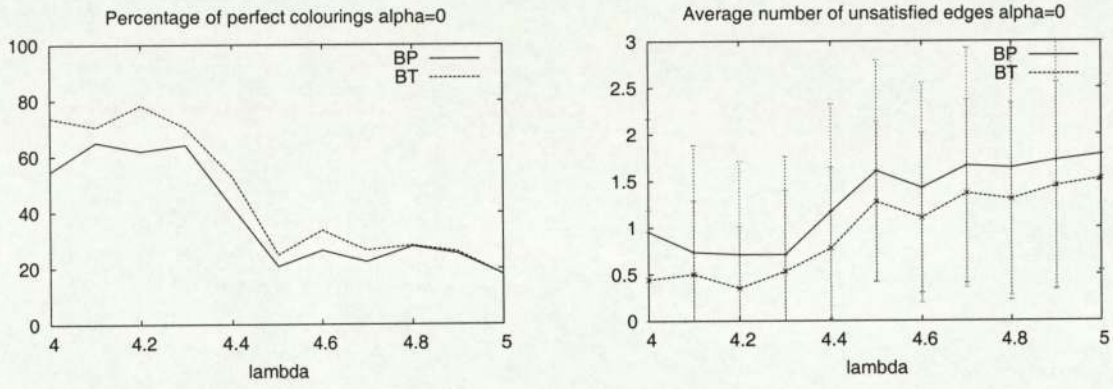


Figure 4.3: Performance of Belief Propagation on random graphs compared with Exact Enumeration. $N_v = 30$, $q = 3$ and $\alpha = 0$, data computed over 10^3 graphs for each connectivity.

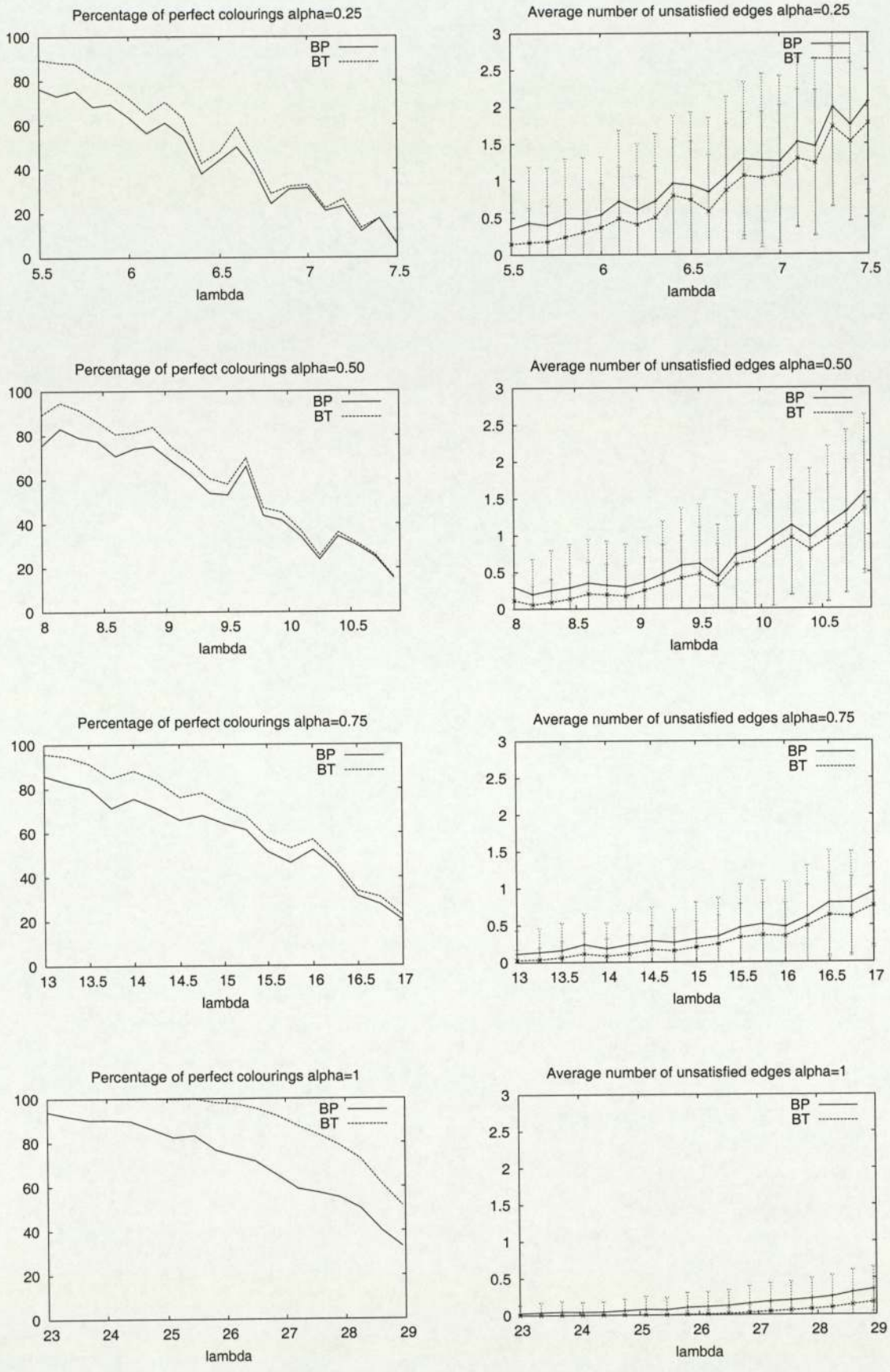


Figure 4.4: Performance of exact enumeration on random graphs compared with Belief Propagation. $N_v = 30$, $q = 3$ and $\alpha \in \{0.25, 0.5, 0.75, 1\}$. Data computed over 10^3 graphs for each connectivity.

