Joana
Dirce
Santos
Martins

Algoritmo de Monte Carlo Wang-Landau e Belief Propagation para bipartição de grafos

Monte Carlo Wang-Landau algorithm and Belief Propagation for graph bipartitioning

Universidade de Aveiro
2022

Joana
Dirce
Santos
Martins

Algoritmo de Monte Carlo Wang-Landau e Belief Propagation para bipartição de grafos

Monte Carlo Wang-Landau algorithm and Belief Propagation for graph bipartitioning

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Computacional, realizada sob a orientação científica do Doutor António Luís Ferreira, Professor associado do Departamento de Física da Universidade de Aveiro.

Dedico este trabalho à minha família pelo incansável apoio.

o júri / the jury

presidente / president

Prof. Doutora Margarida Maria Resende Vieira Facão
professora auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor João Manuel Viana Parente Lopes
professor auxiliar convidado da Faculdade de Ciências da Universidade do Porto

Prof. Doutor António Luis Campos de Sousa Ferreira
professor associado da Universidade de Aveiro

agradecimentos / acknowledgements

Agradeço todo o apoio do meu orientador e de outros docentes da UA.

Palavras Chave grafos, partição, método de Wang-Landau, algoritmo de belief propagation.

Resumo

A divisão de grafos, como descrições abstratas de componentes que interagem num sistema, é uma das operações fundamentais em grafos, e pode ser feita usando uma grande variedade de algoritmos. Este trabalho está focado na divisão de um grafo em duas partes usando métodos de inferência. Um deles é o algoritmo de Belief Propagation, especialmente conhecido pelo seu curto tempo de execução, principalmente quando comparado com métodos de Monte Carlo. No entanto, apesar do seu sucesso na estimativa do limite inferior do custo de partição na maioria das situações, quando se trata de determinar os nós do grafo que pertencem a cada grupo de modo a que tal custo ótimo seja obtido, a abordagem rápida da Belief Propagation tende a falhar. Neste contexto, foi desenvolvido um algoritmo de partição baseado no algoritmo de Wang-Landau (um método de Monte Carlo), que apesar de ter um tempo de execução mais longo é frequentemente mais bem sucedido em solucionar problemas concretos.

Keywords graphs, factor graphs, partitioning, Wang-Landau method, belief propagation algorithm.

Abstract The division of graphs, as abstract descriptions of interacting components of a system, is one of the fundamental operations made on graphs, and it can be done using a great variety of algorithms. This work is focused on the division of a graph into two parts using inference methods. One of them is the Belief Propagation algorithm, known for its small runtime, specially when compared with Monte Carlo based methods. However, despite its success on the estimation of the partitioning cost lower bound in most of the situations, when it comes to determine the graph nodes that belong to each group so that such an optimal cost is obtained, the fast Belief Propagation approach tends to fail. In this context, it was developed a partitioning algorithm based on the Wang-Landau algorithm (a Monte Carlo method), that despite having a longer runtime is often more successful as a problem solver.

Contents

List of Figures

List of Tables

Glossary

BP	belief propagation
WL	Wang-Landau
EO	extremal optimization
1-RSB	1-replica symmetry breaking
b_E^{BP}	BP energy estimated cost
b^{BP}	BP actual cost
$b^{BP_{dec}}$	BP with decimation actual cost
b^{WL}	Wang-Landau (WL) actual cost
b^{EO}	EO actual cost
b_E^{1-RSB}	1-RSB energy estimated cost
m^d	desired magnetization value

CHAPTER 1

Introduction

Many systems of interest, from social sciences to computer vision or statistical physics, may be decomposed in individual parts linked together by dependencies or constraints. Examples of this include human societies, as collections of people linked by social interactions, or even the Internet, as a collection of computers linked by data connections [1].

Such systems can be represented by graphs, which are collections of objects, called vertices/nodes, joined together by lines, the edges. Even though, a lot of information is usually lost in the process of reducing a full system to an abstract graph representation, where only the basics of connection patterns and little else is captured. Still, graphs are powerful means of representing patterns of connections or interactions between the parts of a system [2].

Dividing a graph into smaller pieces is one of the fundamental algorithmic operations made on graphs [3]. There are a number of reasons why one might want to divide a network into clusters, but all come down to two different classes of problems: either *graph partitioning* problems or *community detection* problems, both answering different questions [1].

In graph partitioning, the number and sizes of the non-overlapping groups are fixed according with a certain objective function. Such problems arise in a variety of circumstances, particularly in computer science, mathematics, physics, and in the study of networks themselves. A typical example is the speed up of computational problems by assigning to each processor a certain task concerning a subset of variables, such that the number of communications, that is, the number of edges between groups, is minimized. Most often we want to assign an equal or roughly equal number of variables to each processor so the workload will be balanced among them [1]. Another example is the design of an electric circuit, where one needs to know on which board to place the different components to minimize the number of links between different boards [4].

Community detection differs from graph partitioning in that the number and sizes of the groups are not specified, being the goal to find the natural divisions of the network. This way, community detection is used as a tool for the analysis and understanding of network data. Some examples are the identification of research groups in a university department, where the

edges represent links between scientists who have coauthored scientific papers; the evaluation of the nature of social interactions in a school, where the edges represent friendships between students of different ethnicity's; the division of a web graph according to related web pages or the division of a metabolic network in order to indicate its functional units [1].

In this work, we will focus on the graph partitioning problem, which is more deterministic, and in particular on bipartitioning, as the division of a graph into two parts.

CHAPTER 2

Graph representations

The interactions between vertices in a graph may happen following different rules, therefore, to better describe this interactions must be considered different types of edges, either directed edges or undirected edges [5].

In a directed graphical model, the probability of occurrence of the state x_i of a variable i is dependent on direct causal dependencies between variable i and other system variables, its parents, $Par(i)$. Such dependencies are expressed by directed edges, each one beginning at a parent node $Par(i)$ and pointing to a node i.

The Bayesian networks, coined by Judea Pearl in 1985 [6], are directed acyclic graphs that offer consistent semantics for representing causes and effects. An example of its application in statistical inference would be the prediction of the disease that provoke certain symptoms, given the probabilistic relationships between diseases and symptoms.

Meanwhile, undirected graphical models, also called *Markov random fields* or *Markov networks*, can represent any probability distribution over N random variables $\underline{x} = \{x_1, \dots, x_N\}$, just like directed graphs. However, a directed graph more naturally represents conditional probabilities directly, whereas an undirected graph more naturally represents conditional independence properties [5].

Although a graph may present only one type of edges, such as a directed graph or an undirected graph, a greater detail can be obtained considering the possible coexistence of both types of edges on the same graph, as a mixed graph. Such type of graph has a wide range of applications, from the modelling problems of engineering to physical science, biological science, communication technology, computer technology and others [7][8].

All this graphical models can be represented by a different kind of graph representation – the factor graph. This kind of graph makes use of the concept of "function nodes", which captures the dependencies between each set of interacting variables by connecting with edges $(ia) \in L$ the variable nodes belonging to the set I to function nodes belonging to the set A, which represents the function $f_a : X^{k_a} \to \mathbb{R}$. This way, each variable node $i \in I$ has a set of neighbouring function nodes $(\partial i \equiv a_1^i, \dots, a_{k_i}^i)$ and each function node has a set of

neighbouring I nodes ($\partial a \equiv i_1^a \, , \ldots \, , i_{k_a}^a$) [9].

Therefore, a factor graph (as the one represented in figure 2.1), besides containing N variable nodes (represented by circles), it also contains F function nodes (represented by squares). Some variables might have a known preferred value, which is expressed by the function $f_i(x_i)$, called 'evidence' [10].

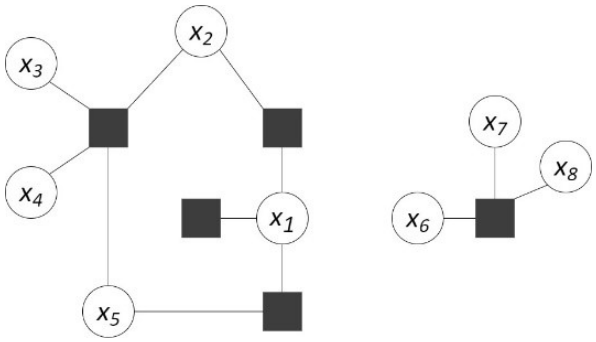

Figure 2.1: Abstract example of a factor graph. The degree of a function node (the number of edges incident to it) is always superior to zero. The variable nodes are represented by circles and the function nodes represented by squares [9].

There are many systems that can be described by a factor graph representation, the only requirement is for them to be translated by a global function that factors into a product of simpler "local" functions, each of them dependent on a subset of variables [9]. The factor graph is useful in the way that easily allows the application of the *belief propagation* (BP) algorithm (also known as the *sum-product* or *replica symmetric* algorithm), which is a computational inference algorithm that computes marginal functions associated with the systems global function.

In the following sections there is a description of different kinds of systems which can be modeled by a graphical arrangement of variables, particularly by a factor graph. Graphical modulation is fundamental so that partitioning methods can be applied to systems.

2.1 Expression Trees

Expression trees are often used in computer science to represent arithmetic expressions as ordered rooted trees in which internal vertices, that is, vertices with descendants, represent arithmetic operators, such as addition, multiplication, negation, etc., and leaf vertices, that is, vertices without descendants, representing variables or constants.

For example, the tree of figure 2.2 represents the expression $x(y + z)$ [11].

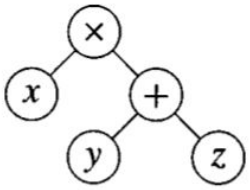

Figure 2.2: An expression tree representing $x(y + z)$, read from bottom to top [11].

Expression trees can be extended so that the leaf vertices represent functions, such as sums and products, and not just variables or constants. These expression trees combine their operands in the usual pointwise manner, in which functions are added and multiplied [11]. An example of this is presented in figure 2.3(a), that represents the marginal function $g_1(x_1)$, written as

$$\begin{aligned} g_1(x_1) &= f_A(x_1)\left[\sum_{x_2} f_B(x_2)\left(\sum_{x_3} f_C(x_1,x_2,x_3)\left(\sum_{x_4} f_D(x_3,x_4)\right)\left(\sum_{x_5} f_E(x_3,x_5)\right)\right)\right] \\ &= f_A(x_1)\sum_{\sim\{x_1\}}\left[f_B(x_2)f_C(x_1,x_2,x_3)\left(\sum_{\sim\{x_3\}} f_D(x_3,x_4)\right)\left(\sum_{\sim\{x_3\}} f_E(x_3,x_5)\right)\right], \end{aligned} \tag{2.1}$$

where the notation $\sim\{x_i\}$ indicates the variable x_i of the domain that is not summed over. The dashed lines in figure 2.3 represent trivial operations, that is, those with one or no operand, and that can be omitted as is the case of the summary operators $\sum_{\sim\{x\}}$ applied to a function with a single argument $\times$ [11].

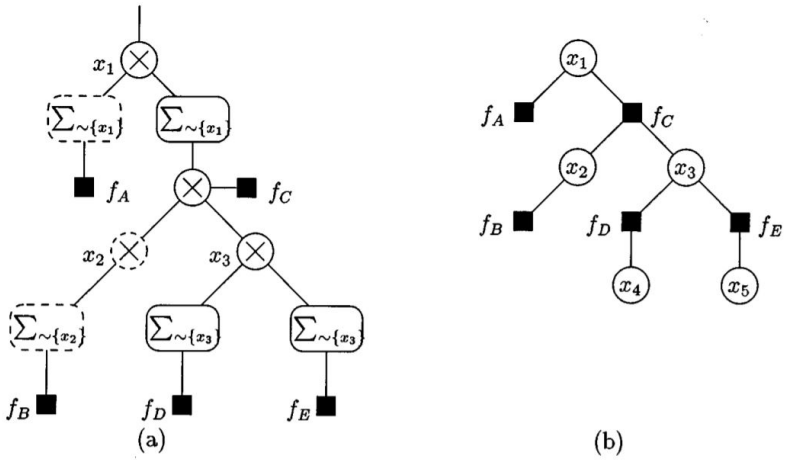

Figure 2.3: (a) A tree representation for the right-hand side of equation 2.1. (b) The factor graph as a rooted tree with x_1 as root [11].

In a similar way, being the same system marginal function $g_3(x_3)$ written as

$$g_3(x_3) = \left(\sum_{\sim\{x_3\}} f_A(x_1)f_B(x_2)f_C(x_1,x_2,x_3)\right)\left(\sum_{\sim\{x_3\}} f_D(x_3,x_4)\right)\left(\sum_{\sim\{x_3\}} f_E(x_3,x_5)\right), \tag{2.2}$$

the corresponding expression tree is obtained as in figure 2.4(a).

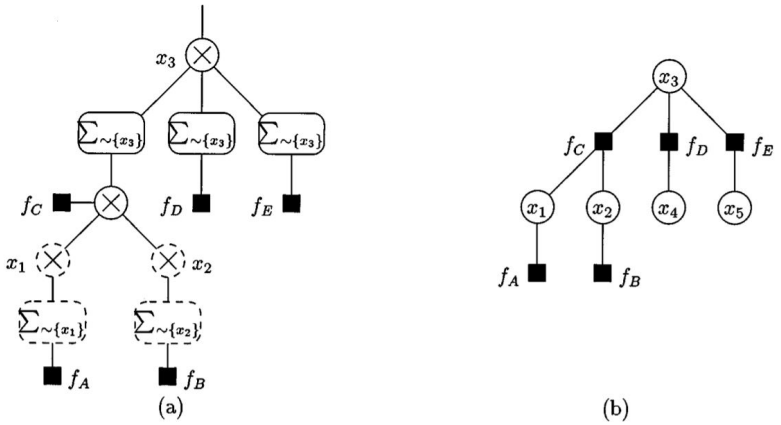

Figure 2.4: (a) A tree representation for the right-hand side of equation 2.2. (b) The factor graph as a rooted tree with x_3 as root [11].

Comparing the factor graphs of figures 2.3(b) and 2.4(b) with the corresponding trees, it is easy to note their correspondence. Moreover, it's observed that although the expression trees of different marginal functions are different, they both translate into the same factor graph. This is possible because the system global function $g(x_1, x_2, x_3, x_4, x_5)$, defined as

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5), \tag{2.3}$$

is translated into a tree like factor graph with no loops [11].

2.2 Behavioral modeling

Behavioral modeling is natural for codes and study's the system behavior by specifying which particular configurations of variables, called *codewords*, are valid [11].

The characteristic (or set membership indicator) function for a behavior B is defined as

$$\chi_B(x_1, ..., x_N) := [(x_1, ..., x_N) \in B], \tag{2.4}$$

In many important cases, membership of a particular configuration in a behavior B can be determined by applying a series of tests (checks), each involving some subset of the variables. A configuration is deemed valid if and only if it passes all tests [11].

Therefore, the predicate $[(x_1, ..., x_N) \in B]$ may be written as a logical conjunction of a series of "simpler" predicates. According to the Iverson's convention, a predicate $[P]$, as a $\{0, 1\}$-valued function that indicates the truth of P, can be factored according to

$$[P_1 \wedge P_2 \wedge ... \wedge P_n] = [P_1][P_2]...[P_n], \tag{2.5}$$

where $\wedge$ denotes the sum in GF(2) [11].

Thus, if P can be written as a logical conjunction of predicates, then $[P]$ can be represented by a factor graph [11].

There are various examples of behavioral models, however in this work we will just present the case of the Tanner graph, as we will restrict ourselves to factor graphs with just two kinds of nodes: factor nodes and variable nodes.

2.2.1 Tanner graph

Tanner graphs for linear codes are defined by an $|A| \times N$ parity-check matrix M, which consists of $|A|$ equations expressed in matrix form over N variables as $M\mathbf{x}^T = 0$. The corresponding characteristic function is given by the logical conjunction of these equations [11].

An example of a Tanner graph is shown in figure 2.5 for the parity-check matrix

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \tag{2.6}$$

also expressed by the characteristic function

$$\begin{aligned} \chi_B(x_1, x_2, ..., x_6) &= [(x_1, x_2, ..., x_6) \in B] \\ &= [x_1 \oplus x_2 \oplus x_5 = 0][x_2 \oplus x_3 \oplus x_6 = 0][x_1 \oplus x_3 \oplus x_4 = 0]. \end{aligned} \tag{2.7}$$

where the symbol $\oplus$ is used to represent the parity checks [11].

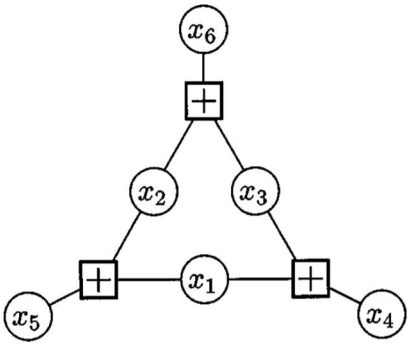

Figure 2.5: A Tanner graph for a binary linear code [11].

From figure 2.5 is easy to see that Tanner graphs work like factor graphs, where the squared parity check nodes correspond to function nodes that represent the predicates.

2.3 Probabilistic modeling

A joint probability distribution $p(\underline{x})$ may also be expressed in terms of a product of functions (the potentials $f_a(\underline{x}_{\partial a})$ and $f_i(x_i)$) divided by a normalization constant Z, the partition function of equation 2.9, as presented in the equation 2.8 [5][12].

$$p(\underline{x}) = \frac{1}{Z} \prod_{a \in A} f_a(\underline{x}_{\partial a}) \prod_{i \in I} f_i(x_i) \tag{2.8}$$

$$Z = \sum_{\underline{x}} p(\underline{x}) \tag{2.9}$$

Factor graphs for probability distributions arise in many situations. In the following sections are described some of them [11].

2.3.1 Coding theory

Coding theory deals with the design of error-correcting codes for the reliable transmission of information across noisy channels [13]. There are many applications using error correcting codes. Some of them are on the transmission of images from distant space, on the quality of sound in CD's, on telephone lines and on computer networks [14].

Consider the standard coding model in which a codeword $\underline{x} = (x_1, ..., x_N)$ is selected from a code C of length n and transmitted over a memoryless channel with the corresponding output sequence $\underline{y} = (y_1, ..., y_N)$ [11].