Chapter 5

Conclusion

5.1 Achievements

In this research period, we extended the implementation of a recent algorithm to a new area: the area of hypergraphs. Different types of hypergraphs have been studied: some with three body-edges and some with a mixture of two and three body-edges. Statistical physics has been tested on these kinds of graphs. It has shown good results and we managed to predict boundaries for the critical connectivities (see Chapter 3). But we also noticed that for three body-edges graphs, we probably reached a limit in the Belief Propagation efficiency.

Its comparison with a deterministic algorithm, which is also something new, reinforced the idea that this algorithm, based on a heuristic method, is trustworthy (see Chapter 4). This result is of great importance since this algorithm is fast and independent of the connectivity of the graph.

The last contribution is the parallelization of the test procedures which is a huge source of time saving (see Chapter 2). Other tools have been adapted as random graph generators or graph pruning.

Unfortunately, due to the lack of time for this project, the dense graphs study has not been achieved. Nevertheless, dealing with dense graphs should highlight that Belief Propagation is an approximate algorithm. When we derived the equations, we made the assumption that the graph was locally cycle-free, which is true for sparse graphs. If we consider a graph whose density is 0.5, which means that a vertex will be on average linked to half of the remaining vertices, it is very probable to

find a loop after two neighbours and then the approximation as above may break down.

5.2 Further work

Exact enumeration is time-consuming. Besides, due to the finite size effects, the results were not clear enough to state with a reasonable certainty what the values of the critical connectivity are. Hence there is a need for more simulations on larger systems.

Due to a problem of implementation, Survey Propagation has not been tested. However, the equations have been constructed, see Appendix A.3. It would be very interesting to compare these two similar algorithms since they are novel and use similar technique.

Since these two algorithms rely on message-passing technique, they are suitable for parallelization. It would allow us to use larger systems ($N_v \simeq 10^6$).

It could be interesting to add constraints on the colours, that is to force one colour to be taken with the higher priority, by changing the chemical potential of the colours. An application to that is the maximal independent set of a graph. One could also have a different set of available colours for each vertex.

As long as the graph colouring is not solved, there is still ongoing research to do in this area.

Appendix A

Pseudo-code of Belief Propagation

A.1 Message-passing technique

Tensors `mv` and `me` denote the messages sent from a vertex to an edge and those from an edge to a vertex. The functions and reserved words start with a capital letter whereas the variables are written in small letters.

A.1.1 Iterate simply

`vol` is the variation of the values of the messages, always positive. When it is too low, this means that the algorithm converges.

```
function Iterate(nmax){
For k = 1 to nmax Do
  items = list(vertices,edges)
  While (items is not Empty) Do
    Pick a random element el in items
    If (el is a vertex) vol += UpdateVertex(el)
    Else UpdateEdge(el)
    Remove el from items
  EndWhile
  If (vol is too low) Break
EndFor
}
```

A.1.2 Iterate and calculate the magnetisation

This algorithm is basically similar to the previous one. The difference is the use of the time averaging. `mag` is the magnetisation matrix filled with zeros at the beginning and `nv` is the number of vertices. The `magnetisation` function is the same as the update of a vertex without the cavity rule.

```
function Calculate(nmax){
Initialize(mag)
For k = 1 to nmax Do
  items = list(vertices,edges)
  While (items is not Empty) Do
    Pick a random element el in items
    If (el is a vertex) vol += UpdateVertex(el)
    Else UpdateEdge(el)
    Remove el from items
  EndWhile
  For j = 1 to nv Do
    For mu = 1 to number_colours Do
      mag[nv][mu] += magnetisation(nv,mu)
    EndFor
  EndFor
  If (vol is too low) Break
EndFor
For j = 1 to nv Do
  For mu = 1 to number_colours Do
    mag[nv][mu] /= k
  EndFor
EndFor
}
```

A.1.3 Fix

This algorithm takes `threshold` as argument.

```
function Fix(threshold){
```

```
magmax = 0
kmax = 1
is_fixed = 0
For k = 1 to nv Do
  For mu = 1 to number_colours Do
    If (mag[k][mu] > threshold)
    Then
      Fix(k)
      is_fixed = 1
    EndThen
    Else
      If (magmax < mag[k][mu])
      Then
        magmax = mag[k][mu]
        kmax = k
      EndThen
    EndElse
  EndFor
EndFor
If (is_fixed = 0) Fix(kmax)
}
```

A.1.4 Iterate and fix

This algorithm works for given maximum number of free iteration `nmax`, maximum number of iterations with calculation `ncalc`, given the fixing `threshold`.

```
While (it remains unfixed vertices) Do
  Iterate(nmax)
  Calculate(ncalc)
  Fix(threshold)
EndWhile
```

A.2 Belief propagation updates

A.2.1 Vertex update

```
function UpdateVertex(v){
vol = 0
For all edges e connected to v
    norm = 0
    tp = mv[v][e][1]
    For mu = 1 to number_colours Do
      mv[v][e][mu] = exp(f_colours[mu])
      For all edges f connected to v apart from e Do
        mv[v][e][mu] *= 1 - me[f][v][mu]
      EndFor
      norm += mv[v][e][mu]
    EndFor
    vol += abs( tp - mv[v][e][1]/norm )
    For mu = 1 to number_colours Do
      mv[v][e][mu] /= norm
    EndFor
EndFor
}
```

A.2.2 Edge update

```
function UpdateEdge(e){
For all vertices v connected to e Do
  For mu = 1 to number_colours Do
    me[e][v][mu] = delta
    For all vertices u connected to e apart from v Do
      me[e][v][mu] *= mv[u][e][mu]
    EndFor
  EndFor
EndFor
```

}

A.3 Survey propagation

A.4 Presentation

Survey Propagation was developed by Braunstein, Mézard and Zecchina for solving the satisfiability problem [6] and adapted soon after to graph colouring [4, 5].