Assuming that the *a priori* distribution $p(\underline{x})$ is uniform over codewords, we have $p(\underline{x}) = \chi_C(\underline{x})/|C|$, where $\chi_C(\underline{x})$ is the characteristic function for C and $|C|$ is the number of codewords in C [11]. If the channel is memoryless, then $f(\underline{y}|\underline{x})$ factors as

$$f(\underline{y}|\underline{x}) = \prod_{i=1}^{N} f(y_i|x_i) \tag{2.10}$$

and we have the global function

$$g(\underline{x}, \underline{y}) = \frac{1}{|C|} \chi_C(\underline{x}) \prod_{i=1}^{N} f(y_i|x_i) \tag{2.11}$$

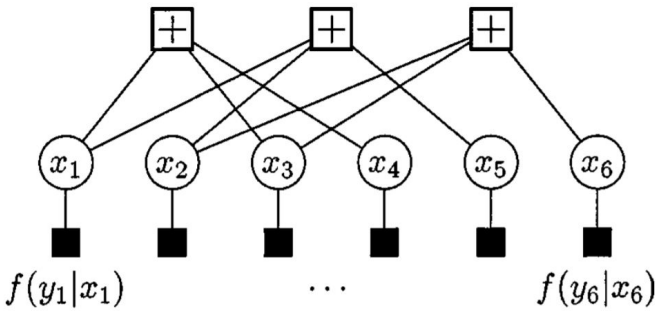

Figure 2.6: Factor graph for a joint APP distribution of codeword symbols [11].

An example of this is shown in figure 2.6, for a characteristic function

$$\begin{aligned} \chi_C(x_1, x_2, ..., x_6) &= [(x_1, x_2, ..., x_6) \in C] \\ &= [x_1 \oplus x_2 \oplus x_5 = 0][x_2 \oplus x_3 \oplus x_6 = 0][x_1 \oplus x_3 \oplus x_4 = 0]. \end{aligned} \tag{2.12}$$

It is easy to see that the example of figure 2.6 corresponds to a combination of behavioral and probabilistic modeling, as we have modeled both the valid behavior, that is, the set of codewords, and the *a posteriori* joint probability mass function over the variables, given the received output of a channel [11].

2.3.2 Markov Random Fields

A Markov random field (MRF) is a graphical model based on an undirected graph in which each node i corresponds to a random variable x_i. The graph is a Markov random field if the distribution $p(x_1, ..., x_N)$ satisfies the property of every variable being independent of non-neighboring variables in the graph, given the values of its immediate neighbors [11].

Various types of Markov models are used in signal processing and communications. In statistical physics, if the joint probability density is positive, the joint probability mass function $p(\underline{x})$ is expressed by the product of Gibbs potential functions as

$$p(\underline{x}) = \frac{1}{Z} \prod_{q \in Q} f_q(\underline{x}_q), \tag{2.13}$$

being Z a normalizing constant and $\underline{x}_q$ the set of variables that belong to a clique $q \in Q$, that is, that belong to a group of nodes that are all pairwise neighbors [11].

Markov random field were introduced as the general setting for the Ising model [15], a highly formalized stochastic model of a ferromagnet. This model consists of a graphical arrangement of discrete variables, each of them representing a magnetic dipole moment of an atomic spin [16][17].

On the Ising model, the spin σ_i of each node i can be in one of two states ($+1$ or -1) and interact with its neighbouring spins (that belong to the set ∂i along the edges (i, j)), as it is assumed that the molecules only exert short-range forces on each other [16]. Therefore, the Ising model may be viewed as a Markov Random Field with alphabet size 2 [18].

At equilibrium, the system spins $\underline{\sigma} = (\sigma_1, ..., \sigma_N)$ take a Boltzmann's probability distribution given by

$$p(\underline{\sigma}) = \frac{1}{Z} e^{-\beta E(\underline{\sigma})}, \tag{2.14}$$

for a partition function $Z = \sum_{\underline{\sigma}} e^{-\beta E(\underline{\sigma})}$, where β is the reciprocal of the thermodynamic temperature of the system. The system energy E (the Hamiltonian) is given by

$$E(\underline{\sigma}) = -\sum_{(i,j)} \sigma_i \sigma_j - H \sum_{i=1}^{N} \sigma_i \tag{2.15}$$

where the second sum represents the interaction of each spin with the external magnetic field H and the first sum the ferromagnetic interaction between each pair of neighbouring spins (i, j) [19] [9].

The system magnetization m is defined as

$$m = \frac{1}{N} \sum_{i=1}^{N} \sigma_i \tag{2.16}$$

and the joint probability mass function can be factored into

$$p(\underline{\sigma}) = \frac{1}{Z} \prod_{(i,j)} e^{\beta \sigma_i (\sigma_j + H)} \tag{2.17}$$

$$= \frac{1}{Z} \prod_{i=1}^{N} e^{\beta H \sigma_i} \prod_{(i,j)} e^{\beta \sigma_i \sigma_j} \tag{2.18}$$

2.3.3 Bayesian Networks

Bayesian networks are graphical models for a collection of random variables that are based on directed acyclic graphs [11]. Representing $f(x_i, \underline{x}_{Par(i)})$ the conditional probability distribution of the state x_i, given its parents, the joint probability distribution of a Bayesian network is given by

$$p\left(\underline{x}\right) = \prod_{i=1}^{N} p\left(x_i \middle| \underline{x}_{Par(i)}\right) \tag{2.19}$$

However, if the node i has no parents, the conditional probability $p\left(x_i \middle| \underline{x}_{Par(i)}\right)$ reduces to the variable marginal probability

$$p(x_i) = \sum_{\underline{x}_{Par(i)}} p(x_i, x_{\underline{x}_{Par(i)}}), \tag{2.20}$$

defined in terms of sums over all the possible states of node i ancestors, $Par(i)$ [10].

In figure 2.7 is shown an example of comparison between a Markov random field and a Bayesian network representation, with the corresponding factor graph representation. From there we conclude that the existence of directed or undirected edges does not have an impact on the overall structure of the factor graph, just on the particular definition of its function nodes.

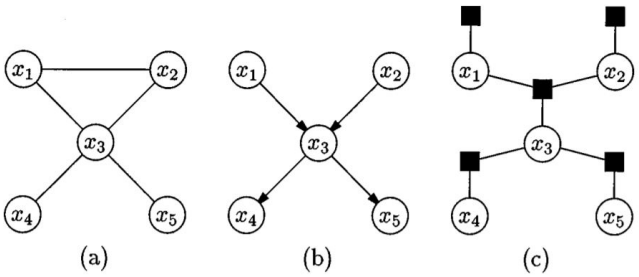

Figure 2.7: Graphical representations of the factorization $p(v_1, v_2, v_3, v_4, v_5) = Z^{-1} f_C(v_1, v_2, v_3) f_D(v_3, v_4) f_E(v_3, v_5)$, view as: (a) a Markov random field, (b) a Bayesian network or (c) a factor graph [11].

CHAPTER 3

Inference in graphical models

Sampling a distribution of strongly correlated variables is a central task in many scientific fields, such as in statistical mechanics, machine learning and statistical analysis [20]. The large number of configurations that exists in graphs with a high number of nodes makes it difficult to analytically evaluate the single node marginal probability distributions, since that requires summing over the entire state space [17]. For this reason, numerical algorithms of inference in graphical models are used.

The most common methods of doing inference are Monte Carlo based methods and variational methods, such as belief propagation (BP).

3.1 Monte Carlo methods

The Monte Carlo (MC) method is a probabilistic method that plays a fundamental role in modern physics and in inference and learning problems of fields like computational biology, machine learning, and simulated annealing [20]. The basic idea behind its application on Ising model systems is that, in order to calculate some thermodynamic quantity, it is enough to randomly sample in a phase space instead of averaging over all states. This means that it's not needed to evaluate the partition function because the computed quantity will eventually converge toward the real thermodynamic quantity, if the sampling is large enough [21] [17].

The Monte Carlo algorithm starts with the generation of a random set of spin configurations (one random value for each spin of the graph). After that it is randomly chosen a new configuration in the phase space, that in the case of the Ising model corresponds to propose a new value for a random spin i and accepting or rejecting the proposal with a certain probability.

After generating iteratively a sufficiently large number of new random configurations, it is generated a new configuration, approximately independent of the original one, and whose distribution moves towards the target distribution (the Boltzmann distribution of the Ising model), which allows to infer about its properties [17].

Advantages of the Monte Carlo method include its simplicity of implementation and theoretical guarantee of convergence [22]. However, it needs long runtimes in order to obtain high-precision estimates, as there are statistical errors and correlations and local energy barriers may prevent the efficient sampling of the relevant sets of configurations [20].

3.1.1 Wang-Landau method

The Wang-Landau (WL) method is an efficient Monte Carlo algorithm that, unlike conventional Monte Carlo methods that directly generate a canonical distribution at a given temperature, it estimates the density of states $W(E, m)$ and entropy $S(E, m) = \ln(W(E, m))$ via a random walk through the space of configurations [23].

The method bases itself on the fact that the histogram $h(E, m)$ of frequencies of energy and magnetization is flat, as the stationary probability of the generated spin configurations is $p \propto \exp(-S(E, m))$ and $\langle h(E, m)\rangle = p\ W(E, m)$ tends to a constant value independent of E and m[24].

The phrase "flat histogram" means that the simulation histogram $h(E, m)$ for all visited combinations of energy E and magnetization m is not less than a percentage of the average histogram $\langle h(E, m)\rangle$ [23]. This enables us to consider that all values of energy and magnetization are visited by the method with almost equal probability. This is an advantage from the regular Monte Carlo algorithm as the Wang-Landau method is expected to be less affected by local energy barriers.

Initially, the Wang-Landau method is not a Markov process, as the matrix $W(E, m)$ is not known, but as the algorithm progresses the $W(E, m)$ matrix slowly converges to a constant value and the process becomes a Markov process, obeying the microscopic reversibility condition

$$p_i\ p_{(i\to j)} = p_j\ p_{(j\to i)}, \tag{3.1}$$

where $p_{(i\to j)}$ is the transition probability between spin configurations [24].

Defining $w(E, m) = \ln(W(E, m))$, initially we have $w(E, m) = 0$ for all possible states (E, m). Then, for each new random proposal of phase transition from a state i to a state r, we accept the proposal as the final j state with probability $p_A(i \to r)$, given by

$$p_A(i \to r) = \min(1, \exp(-\Delta w)), \tag{3.2}$$

where $\Delta w = w(E_r, m_r) - w(E_i, m_i))$.

The corresponding value of $w(E_j, m_j)$ is updated to be equal to its previous value plus $f > 0$. This update for the visited configurations results on a decrease of the probability of them being visited again, favouring the visit of the less visited states. From repeating this procedure a number of times such that all possible (E, m) values have been visited many times, the matrix $h(E, m)$ becomes what we consider to be flat [24].

Still, as the transitioning probability depends on the particular history of the simulation until then, this is not yet a Markov process and the function $w(E, m)$ is not equal to the microscopic entropy $S(E, m)$ [24]. In order to become a Markov process the values of the matrix $w(E, m)$ must become constant, and so, the value of f must tend to zero. To achieve this the value of f diminishes after all possible values of energy have been visited a number of times. As the simulation progresses, it is possible to estimate the extreme visited values of energy, E_{min} and E_{max}. For that we start by initializing E_{min} and E_{max} with the energy value of the system initial state and then update them as other values of energy are visited. It's considered that all the values of energy have been visited when we have gone between the lastly determined E_{min} and E_{max} a number n_f of times without updating them. Each time this happens the value of f is changed to $f/2$ [24]. The process is repeated a certain large number of times or until f is so small that can be considered zero.

For higher values of f the time for visiting all possible values of energy is $\propto N^2$. Meanwhile, for smaller values of f, if there are rare states (E, m) that weren't visited in previous steps, the algorithm can take a huge amount of time to produce a constant matrix $w(E, m)$, as there are necessary many visits to this state in order to equal the corresponding value of $w(E, m)$ with the w values of states previously visited [24].

3.2 Belief Propagation algorithm

Apparently different methods as the forward-backward algorithm, the Viterbi algorithm, iterative decoding algorithms for Gallager codes and turbocodes, Pearl's belief propagation algorithm for Bayesian graphs, the Kalman filter, and the transfer-matrix approach in physics are all special cases of the Belief propagation algorithm discovered in different scientific communities.

The Belief Propagation (BP) algorithm is based on 'message passing' through nodes and it was originally proposed by Judea Pearl in 1982 [25], who formulated it as an exact inference algorithm on tree-like graphs, allowing the marginals (single nodes probability distributions) to be computed much more efficiently than with a naive algorithm [9].

However, on graphs with loops the situation is much less clear [26]. Some researchers have empirically demonstrated good performance of the BP on graphs with loops when iterating the messages until convergence. This happens near the Shannon-limit performance of "Turbo codes" and for some problems in computer vision. On the other hand, for other graphs with loops, BP may give poor results or fail to converge. The intuition behind the relative success of the BP algorithm lies in the fact that, although the graph may not be a tree, 'locally' it may act like a tree. However, even in such cases, its applications are limited to distributions in which far apart variables become almost uncorrelated [9].

It must be noticed that, in some situations, the loops can be eliminated by clustering nodes of like type, that is, all variable nodes or all function nodes, as it is always possible to cluster nodes of like type without changing the graph global function. This enables the BP algorithm to compute marginals exactly. However, while a new function domain will result from the sum of the old functions domains without increasing the complexity of the algorithm,

a new variable domain, being given by the product of the old variables domains, can imply a substantial increase on the complexity of the algorithm [11].

3.2.1 Belief Propagation formulation on a factor graph

The seminal work of Yedidia [26] formulated loopy belief propagation in terms of optimizing the Bethe free-energy. The Bethe free-energy is an approximate Gibbs free-energy functional, as it considers a mean-field like approximation where the free-energy is written in terms of local quantities that approximate the partition function of the Ising model to its tree-like formulation [9]. Such a BP algorithm can only converge to a fixed point that is also a minimum point of the Bethe approximation to the Gibbs free-energy [10].

The Bethe approximation to the Gibbs free-energy [9] is expressed in terms of pseudo-marginals (called 'beliefs') by

$$\mathbb{F}^{BP} = -\sum_{a\in A}\sum_{\underline{x}_{\partial a}} b_a(\underline{x}_{\partial a}) \log\left[\frac{b_a(\underline{x}_{\partial a})}{f_a(\underline{x}_{\partial a})}\right] - \sum_{i\in I}(1-k_i)\sum_{x_i} b(x_i)\log(b(x_i)), \tag{3.3}$$

where k_i denotes the number of neighbouring function nodes of variable i, while $b(x_i)$ and $b_a(x_a)$ represent the estimated marginal distributions of a variable node i and of a function node a, respectively. This quantity's follow the constraint $\sum_{x_a\setminus i} b_a(x_a) = b(x_i)$.

On a factor graph the BP algorithm acts through two different types of messages: messages sent by variable nodes to its nearest neighbour function nodes ($v_{i\to a}$), and messages received by variable nodes from its nearest neighbour function nodes ($\hat{v}_{a\to i}$), always following the existing edges. These messages take values in the space of probability distributions over the single variable space X. Therefore, $v_{i\to a} = \{v_{i\to a}(x_i) : x_i \in X\}$, with $v_{i\to a}(x_i) \geq 0$ and $\sum_{x_i} v_{i\to a}(x_i) = 1$. The same happens with the $\hat{v}_{a\to i}$ messages [9].

To obtain good estimates of the variable's marginals on a loopy graph it is necessary to update the messages until its convergence. The BP messages are then computed for each iterative step t based on specified initial conditions, as described in equations 3.4 and 3.5 (where the symbol $\cong$ denotes 'equality up to a normalization' and the symbol $\setminus$ the set subtraction) [9][12]:

$$\hat{v}^{(t)}_{a\to i}(x_i) \cong \begin{cases} \sum_{\underline{x}_{\partial a\setminus i}} f_a(\underline{x}_{\partial a}) \prod_{j\in\partial a\setminus i} v^{(t)}_{j\to a}(x_j) & , if\ |\partial a\setminus i| > 0 \\ f_a(x_i) & , if\ |\partial a\setminus i| = 0 \end{cases} \tag{3.4}$$

$$v^{(t+1)}_{i\to a}(x_i) \cong \begin{cases} f_i(x_i) \prod_{b\in\partial i\setminus a} \hat{v}^{(t)}_{b\to i}(x_i) & , if\ |\partial i\setminus a| > 0 \\ f_i(x_i) & , if\ |\partial i\setminus a| = 0 \end{cases} \tag{3.5}$$

The message $\hat{v}_{a\to i}$ results from the sum of all possible states of the neighbouring variable nodes of function node a, except for the variable node i. This sum includes the product of all messages that the function node a receives from those neighbours. However, if i is the only

neighbouring variable node of function node a, there are no messages to consider and $\hat{v}_{a\to i}$ is only dependent on the action of the function node a.

Meanwhile, the message $v_{i\to a}$ results from the evidence of node i multiplied by the product of all the messages sent from the neighbouring function nodes of variable node i, except for the function node a. However, if a is the only neighbouring function node of the variable node i, $v_{i\to a}$ is only dependent on the evidence of the variable i.

It's expected that for $t \to \infty$ the messages had already converged to a fixed point. In this situation the message $v_{i\to a}^{(\infty)}(x_i)$ will give the marginal distribution of x_i on a modified graph that does not include the function node a. Analogously, the message $\hat{v}_{a\to i}^{(\infty)}(x_i)$ will coincide with the marginal distribution of x_i in a graph where all function nodes in ∂i, except a, have been erased.

After the convergence at an iteration step t, it can be estimated the marginal distribution $p(x_i)$ of a variable i as

$$p(x_i) \approx b(x_i) \cong f_i(x_i) \prod_{a\in\partial i} \hat{v}_{a\to i}^{*}(x_i) \tag{3.6}$$

using all its incoming messages $\hat{v}_{a\to i}^{(t)} = \hat{v}_{a\to i}^{*}$ (where '*' denotes the BP fixed point). Meanwhile [12],

$$b_a(x_a) \cong f_i(x_i) \prod_{i\in\partial a} v_{i\to a}^{*}(x_i) \ . \tag{3.7}$$

The previous BP equations used the potential $f_i(x_i)$ to include the preferred value of a variable node i. However, there is another way of fixing a variable value on a factor graph: adding a new function node, of degree 1 (with one nearest variable node), to the variable node whose value we want to affect. This way it can be considered an equal potential $f_i(x_i)$ for every possible state x_i of variable node i. Because of that, given the normalization, it can be considered that $f_i(x_i) = 1$ if the preferred value is not dependent on x_i.