To help Belief Propagation to converge in the hard parameter region, we used time averaging. But, from the statistical point of view, Braustein et *al.* showed that the Replica Symmetric approximation is no longer valid in this region, and they have introduced the Survey Propagation algorithm based on the 1-step Replica Symmetry Breaking for the exact ground state. It is beyond the scope of this thesis to go into details of this calculation, but it allows us to interpret the meaning of the Survey Propagation algorithm as follows.

With Survey Propagation, one wants to explore all the perfect colourings, i.e. the colourings that have exactly 0 energy. That means that, in the end, there can be no contradictions, i.e. no monochromatic edges. That is why, to avoid these contradictions, we require an extra state, the so-called joker state, denoted by a '*'. Consequently, the interpretation of the cavity magnetisations is slightly modified.

A.5 The formulae

Now, $\eta^{\mu}_{v\to e}$ is the probability that vertex v is forced to take the colour μ in the absence of neighbouring edge e, i.e. μ is the only colour left free for v, while $\eta^{*}_{v\to e}$ is the probability that vertex v is not forced to take any specific colour μ in the absence of neighbouring edge e, i.e. at least two colours are available. In this way, all local contradictions are avoided, by throwing out all the configurations where there is a clash.

Note that we deal here with the 3-colouring problem.

From [18], we get the vertices updates:

$$\eta^{\mu}_{v\to e} = \frac{\prod_{f\in n(v)\setminus e}(1-\eta^{\mu}_{f\to v}) - \sum_{\nu\neq\mu}\prod_{f\in n(v)\setminus e}(\eta^{*}_{f\to v}+\eta^{\mu}_{f\to v}) + \prod_{f\in n(v)\setminus e}\eta^{*}_{f\to v}}{\sum_{\mu}\prod_{f\in n(v)\setminus e}(1-\eta^{\mu}_{f\to v}) - \sum_{\mu}\prod_{f\in n(v)\setminus e}(\eta^{*}_{f\to v}+\eta^{\mu}_{f\to v}) + \prod_{f\in n(v)\setminus e}\eta^{*}_{f\to v}}, \quad \text{(A.1)}$$

$$\eta^{*}_{v\to e} = 1 - \sum_{\mu}\eta^{\mu}_{v\to e} \quad \text{(A.2)}$$

To compute the messages sent by an edge to a vertex, one has to notice that an edge e will forbid a colour μ to vertex v. From this principle, we found the new edges updates:

$$\eta^{\mu}_{e\to v} = \prod_{w\in n(e)\setminus v} \eta^{\mu}_{w\to e} \tag{A.3}$$

$$\eta^{*}_{e\to v} = 1 - \sum_{\mu} \eta^{\mu}_{e\to v} \tag{A.4}$$

One can now check that, if a vertex v_1 is forced to take a colour μ_1 and v_2 is forced to take the same colour, and v_1, v_2 and v_3 are connected by an hyperedge, then μ_1 will be forbidden to v_3. But if v_1 and v_2 have different colours, then v_3 will not be forced to take any colours and then will be in the joker state.

The magnetisations can be deducted from Equations (A.1) and (A.2) by simply ignoring the cavity rule.

A.6 The algorithm

The principle of Survey Propagation is slightly different from Belief Propagation'. Basically, here, vertices and edges are updated until convergence. Then the magnetisations are calculated and the fixing can take place. The algorithm stops when no vertex can be fixed. Finally, only a part of the graph will be coloured, the rest being in the joker state. One has then to apply another colouring algorithm such as Belief Propagation or Simulated Annealing to finish the colouring.

The update functions are similar to the ones for Belief Propagation.

Appendix B

Parallelization algorithm

Note that this algorithm requires the package MPI.

B.1 Master machine

```
a = array of the states of all the other processors
For k = 1 to (nb_processors-1) Do
 Reception_state_Non_Blocking(from processor k to send a signal in a[k])
EndFor

While (jobs to be done) Do
  For k = 1 to (nb_processors-1) Do
    If(k sent a signal in a[k]) Do
      jobs=jobs-1
      Send to processor k job to do
      Reception_state_Non_Blocking(from processor k to send a signal in a[k])
    EndIf
  EndFor
EndWhile

For k = 1 to (nb_processors-1) Do
  Send to processor k job finished
```

```
EndFor
```

B.2 Slave machine

```
finished = 0;
first_time = 1;
While (not finished) Do
  If (not first_time)
  Then
    Reception_state_Blocking(from master processor to send job to be done)
    Do job
  EndThen

  Send to master processor signal in a[k]
  first_time = 0
EndWhile

Send to master processor signal in a[k]
```

The signal sent by a slave processor to the master is an integer saying whether it is free or not.

At the beginning, the master processor is waiting for the first free processor to send a signal saying that it is free. Then a job is sent to this processor and, since this processor is now busy, the master processor is waiting for this processor to be ready again to send it a job, and so on. Meanwhile, the master processor will permanently check for a free machine (by looping over all the slave processors while jobs remain), assigning a job to the first free processor encountered. The master processor is always either sending a job or checking for a free processor.