The joint distribution of a subgraph R may be approximated by

$$p\left(\underline{x}_R\right) \approx b\left(\underline{x}_R\right) = \frac{1}{Z_R} \prod_{a\in F_R} \psi_a\left(\underline{x}_{\partial a}\right) \prod_{b\in F_{\partial R}} \hat{v}_{b\to i(b)}^{*}\left(x_{i(b)}\right) \tag{3.8}$$

with the condition that all function nodes b adjacent to R are only connected to one of its variable nodes (denoted by $i(b)$). Z_R denotes the subgraph partition function and F_R and V_R refer to the set of function nodes and the set of variable nodes, respectively, that belong to the subgraph R, in such a way that V_R includes all variable nodes adjacent to function nodes of F_R. The set of function nodes that are not in F_R but that are adjacent to a variable node in V_R is denoted by $F_{\partial R}$. This is illustrated in figure 3.1 [9].

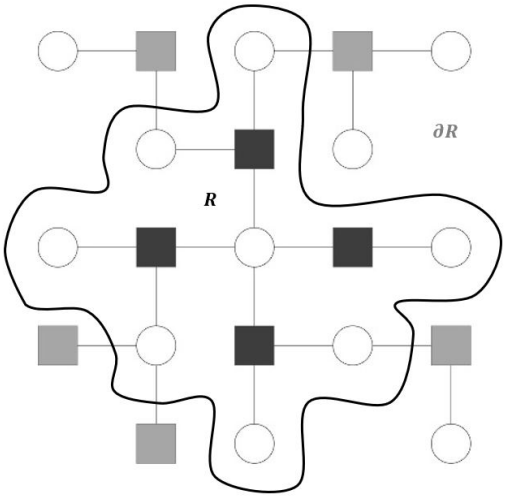

Figure 3.1: Representation of a subgraph R where all ∂R function nodes adjacent to it are only connected to one variable node from V_R.

With the knowledge of equation 3.8, the internal energy [9]

$$U = -\sum_{\underline{x}} p(\underline{x}) \sum_{a \in A} \log\left(f_a(\underline{x}_{\partial a})\right), \tag{3.9}$$

as the expected value of the total energy of the system, can be computed in terms of the BP messages as

$$U \approx -\sum_{a \in A} \frac{1}{Z_a} \sum_{\underline{x}_{\partial a}} f_a(\underline{x}_{\partial a}) \log\left(f_a(\underline{x}_{\partial a})\right) \prod_{i \in \partial a} v^*_{i \to a}(x_i), \tag{3.10}$$

for

$$Z_a = \sum_{\underline{x}_a} f_a(\underline{x}_a) \sum_{i \in \partial a} v^*_{i \to a}(x_i). \tag{3.11}$$

Also using equation 3.8 it is possible to express the free-entropy of equation 3.3 [9] in terms of fixed-point messages as

$$\mathbb{F}^{BP} = \sum_{a \in A} \mathbb{F}_a + \sum_{i \in I} \mathbb{F}_i - \sum_{(i,a) \in L} \mathbb{F}_{ia} \tag{3.12}$$

for

$$\mathbb{F}_a = \log\left[\sum_{\underline{x}_{\partial a}} f_a(\underline{x}_{\partial a}) \prod_{i \in \partial a} v^*_{i \to a}(x_i)\right], \tag{3.13}$$

$$\mathbb{F}_i = \log\left[\sum_{x_i} \prod_{b \in \partial i} \hat{v}^*_{b \to i}(x_i)\right] \tag{3.14}$$

and

$$\mathbb{F}_{ia} = \log\left[\sum_{x_i} v^*_{i \to a}(x_i) \hat{v}^*_{a \to i}(x_i)\right], \tag{3.15}$$

where L is the set of neighbouring nodes and functions.

The time complexity of the BP algorithm is $O(MT)$, where M is the number of edges and T is the number of iterations required until convergence. If the graph has loops, the number of iterations will be superior to one [27].

3.2.2 Belief Propagation formulation on a pairwise system

In a system where the interactions between variables happen in pairs, such as it happens in the Ising model, the function nodes of the corresponding factor graph are either connected to one or two variable nodes. Such a regularity enables us to ignore the function nodes and formulate the BP algorithm directly on a simple graph.

In this context the messages of equations 3.4 and 3.5 are reformulated as

$$\hat{v}^{(t)}_{j \to i}(x_i) \cong \sum_{x_j} f_{ij}(x_i, x_j) v_{j \to i}(x_j) \tag{3.16}$$

and

$$v^{(t+1)}_{i \to j}(x_i) \cong f_i(x_i) \prod_{r \in \partial i \setminus j} \hat{v}^{(t)}_{r \to i}(x_i) \tag{3.17}$$

respectively, which can be reduced to only one kind of message, either

$$\hat{v}^{(t)}_{i \to j}(x_j) \cong \sum_{x_i} f_i(x_i) f_{ij}(x_i, x_j) \prod_{r \in \partial i \setminus j} \hat{v}^{(t-1)}_{r \to i}(x_i) \tag{3.18}$$

or

$$v^{(t)}_{i \to j}(x_i) \cong f_i(x_i) \prod_{r \in \partial i \setminus j} \sum_{x_r} f_{ir}(x_i, x_r) v_{r \to i}(x_r). \tag{3.19}$$

Here the set ∂i refers, not to the group of neighboring function nodes of variable node i, but to the group of its neighbouring variables, while the function $f_{ij}(x_i, x_j)$ describes the interaction between the neighbouring variable nodes i and j.

With this new messages we can formulate the marginal probabilities as

$$p(x_i) \approx b(x_i) \cong f_i(x_i) \prod_{j \in \partial i} \hat{v}^*_{j \to i}(x_i) \ , \tag{3.20}$$

or alternatively, as

$$p(x_i) \approx b(x_i) \cong f_i(x_i) \prod_{j \in \partial i} \sum_{x_j} f_{ij}(x_i, x_j) v^*_{j \to i}(x_j). \tag{3.21}$$

For pairwise models, the free-entropy [9] [4] of equation 3.12 is written in terms of the messages $v^*_{j \to i}$ as

$$\mathbb{F}^{BP} = \sum_{i \in I} \mathbb{F}_i - \sum_{(i,j) \in D} \mathbb{F}_{ij} \tag{3.22}$$

for

$$\mathbb{F}_i = \log [\sum_{x_i} f_i(x_i) \prod_{j \in \partial i} (\sum_{x_j} f_{ij}(x_i, x_j) v^*_{j \to i}(x_j))] \tag{3.23}$$

and

$$\mathbb{F}_{ij} = \log [\sum_{x_i, x_j} v_{i \to j}(x_i) f_{ij}(x_i, x_j) v^*_{j \to i}(x_j)], \tag{3.24}$$

where D is the set of neighbouring variable nodes.

In the case of the Ising model, from equation 2.17 we have that $f_{ij}(x_i, x_j) = e^{\beta \sigma_i \sigma_j}$ and $f_i(x_i) = e^{\beta H \sigma_i}$. If $H = 0$, defining the relation

$$v^{(t)}_{i \to j}(\sigma_i) \cong \exp(\beta \sigma_i h^{(t)}_{i \to j}) \tag{3.25}$$

it is possible to parameterize the messages $v_{i \to j}$ by their log-likelihood ratio $h_{i \to j}$, also called a 'local magnetic field' which, following the physics convention, was rescaled by a factor $1/(2\beta)$. From equation 2.14 we have that the relation 3.25 can be interpreted as an 'effective' Boltzmann distribution for the spin σ_i and with an energy function $-\sigma_i h^{(t)}_{i \to j}$ [9].

The BP update equations are written in terms of the local magnetic fields $h_{i \to j}$ [9] as

$$h^{(t+1)}_{i \to j} = \sum_{k \in \partial i \setminus j} \frac{1}{\beta} [\tanh(\beta) \tanh(\beta h^{(t)}_{k \to i})], \tag{3.26}$$

After the convergence of the messages we obtain the cavity fields $\{h^*_{i \to j}\}$ from which we can compute the local marginals

$$p(x_i) \approx b(x_i) \cong \exp\left(\beta\sigma_i \sum_{k\in\partial i} \frac{1}{\beta}[\tanh(\beta)\tanh(\beta h^*_{k\to i})]\right). \tag{3.27}$$

The free-entropy of equation 3.22 can also be express in terms of the cavity fields $\{h^*_{i\to j}\}$ as

$$\begin{aligned}\mathbb{F}^{BP} = &\sum_{(ij)\in D} \log\cosh(\beta) - \sum_{(ij)\in D} \log\left(1+\tanh(\beta)\tanh(\beta h_{i\to j})\tanh(\beta h^*_{j\to i})\right) \\ &+\sum_{i\in I} \log\Big[\prod_{j\in\partial i}\left(1+\tanh(\beta)\tanh(\beta h^*_{j\to i})\right) + \prod_{j\in\partial i}\left(1-\tanh(\beta)\tanh(\beta h^*_{j\to i})\right)\Big].\end{aligned} \tag{3.28}$$

CHAPTER 4

Partitioning algorithms

The most common objective function in graph partitioning consists on the minimization of the number of edges between groups, given by the *cut size* as

$$R = \sum_{(i,j)} (1 - \delta(s_i, s_j)), \tag{4.1}$$

where $\delta(s_i, s_j)$ is the Kronecker delta

$$\delta(s_i, s_j) = \begin{cases} 1 \ if \ s_i = s_j \\ 0 \ if \ s_i \neq s_j \end{cases} \tag{4.2}$$

and s_i denotes the identification number of the group that includes the variable node i.

In this context, the simplest graph partitioning problem is the graph *bisection* our *bipartitioning* problem, as the division of a graph into just two parts. An exhaustive search of the optimal partition, that is, of the division with the smallest cut size, within the approximately $\frac{2^{N+1}}{\sqrt{N}}$ possible divisions, would turn out to be prohibitively costly. It would be only feasible to values of N up to about $N = 30$, as the time required to look through all the possible divisions would go up roughly exponentially with the size of the graph [1].

Because of this, other algorithms are used to solve the partitioning problem. Although they might fail to find the very best division of a system they may still find a pretty good one, and for many practical purposes that might be good enough [1].

Note that the cut size objective function does not take into account the different relevance of the relations between different variables. This way, the only relevant information for the graphical representation of a system is the existence or not of a connection (edge) between each pair of variable nodes, and therefore, there is no need to represent factor nodes.

In the following sections some clustering algorithms will be presented to solve bipartitioning problems.

4.1 Belief propagation partitioning

The graph partitioning problem can be solved by finding the ground state of the ferromagnetic Ising model at a fixed magnetization m (given by equation 2.16), as this is equivalent to dividing the graph into a group of spins $+1$ with size $n_+ = N(1+m)/2$ and a group of spins -1 with size $n_- = N(1-m)/2$. If the objective is to divide the graph into balanced groups the desired magnetization is zero [4].

The minimization of the cut size at a given magnetization is done by considering a cost function

$$b = \frac{R}{N}. \tag{4.3}$$

At $H = 0$, we have that the a cut size $R(m)$ is dependent on the magnetization value, as it is related with the system energy $E(m)$ by

$$R(m) = \frac{E(m) + M}{2}, \tag{4.4}$$

where M is the number of edges.

If it is found the system configuration with the minimum value of energy ($E(m) = -M$), at the zero temperature, then $b(m) = 0$. This means that a minimization of the cost corresponds to the minimization of the energy, and therefore, finding the ground state of the system. However, the energy at a fixed magnetization is not necessarily a convex function and finding the ground state energy can be challenging. A method of doing this is introducing a uniform external magnetic field H (possibly non-zero) and adjusting its value after every update of the BP messages, so that the spins orientations evolve in the direction of the desired magnetization value (m^d) [4].

Using the Legendre transformation $e(h) = e(m) - Hm$ [4] it is possible to relate the ground energy density $e(m) = E(m)/N$ at $H = 0$ with the ground energy density $e(H)$ at any given external magnetic field H, so that, having $M = N\alpha/2$, we obtain the cut size $R(H)$ by

$$R(H) = N\frac{e(H) + Hm + \alpha/2}{2}. \tag{4.5}$$

From equation 3.21 and 2.16, we have that the system magnetization m can be computed from the BP messages on a pairwise system by

$$m \approx \frac{1}{N}\sum_i \sum_{\sigma_i} \sigma_i b(\sigma_i) = \frac{1}{N}\sum_i \frac{\sum_{\sigma_i} \sigma_i e^{\beta H \sigma_i} \prod_{k\in\partial i}(\sum_{\sigma_k} e^{\beta\sigma_i\sigma_k} v^*_{k\to i}(\sigma_k))}{\sum_{\sigma_i} e^{\beta H\sigma_i} \prod_{k\in\partial i}(\sum_{\sigma_k} e^{\beta\sigma_i\sigma_k} v^*_{k\to i}(\sigma_k))}. \tag{4.6}$$

Like it was done in the previous section for the pairwise system, using the relation

$$e^{2\beta h_{i\to j}} \equiv \frac{v_{i\to j}(+1)}{v_{i\to j}(-1)}, \tag{4.7}$$

that describes the messages $v_{i\to j}$ in terms of a 'local magnetic field' $h_{i\to j}$, it is possible to deduce the iterative equation

$$h_{i\to j}^{(t)} = H + \sum_{k\in\partial i\setminus j} [\max(1 + h_{k\to i}^{(t-1)}, 0) - \max(h_{k\to i}^{(t-1)}, 1)] \tag{4.8}$$

from equation 4.6 at the zero temperature limit ($\beta \to \infty$) [4].

Note that the iterative equation 4.8 is equivalent to the replica symmetric equation 3.26 and that its term in the sum is -1 for $h_{k\to i} \leq -1$, $+1$ for $h_{k\to i} \geq 1$, and $h_{k\to i}$ for $-1 < h_{k\to i} < 1$ [4].

The BP total cavity field for node i is calculated by

$$h_i^{(t)} = \sum_{k\in\partial i} [\max(1 + h_{k\to i}^{(t)}, 0) - \max(h_{k\to i}^{(t)}, 1)]. \tag{4.9}$$

The computation of h_i with equation 4.9 results from the summation over the messages that are incident in node i, being that if $h_{k\to i} < -1$ the message is reduced to -1 and if $h_{k\to i} > 1$ the message is reduced to 1.

The solution of the zero temperature cavity equations described gives $m = 1$ for $H > 0$ or $m = -1$ for $H < 0$, with all spins either positive or negative, respectively. In order to compute the ground-state energy for $|m^d| < 1$, starting from an arbitrary value the external field H must be adapted after every iteration of the algorithm equations so that

$$m^d = (n_+^{(t)} - n_-^{(t)})/N, \tag{4.10}$$

where $n_+^{(t)}$ and $n_-^{(t)}$ are the number of positive and negative spins, respectively, obtained after each iteration t. The value n_+ is determined by the number of sites where $H + h_i > 0$ and the value n_- by the number of sites where $H + h_i < 0$. For the sites with $H + h_i = 0$ its attributed another spin category where the spin value can be either -1 or $+1$. The number of such spins is represented by n_0 and for any graph partition we have $n_+ - n_- - n_0 \leq m \leq n_+ - n_- + n_0$ [4].

Satisfying the equation 4.10 equality is equivalent to attributing to H the symmetric value of the total cavity field which separates the first $n_-^d = N(1 - m^d)/2$ smallest values of $h_i^{(t)}$ from the other $n_+^d = N(1 + m^d)/2$ biggest values [4].

At each iteration the messages may be updated by

$$h_{i\to j}^{(t)} = memory \times h_{i\to j}^{(t-1)} + (1 - memory) \times h_{i\to j}^{(t)} \tag{4.11}$$

considering a parameter of *memory* which prevents the messages from oscillating [4]. The algorithm terminates when the messages have converged or when it is achieved the maximum number of iterations.

Finally, in order to compute the partition cost we may either use the knowledge of the obtained system configuration (based on the values of H and h_i after the convergence of the algorithm) and obtain a partition cost b^{BP} or compute the Bethe estimate of the ground-state energy and from there compute the partition cost b_E^{BP} using the cut size given by equation 4.5.

From equation 3.10 it is obtain the Bethe estimate of the ground-state energy, given by

$$E(H, \{h_{i\to j}\}) = \sum_i E_i - \sum_{(i,j)} E_{ij}, \tag{4.12}$$

for

$$E_i = H + \alpha_i + 2\sum_{k\in i} \max(0, h_{k\to i}) - 2\max[H + \sum_{k\in i} \max(1 + h_{k\to i}, 0), \sum_{k\in i} \max(h_{k\to i}, 1)] \tag{4.13}$$

and

$$E_{ij} = 1 + 2\max(0, h_{i\to j}) + 2\max(0, h_{j\to i}) - 2\max(1 + h_{i\to j} + h_{j\to i}, 1, h_{j\to i}, h_{i\to j}), \tag{4.14}$$

where α_i is the degree of node i.

Note, however, that the BP estimate of the energy is only a lower bound on the true asymptotic energy. Therefore, while b^{BP} corresponds to the partition cost of the actual partition determined by the algorithm after convergence, b_E^{BP} corresponds to the partition cost that the algorithm predicts it would be the optimal, despite not having necessarily determined the partition that provides such a cost.

The pseudo-code of our formulation for the described algorithm may be find in section 7 under the name *Algorithm* 1. There is a problem with this algorithm, however. The magnetization value m^* of the determined partition may not correspond to the desired value m^d, either because the equations had not converged or because it was not found the actual ground-state energy of the system [4].

To get around this problem it is used a decimation technique which consists on repeating the previous algorithm and fixing spin values until the number of fixed spins to $+1$ or -1 agrees with the desired magnetization. For that, each time, after the convergence of the messages, a vertex i is chosen such that h_i is the highest (or lowest if at an even repetition number) and all outgoing messages from this node are fixed to $+\infty$ (or $-\infty$ for an even repetition number), and therefore, the spin i is fixed to $+1$ (or -1 at an even repetition number). The running time of this decimation solver is $O(N^2)$ [4].

Although such an algorithm can be applied on its own to perform graph partitioning, it is not expected that it will be competitive with the highly tuned implementations of the multi-level methods, that can be applied to graphs of one hundred million of nodes or even more [4] [28]. Even though, as the multi-level methods consist of grouping variable nodes into

super-nodes, super-nodes into bigger super-nodes and so on until the size of the system is small enough to solve the partitioning problem exactly, the belief propagation can be used as a component of this method by estimating the probability for which two nodes can be grouped in the same super-node [4].

4.2 Wang-Landau partitioning

Exploring the idea that the graph partitioning problem can be solved by finding the ground state of the ferromagnetic Ising model with a fixed magnetization, a new approach was developed in this work that uses not a Belief Propagation approach for inference but a Monte Carlo approach, in particular, the Wang-Landau method.