Appendix C

Backtrack algorithm

C.1 Backtrack algorithm for perfect colourings

```
For each vertex Do
  Initialize_available_colours
EndFor

perfect = 0
i = 1

Step :
  Take the first available colour for i
  Goto Test

Test :
  If (colouring not perfect perfect so far) Goto Flip
  If (i == number_vertices) Goto Block
  i = i+1
  Goto to Step

Block :
  If (colouring is good)
```

```
  Then
    perfect = 1
    Goto End
  EndThen

Flip :
  If (i == 1) Goto End
  If (Exist available_colours for i)
  Then Do
    Take the next one
    Goto Test
  EndThen
  Else Do
    i = i-1
    Goto Flip
  EndElse

End :
  If (perfect == 1) Print Perfect colouring found
  Else Printf No perfect colouring found
```

C.2 Backtrack algorithm for optimal solutions

This algorithm takes `ref`, a number of unsatisfied edges, as argument. If it is run alone, one should take a high value for `ref`. But to save time, Belief Propagation is applied first so as to minimize this argument.

```
i = 1

Step :
  Give the first colour to i
  Goto Test
```

```
Test :
  If (number of unsatisfied edges with the current colouring >= ref) Goto Flip
  If (i == number_vertices) Goto Block
  i = i+1
  Goto to Step

Block :
  nb_unsat = final number of unsatisfied edges
  If (nb_unsat == 0) Goto End
  ref = nb_unsat
  Goto Flip

Flip :
  If (i == 1) Goto End
  If (Exist available_colours for i)
  Then Do
    Take the next one
    Goto Test
  EndThen
  Else Do
    i = i-1
    Goto Flip
  EndElse

End :
  If (nb_unsat == 0) Print Perfect colouring found
  Else Print Best number of unsatisfied edges : ref
```

C.3 Graphical diagram of the algorithm

The two algorithms described above can be visualized using the diagram in Figure C.1

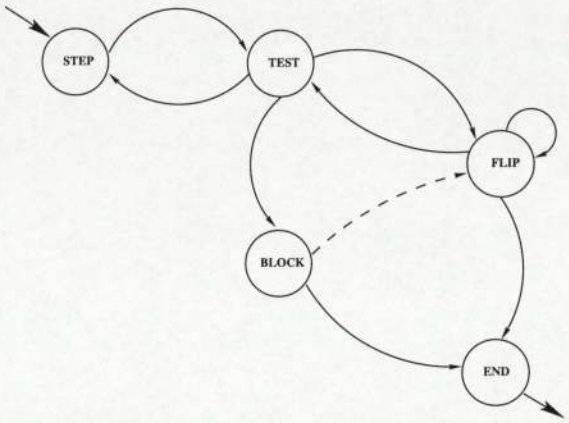


Figure C.1: The backtrack states diagram. Note that the dashed arrow is present only in the second version of this algorithm.

Appendix D

Statistical physics notations

First, the Hamiltonian or the cost function, representing the microscopic energy given a colouring $\vec{c}$ is defined by:

$$\mathcal{H}(\vec{c}) \equiv \sum_{e \in N_e} \mathcal{Z}_e(\vec{c}_e), \tag{D.1}$$

where $\mathcal{Z}_e$ is the function defined in Section 3.1 and $\vec{c}_e$ is the restriction of $\vec{c}$ to the variables which are the arguments of $\mathcal{Z}$. Then we define the partition sum, $\mathcal{Z}$, β being the inverse temperature:

$$\mathcal{Z} \equiv \sum_{\vec{c}} \exp(-\beta \mathcal{H}(\vec{c})). \tag{D.2}$$

We can now write the probability of a colouring $\vec{c}$ to be chosen, using Boltzmann's Law:

$$p(\vec{c}) = \frac{\exp(-\beta \mathcal{H}(\vec{c}))}{\mathcal{Z}}. \tag{D.3}$$

Now, let us define the average value of a physical quantity $\mathcal{Q}$, depending on the colouring, over the ensemble of all possible colourings:

$$\langle \mathcal{Q}(\vec{c}) \rangle_{\vec{c}} \equiv \sum_{\vec{c}} \mathcal{Q}(\vec{c}) p(\vec{c}). \tag{D.4}$$

Then the expression of the internal energy, E, comes immediately:

$$E \equiv \langle \mathcal{H}(\vec{c}) \rangle_{\vec{c}} = \sum_{\vec{c}} \frac{\mathcal{H}(\vec{c}) \exp(-\beta \mathcal{H}(\vec{c}))}{\mathcal{Z}}. \tag{D.5}$$

The definition of the free energy is:

$$\mathcal{F} \equiv -\frac{1}{\beta} \log(\mathcal{Z}). \tag{D.6}$$

Using the relations of thermodynamic: $\mathcal{F} = E - TS$, we get the internal energy and the entropy:

$$E = \frac{\partial \beta F}{\partial \beta}, \tag{D.7}$$

$$S = \beta(E - \mathcal{F}). \tag{D.8}$$

Appendix E

Energy calculations

Let us recall that :

$$\mathcal{F}_e = \frac{1}{\beta}\frac{1}{N_e}\left(N_v \sum_{\mu\in[\![1,p]\!]} f_\mu \hat{f}_\mu + G_1 - G_2 - G_3\right)$$

where :

$$\begin{aligned}
G_1 &= \sum_{v\in V}\sum_{e\in\varepsilon(v)} \log\left(1 - \sum_{\mu\in[\![1,p]\!]} \eta^\mu_{v\to e}\eta^\mu_{e\to v}\right)\\
G_2 &= \sum_{e\in E} \log\left(1 - \Delta \sum_{\mu\in[\![1,p]\!]} \prod_{v\in\nu(e)} \eta^\mu_{v\to e}\right)\\
G_3 &= \sum_{v\in V} \log\left(\sum_{\mu\in[\![1,p]\!]} e^{\hat{f}_\mu} \prod_{e\in\varepsilon(e)} (1-\eta^\mu_{e\to v})\right)
\end{aligned}$$

with E and V are the set of edges and vertices respectively, β the inverse temperature, $\Delta = 1-e^{-\beta}$, $\nu(e)$ the set of vertices of edge e and $\varepsilon(v)$ the set of edges of vertex v.

Note the presence of f_μ, the actual fraction of vertices of colour μ, and $\hat{f}_\mu$, the corresponding chemical potential, for the constrained fraction of vertices. Then the ground state energy per edge and the entropy per vertex are :

$$E_{0,e} = \lim_{\beta\to\infty} \frac{1}{N_e}\sum_{e\in E} \frac{e^{-\beta}\sum_{\mu=1}^{p}\prod_{v\in\nu(e)}\eta^\mu_{v\to e}}{1-\Delta\sum_{\mu=1}^{p}\prod_{v\in\nu(e)}\eta^\mu_{v\to e}} \tag{E.1}$$

$$S_{0,v} = \beta\frac{N_e}{N_v}(E_{0,e} - \mathcal{F}_e) \tag{E.2}$$

Note also that it is convenient to consider the energy per edge ($E_{0,e}$) and the entropy per vertex ($S_{0,v}$) so that $E_{0,e}$ is the fraction of unsatisfied edges (i.e. $0 \leq E_{0,e} \leq 1$) and $S_{0,v}$ is the entropy per degree of freedom (i.e. $0 \leq S_{0,v} \leq \log(p)$), the number of different colouring schemes that share the

minimum number of unsatisfied edges.

In the case where $K_{avg} = \sum_k kP(K=k)$, and knowing that in the paramagnetic state, we have : $\forall \mu \in [\![1,p]\!], \quad v \in V, \quad e \in E,$

$$\begin{aligned} \eta^{\mu}_{v\to e} &= \frac{1}{p} \\ \eta^{\mu}_{e\to v} &= \frac{\Delta}{p^{k(e)-1}} \end{aligned}$$

then

$$\begin{aligned} E_{o,e,pm} &= \sum_k P(K=k)\frac{e^{-\beta}}{p^{k-1}-\Delta} \\ G_1 &= N_v\lambda\sum_k P(K=k)\log((1-\frac{\Delta}{p^{k-1}})) \\ G_2 &= N_e\sum_k P(K=k)\log(1-\frac{\Delta}{p^{k-1}}) \\ G_3 &= \sum_{v\in V}\sum_k P(K=k)\log\left(p\prod_{e\in\varepsilon(v)}(1-\frac{\Delta}{p^{k-1}})\right) \\ &= N_v\log(p) + N_v\lambda\sum_k P(K=k)\log(1-\frac{\Delta}{p^{k-1}}) \\ \mathcal{F}_e &= \frac{1}{\beta}\left(\frac{\lambda k_{avg}-\lambda-k_{avg}}{\lambda}\log(p) - \sum_k P(K=k)\log(p^{k-1}-\Delta)\right) \end{aligned}$$

If we consider we have a mixture of $1-\alpha$ edges linked to K_{min} and α edges linked to K_{max}, then $K_{avg} = (1-\alpha) * K_{min} + \alpha * K_{max}$ and the quantities defined above become :

$$\begin{aligned} E_{o,e,pm} &= (1-\alpha)\frac{e^{-\beta}}{p^{K_{min}-1}-\Delta} + \alpha\frac{e^{-\beta}}{p^{K_{max}-1}-\Delta} \\ \mathcal{F}_e &= \frac{1}{\beta}\left(\frac{\lambda k_{avg}-\lambda-k_{avg}}{\lambda}\log(p) - (1-\alpha)\log(p^{K_{min}-1}-\Delta) - \alpha\log(p^{K_{max}-1}-\Delta)\right) \end{aligned}$$

Appendix F

Additional simulations

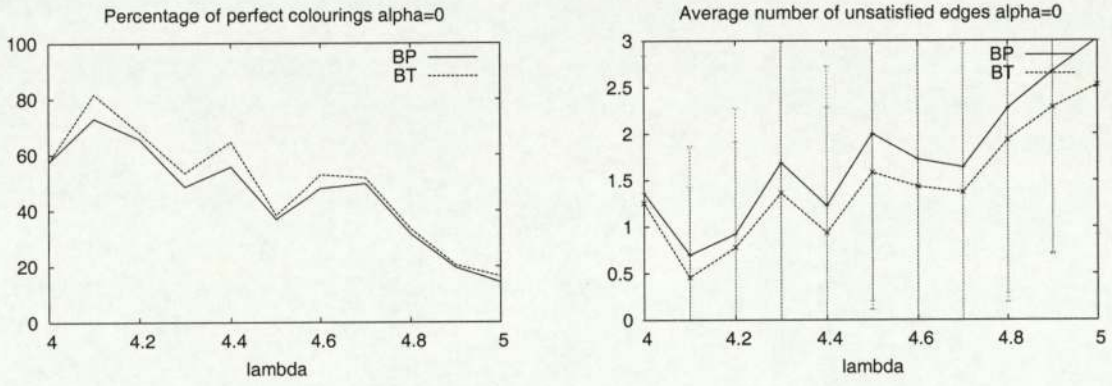


Figure F.1: Performance of Exact enumeration on random graphs compared with Belief Propagation. $N_v = 20$, $q = 3$ and $\alpha = 0$, data computed over 10^3 graphs for each connectivity.