The advantage of this method from a regular Monte Carlo algorithm is that it enables the exploration of all possible configurations of the system without being trapped in a local energy minimum, as the best configuration for the partitioning problem may not correspond to a locally good solution of a local energy minimum. Ideally, the minimum value of energy would be found so the partitioning cost b^{WL}, computed from the determined system configuration, could be zero.

Given a certain desired value of magnetization m^d and a random system configuration with that value of magnetization, there are two possible ways of proposing a configuration update. One is proposing an interchange of spins between a random pair of opposite spin values, so the magnetization is conserved. Another is proposing the flip of a random spin with the condition that the magnetization value of the system does not go beyond a threshold of dm. This way we can either be fixating a magnetization $m = m^d$ or enabling the variation of the magnetization value in the interval $[m^d - dm, m^d + dm]$. The advantage of this last approach is that it may help the algorithm to find paths to new clusters of states.

The two approaches can be used alternately in the configuration proposal (update) part of the algorithm. After all the n_{MC} Monte Carlo steps were computed, the algorithm outputs the system configuration with the lowest energy among all the explored configurations with a magnetization that agrees with the desired one.

The pseudo-code used for the Wang-Landau partitioning algorithm can be seen in section 7 under the name *Algorithm* 2.

4.3 Extremal optimization for graph partitioning

The extremal optimization (EO) method is modelled after the Bak-Sneppen mechanism, introduced to describe the dynamics of co-evolving species, and is based on the dynamics of non-equilibrium processes, in particular, those who exhibit self-organized criticality (where better solutions emerge dynamically without the need for parameter tuning) [29].

In graph bipartitioning, the EO method considers each node i of the graph as an individual variable with its own "fitness" parameter $\lambda_i = s_i/(s_i + d_i)$, where s_i is the number of edges connecting i to other nodes of the same partition and d_i is the number of edges connecting i to nodes of the opposite partition. For isolated nodes we have $\lambda_i = 1$ [29].

Note that node i has a degree of $\alpha_i = s_i + d_i$, while the overall mean degree of a graph is given by $\alpha = \sum_i \alpha_i / N$ and the cut size of a configuration by $R = \sum_i d_i / 2$ [29].

At all times, an ordered list of the nodes fitness is maintained in the form of a permutation of its labels as $\lambda_{\Pi(1)} \leq \lambda_{\Pi(2)} \leq ... \leq \lambda_{\Pi(N)}$, where $i = \Pi(n)$ is the label of the nth ranked node in the list [29].

In order to do a local search in the configuration space it is defined the "neighborhood" of configurations for which each configuration of the space can be updated to over the course of t_{max} update steps. Such neighbourhood is obtained by the exchange of any two nodes between the partitions of the current configuration [29].

The algorithm was found to wield better results when the exchange is done between node $i_1 = \Pi(n_1)$ and node $i_2 = \Pi(n_2)$, given the integers $n_1, n_2 \in [1, N]$ drawn from the probability distribution $P(n) \propto n{-}\tau$, for $\tau = 1.4$. Note that the drawing of n_2 is done until i_1 and i_2 don not belong to the same partition [29] [30].

After each update the fitness rank list is also updated by reevaluating the fitness of the nodes i_1 and i_2 and of all the nodes they are connected to. Using a 'heap', the computational cost of such an operation is $O(\alpha \ln N)$.

Note that, unlike in a greedy algorithm, that moves towards the configurations that yield an improvement from the last ones, in the EO method all moves are accepted, even if they are worst, as the selection follows a scale-free power-law distribution. The best solution (with the smallest cut size) is then determine from all the t_{max} explored configurations, for t_{max} linear with the system size [29]. The partitioning cost b^{EO} is computed from the smallest cut size.

The EO method for graph bipartitioning yields state-of-the-art solutions, even for systems of $N > 10^5$ variables. In particular, for large graphs of low connectivity, it has been shown to be faster than genetic algorithms and more accurate than simulated annealing, which are other two widely applied methods [29].

CHAPTER 5

Bipartition analysis

5.1 Erdos-Rényi random graphs

The simplest characteristic of a node in a graph is its degree α. Properties like the degree distribution of a graph can have huge effects on the behaviour of systems. Therefore, one of the best ways to understand the graphs behaviour is to build mathematical models of the systems of interest [1].

To generate a random graph with a mean degree α we can use the Erdos-Rényi model, where all possible graphs with N nodes and M edges have equal probability of realization [31]. This is equivalent to choose, with uniform probability, M of the $N(N-1)/2$ possible edges between the N nodes, generating a graph with a mean number of nearest neighbours given by $\alpha = 2M/N$.

The Gilbert model is also a model of generation of random graphs. In this model, between each pair of nodes an edge is present with a fixed probability p, and the mean total number of edges is fixed to $< M >= pN(N-1)/2$ [31].

For graphs with a high number of nodes, the Gilbert model with $p = \alpha/(N-1)$ is equivalent to the Erdos-Rényi model. This happens because, in the Gilbert model, the fluctuations of M go to zero as N grows ($< M > \to M$).

5.1.1 Random graphs bipartitioning

A prominent feature of the Erdos-Rényi model is the presence of a giant component, as the biggest connected component of the graph for $\alpha > 1$.

For Erdos-Rényi random graphs with $N \to \infty$ vertices and mean degree α, the size of the largest component is gN at the limit of large N, being g the fraction of nodes on the giant component, which satisfies $g = 1 - \exp(-\alpha g)$ [4].

In order to divide the graph into two parts of sizes $n_+ = N(1+m)/2$ and $n_- = N(1-m)/2$ such that the partitioning cost is zero, all the giant component nodes must present the same spin value. If $n_+ \geq n_-$, the condition $g \leq n_+/N$ has to be verified, and therefore, the size of

the giant component must be at maximum $(1+m)/2$. That is possible for average degree $\alpha \leq \alpha_S$ where [4]

$$\alpha_S = -\frac{2}{1+m} \ln \frac{1-m}{2}. \tag{5.1}$$

As for $\alpha > \alpha_S$, the minimal bipartition will not have a cost of zero as there are edges in the giant component that still need to be cut in order to obtain the minimal bipartition. The value α_S is hence named the 'satisfiability threshold' for graph partitioning on Erdos-Rényi random graphs [4].

An Erdos-Rényi random graph of average degree $\alpha < 1$ basically looks like a collection of small disconnected trees, what makes possible to split the graph into small components with the same spin values, so the cost of the partition is zero for all magnetization values. In this situation the BP partitioning algorithm is expected to converge to a fixed point with $H = 0$ [4].

For $1 < \alpha < \alpha_S$, the giant component has all spins either positive or negative and the rest of the graph can be split into two properly sized groups with the same spin values and zero partitioning cost, as the small components are numerous. In this situation the BP partitioning algorithm is expected to present local fields with the same sign inside each component [4].

After the satisfiability threshold ($\alpha > \alpha_S$) the giant component is bigger than the number of nodes that are in the larger of the two groups, so there will be necessarily neighbouring nodes with opposite spin values on the giant component, which will be split into two groups. All the other components of the graph will present the spin value that was less present in the giant component and they will not contribute to the partitioning cost. In this situation the BP partitioning algorithm is expected to present positive and negative local fields in the giant component and a nonzero external field [4].

5.1.2 Improving the Wang-Landau partitioning method on random graphs

Based on these observations, a way of solving the partitioning problem would be, firstly, to identify all the components of the graph and determine the group G of nodes of the giant component. Only then, if $\alpha > \alpha_S$, would be applied the partitioning algorithm only to the giant component. Otherwise, for other values of α, a way of determining the nodes spin values outside the giant component would be to progressively assign to all the nodes of the next non-solved largest component the spin value that in greater number waits to be assigned to the graph. This procedure yield the correct result because, as it is already known, for $\alpha < \alpha_S$ it exists a system configuration with cost zero where all the nodes of the same component have the same spin.

This logic was applied to the Wang-Landau partitioning method proposed in this work. Therefore, when the desired valued of magnetization implied that n_+ or n_- would be greater than the giant component size N_G, the partitioning cost was simply attributed to zero.

Meanwhile, if n_+ and n_- are less than N^G, the number of positive spins in the giant component is $n_+^G = N^G - n_+$ if $n_+ > n_-$, or $n_+^G = n_+$ otherwise, while the number of

negative spins is $n_-^G = N^G - n_+^G$. In this context, there is also a desired local magnetization of $m^G = n_+^G - n_-^G$ on the giant component. It is with this magnetization value that we apply the Wang-Landau partitioning algorithm to a subgraph made only with the giant component.

From the ground state energy E^G computed from such a subgraph, being $E = E^G + E^S$ (where $E^S = -M^S = M^G - M$ is the known ground state energy of the set S of all small components), from equation 4.4 we get the partition cost of the original graph by

$$b^{WL} = \frac{E^G + M^G}{2N}. \tag{5.2}$$

5.2 Regular random graphs

A particular case of random graphs is the situation of regular graphs, where every node presents the same fixed degree. This is the hardest case for graph bipartitioning, as the graphs look locally alike from every point and no apparent structure can be explored to decide if two nodes should be in the same group or not [4].

In such situations, the graph only presents one big component and the lower the magnetization value $|m^d|$ the greater the number of edges that have to be cut, and so, the greater the partitioning cost. Only for $|m^d| = 1$ we have a zero cost. For this reason, it is not possible to improve the Wang-Landau performance as described before. It is necessary to run the algorithm for any given value of magnetization in order to infer about the best partitioning cost.