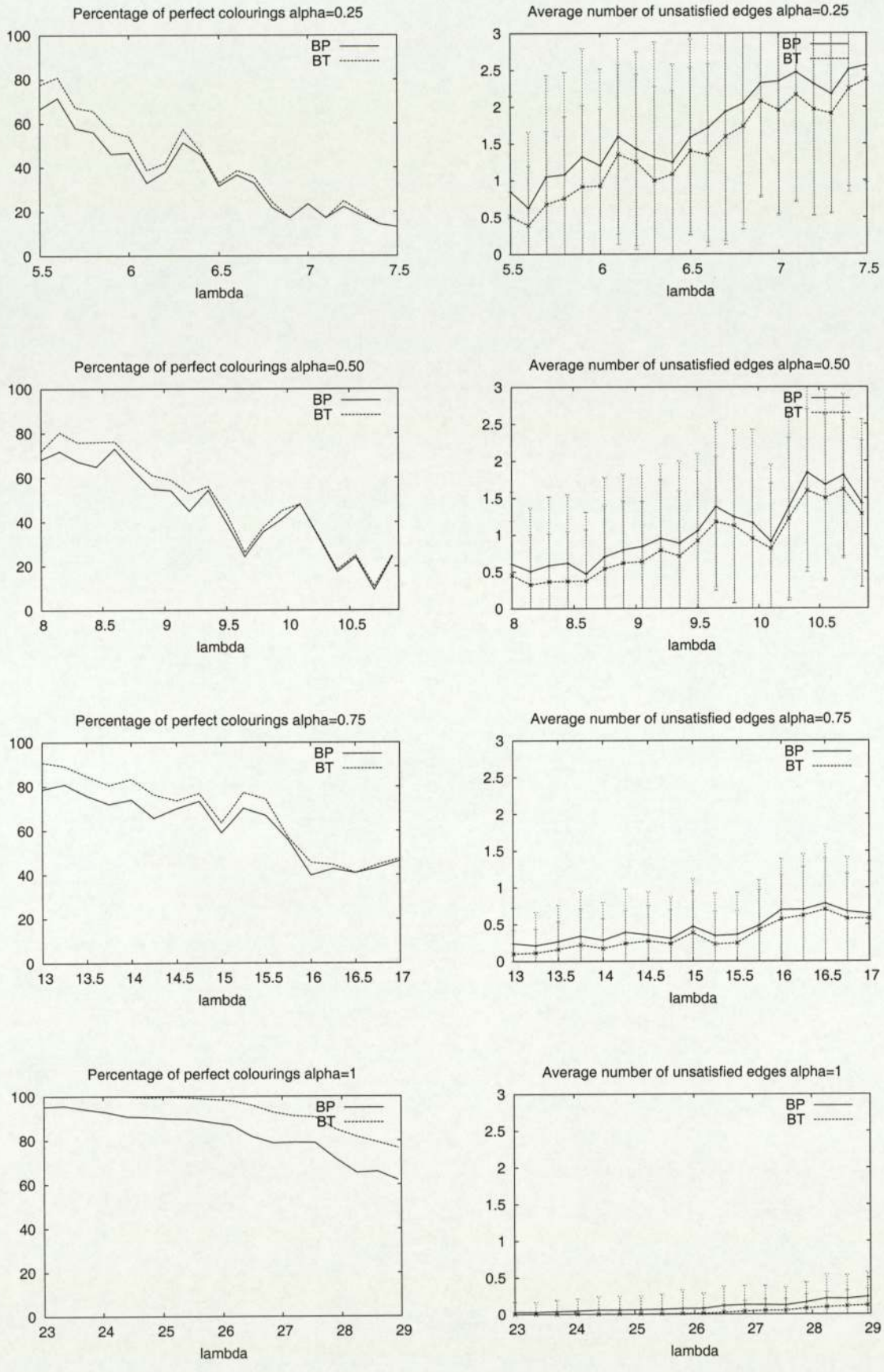


Figure F.2: Performance of exact enumeration on random graphs compared with Belief Propagation. $N_v = 20$, $q = 3$ and $\alpha \in \{0.25, 0.5, 0.75, 1\}$. Data computed over 10^3 graphs for each connectivity.

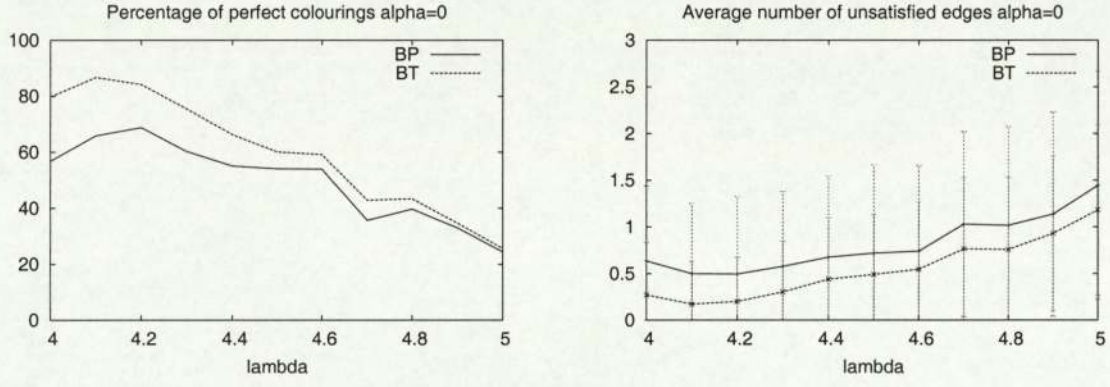
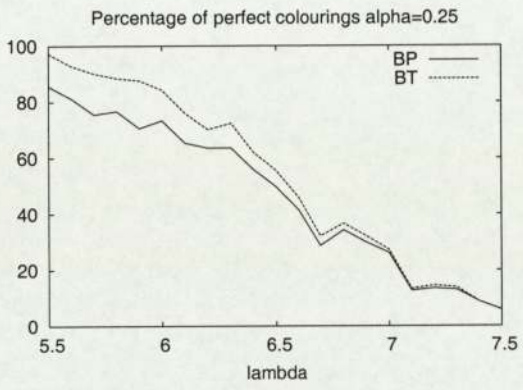
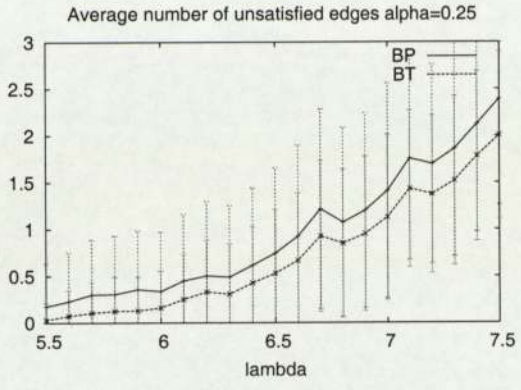
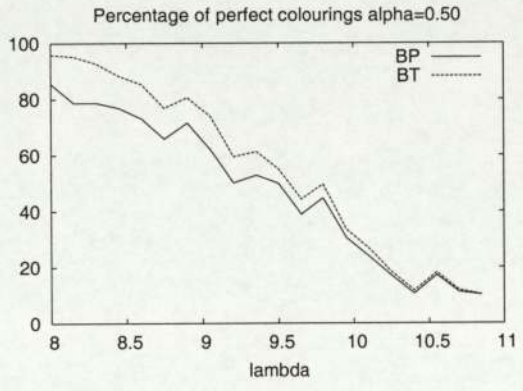
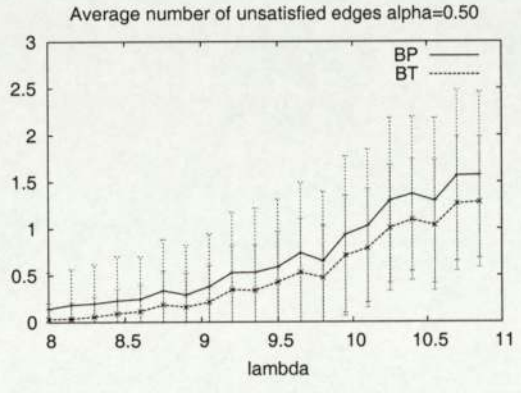


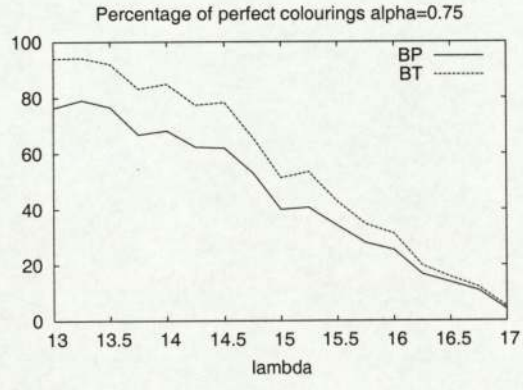
Figure F.3: Performance of Exact enumeration on random graphs compared with Belief Propagation. $N_v = 40$, $q = 3$ and $\alpha = 0$, data computed over 10^3 graphs for each connectivity.

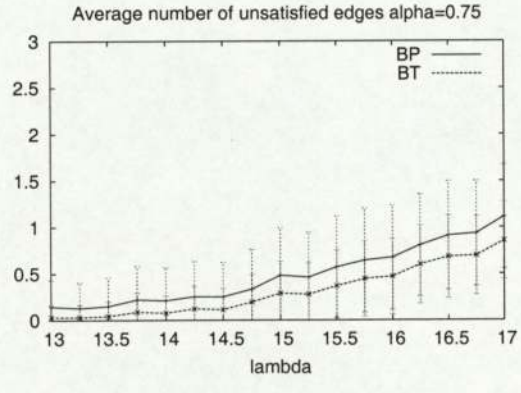


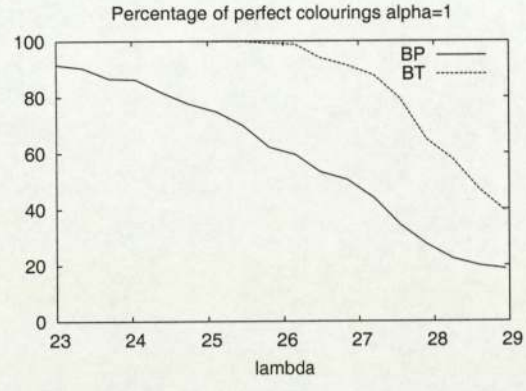












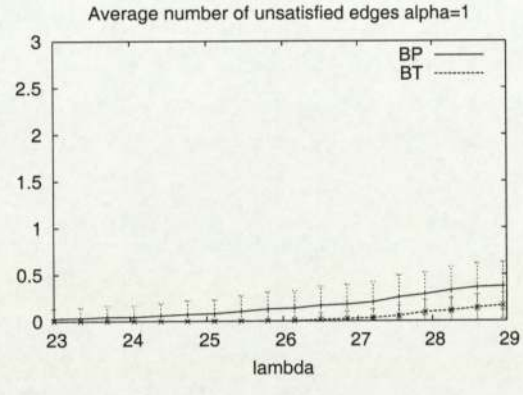


Figure F.4: Performance of exact enumeration on random graphs compared with Belief Propagation. $N_v = 40$, $q = 3$ and $\alpha \in \{0.25, 0.5, 0.75, 1\}$. Data computed over 10^3 graphs for each connectivity.

ASTON UNIVERSITY
LIBRARY & INFORMATION SERVICES

Bibliography

[1] D. Achlioptas and C. Moore. Almost all graphs with average degree 4 are 3-colorable. *Journal of Computer and System Sciences*, pages 441–471, 2003.

[2] K. Appel and W. Haken. Every planar map is four colorable. 1: Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977.

[3] K. Appel and W. Haken. Every planar map is four colorable. 2: Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977.

[4] A. Braunstein, M. Mézard, A. Pagnani, M. Weigt, and R. Zecchina. Polynomial iterative algorithms for colouring and analyzing random graphs. *Physical Review E*, 68:036702, 2003.

[5] A. Braunstein, M. Mézard, M. Weigt, and R. Zecchina. Constraint satisfaction by survey propagation. *Sante Fe series on Complexity*, 2002.

[6] A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation : an algorithm for satisfiability. *Random Structures and Algorithms*, november 2003.

[7] D. Brelaz. New methods to color the vertices of a graph. *Communications of the Association for Computer Machinery*, 22(4):251–256, 1979.

[8] P. Briggs, K. Cooper, K. Kennedy, and L. Torczon. Coloring heuristics for register allocation. *In ASCM Conference on Program Language Design and Implementation*, pages 275–284, 1989.

[9] R. J. Brown. Chromatic scheduling and the chromatic number problem. *Management Science*, 19:456–463, 1972.

[10] A. Capitanio, N. Dutt, and A. Nicolau. Partitioning of variables for multiple-register-file architectures via hypergraph coloring. In *Parallel Architectures and Compilation Techniques*, pages 319–322. IFIP, 1994.

[11] A. Capitanio, N. Dutt, and A. Nicolau. Partitioning of variables for multiple-register-file vliw architectures. In *International Conference on Parallel Processing*, pages 298–301, 1994.

[12] T. Castellani, V. Napolano, F. Ricci-Tersenghi, and R. Zecchina. Bicoloring random hypergraphs. *Journal of Physics A*, 36:11037, 2003.

[13] G. Chaitin. Register allocation and spilling via graph coloring. In *Proceedings of the ACM SIGPLAN 82 Symposium on Compiler Construction*, pages 98–105, 1982.

[14] M. Chams, A. Hertz, and D. De Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operations Research*, 32:260–266, 1987.

[15] N. Christofides. *Graph Theory : An Algorithmic Approach.* Academic Press, 1975.

[16] J. Culberson and I. P. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265:227–264, 2001.

[17] P. Erdös and A. Rényi. On random graphs 1. *Publitiones Mathematicae Univeritatis Debreaniensis*, 6:290–297, 1959.

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

[19] A. Ferreira, S. Perennes, H. Rivano, A.W. Richa, and N. Stier-Moses. Models, complexity and algorithms for the design of multi-fiber wdm networks. *Telecommunication Systems*, 24:123, 2003.

[20] R. G. Gallager. Low density parity check codes. *Research Monograph Series*, 21, 1963.

[21] A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology*, 35(1):8–14, 1986.

[22] M. R. Garey and D. S. Johnson. The complexity of near-optimal graph colouring. *Journal of the Association for Computing Machinery*, 23:43–49, 1976.

[23] A. H. Gebremehdim. Hypergraph coloring : Algorithms and applications. march 2003.

[24] A. Hertz and D. De Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1988.

[25] http://www.mhpcc.edu/workshop/mpi. Parallel programming workshop.

[26] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; pat ii, graph coloring and number partitioning. *Operations Research*, 39:378–406, 1991.

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

[28] A. C. Kaporis, L. M. Kirousis, and Y. C. Stamatiou. A note on the non-colorability threshold of a random graph. *Electronic Journal of Combinatorics*, 7, 2000.

[29] S. Khot. Hardness results for coloring 3-colorable 3-uniform random hypergraphs. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 23–32, November 2002.

[30] F. Kschischang, B. Frey, and H.A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Infortmation Theory*, 47(2):498–519, February 2001.

[31] M. Kubale and B. Jackowski. A generalized implicit enumeration algorithm for graph colouring. *Communications of the Association for Computation Machinery*, 28:412–418, 1985.

[32] F. T. Leighton. A graph colouring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84:489–503, 1979.

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

[34] D. Matula, G. Marble, and J. Isaacson. Graph coloring algorithms. *Graph Theory and Computing*, pages 109–122, 1972.

[35] C. Morgenstern and H. Shapiro. Coloration neighborhood structures for general graph colouring. *First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 226–235, 1990.

[36] R. Mulet, A. Pagnani, M. Weigt, and R. Zecchina. Coloring random graphs. *Physical Review Letters*, 89, 2002. 268701.

[37] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann Publishers, second edition, 1988.

[38] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C : The Art of Scientific Computing.* Cambridge University Press, second edition, 1992.

[39] S. Pumphrey. Solving the satisfiability problem using message-passing techniques. *Physics Project Report Part III*, 2001.

[40] J. van Mourik and D. Saad. Random graph colouring - a statistical physics approach. *Physical Review E*, 66:056120, 2002.

[41] F. Wu. The potts model. *Rev. Mod. Physics*, 54:235–268, 1982.

Index

q-COL/UNCOL, 13

q-colouring, 9

backtrack algorithm, 16, 41, 48, 57

Bayesian network, 15, 24

belief propagation, 15, 16, 26, 29, 34, 44, 47, 49

cavity rule, 26

chemical potential, 28, 48

chromatic number, 9, 10, 15

connectivity, 13, 14

- critical, 13, 16, 29, 35, 36, 41
- dynamical, 13, 20, 29, 31

constraint, 8, 10, 11, 15, 26

deterministic, 13

distributed storage, 10

early stopping, 42

energy, 27, 28, 34, 62

- free energy, 27, 62

entropy, 27, 28, 62

error correcting codes, 15

exact enumeration, 14–16

fixing, 20, 50

- threshold, 21

frequency assignment, 10

graph

- bi-partite, 15
- definition, 8
- hypergraph, 8, 10–12, 18
- random colourable graph, 39
- random colourable graph generation, 18, 47
- random graph, 36
- random graph generation, 17, 47

graph generation, *see* graph

heuristic methods, 14, 47

hyperedge, 8, 10, 11

hypergraph, *see* graph

implicit enumeration, *see* exact enumeration

magnetisation, 15, 20, 21, 35, 50

message-passing techniques, 15, 36

mixture coefficient, 8, 12, 63

NP-hard, 13, 36

optimal colouring, 9

parallelization, 16, 21, 47, 55

paramagnetic state, 20, 29, 31, 63

partitioning, 11, 12

phase diagram, 13

pruning, 19, 47

register allocation, *see* partitioning

scheduling, 10

simulated annealing, 14

stationary distribution, 31

statistical physics, 27

successive augmentation, 14

sum-product algorithm, 15, 24

survey propagation, 15, 16, 48

tabu search, 14

time averaging, 34

update rules, 25, 26, 52