CHAPTER 6

Numerical results

6.1 Results on Erdos-Rényi random graphs

Firstly, in order to validate the results obtained with our Algorithm 1 we compared them with the ones presented in [4] for the BP partitioning with no decimation on Erdos-Rényi random graphs. We will refer with the suffix *art* to every results taken from Ref. [4].

Algorithm 1 was run with the parameters $memory = 0.7$, $nIterMax = 10000$ and $eps = 0.00001$, being that it was found that it work the best when first applied to the higher magnetization values of the range of interest. As we will comment further in this section, at higher values of magnetization it takes less iterations to the BP algorithm to converge. Then, for the next highest value of magnetization, we can skip the messages random initialization and instead preserve the ones that had already been converged in the previous simulation, facilitating the new convergence process.

Computing the BP energy estimated cost (b_E^{BP}) for different values of magnetization on graphs of size $N = 100000$ and mean degree $\alpha = 1.44$ and 1.60, we observed in figure 6.1 a good accordance with the $b_E^{BP^{art}}$ values when averaging over 8 random graph samples. It must be noticed, however, that different random graphs present different optimal partition costs, and so, the particular set of graph samples influence the values of partitioning cost observed.

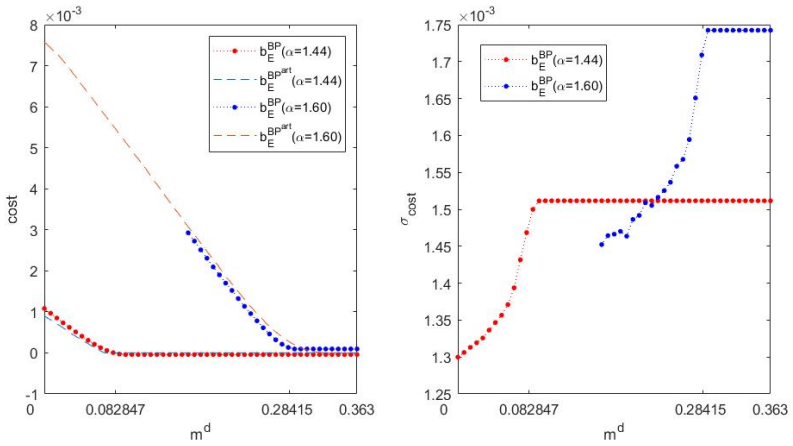

Figure 6.1: On the left, the partition cost b_E^{BP} was obtained from Algorithm 1 by averaging over 8 samples of random graphs with size $N = 100000$ and a particular mean degree (either $\alpha = 1.44$ or $\alpha = 1.60$). The $b_E^{BP^{art}}$ values were taken from Ref. [4] for the same N. The number of samples used in Ref. [4] is not clear. On the right it is represented the standard deviation on the set of 8 samples for the partitioning costs obtained for each magnetization value.

Moreover, it is verified an agreement between the value of magnetization $m^d = m_S$ at which, from equation 5.1, we have $\alpha_S(m_S = 0.082847) \approx 1.44$ and $\alpha_S(m_S = 0.28415) \approx 1.60$, and the magnetization value where, in figure 6.1, the evolution of the partition cost changes from an almost constant behaviour near the zero cost to a gradual increase, as for $m^d < m_S(\alpha)$ there is a connected component, the giant component, that starts including both spin values.

From all these observations we conclude that the results obtained with the developed implementation of Algorithm 1 are in agreement with the results of Ref. [4].

From the set of 8 random graphs generated, it is shown in figure 6.1 the standard deviation of the b_E^{BP} values. This quantities, σ_{cost}, tend to diminish as the partition cost increases for smaller values of magnetization. Therefore, the results obtained with the BP algorithm at $m^d = 0$, that is, for the balanced bipartitioning, are the most accurate ones.

After validating our BP results of Algorithm 1 we were able to compare them with the ones obtained with the WL partitioning of Algorithm 2 (run for $dm = 2$, $intm = 2$ and $n_{MC} = 10^7$). This time, as the WL partitioning tends to be slower than the BP partitioning, we reduced the size of the random graphs to $N = 10000$, while increasing the number of samples to 100. We also computed the BP actual cost (b^{BP}) values, for the same N, in order to compare them with the WL actual cost (b^{WL}) values.

From figure 6.2 it is observed that b^{WL} is higher than b_E^{BP} for $m^d < m_S(\alpha)$. For $m^d > m_S(\alpha)$, most of the generated graphs presented a giant component size smaller than either n_+ or n_-, as it was expected from equation 5.1, so the b^{WL} costs were simply attributed to zero. Because of this, we have limited ourselves to compare the WL and BP results for $m^d < m_S(\alpha)$.

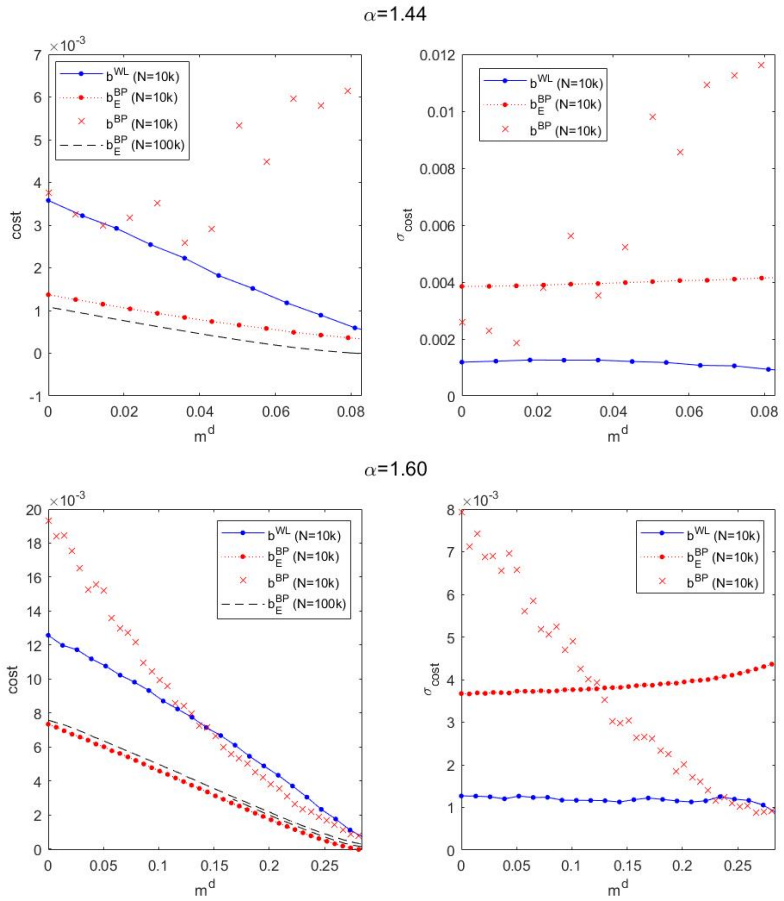

Figure 6.2: On the left column are the results obtained when averaging over 100 samples of random graphs with size $N = 10000$ and mean degree $\alpha = 1.44$ (for the upper row) or $\alpha = 1.60$ (for the lower row). The partitioning costs b_E^{BP} and b^{BP} were computed using Algorithm 1 and the partitioning costs b^{WL} using Algorithm 2. Were also included the b_E^{BP} results obtained averaging over 8 samples of random graphs with size $N = 100000$ and the respective mean degree α. On the right column it is represented the standard deviation from the 100 samples partition costs obtained at each magnetization value. Note that the cost values were only represented for $m^d < (m_S = 0.082847)$ if $\alpha = 1.44$ or for $m^d < (m_S = 0.28415)$ if $\alpha = 1.60$, according to equation 5.1.

When comparing the WL results with the partition cost of the actual partition found by the BP algorithm at this magnetization range, we observe that b^{WL} tends to be similar or lower than b^{BP}, for every magnetization value. This observations make us conclude that, as defended in the article [4], the b_E^{BP} results work as a lower bound to the true optimal partitioning costs, as they are lower than b^{BP} and b^{WL}.

Being that the WL algorithm presents a lower or similar standard deviation than the BP algorithm, and therefore, more stable results across different samples, we conclude that the WL actual costs are in general better than the BP actual costs of Algorithm 1. This was expected as Algorithm 1 is not guaranteed to provide a solution with a magnetization that agrees with the desired one, while the WL partitioning of Algorithm 2 only gives solutions that agree with the desired magnetization.

Moreover, it was expected that, for higher magnetization values, the existence of small

components should make the determined best costs to vary close to the zero value, while for smaller magnetization values the different ways of partitioning the giant component should provide very different partitioning costs. For different α values, the standard deviation values of the WL actual costs are more consistently agreeing with this behaviour than the standard deviation values of the BP actual costs, as in figure 6.2 these last ones are higher for higher magnetizations if $\alpha = 1.44$.

From figure 6.2 it is also observed a slight size variation of b_E^{BP} for $N = 100000$ and $N = 10000$, which is an increase or a decrease depending on the value of α considered. Anyway, the magnetization at which the partition costs changes from being almost constant to a non-zero value its preserved, what agrees with its formulation on equation 5.1, where it is not dependent on the system size.

In figure 6.3 it is possible to observe, at $m^d = 0$, the variation of the costs for different mean degrees α.

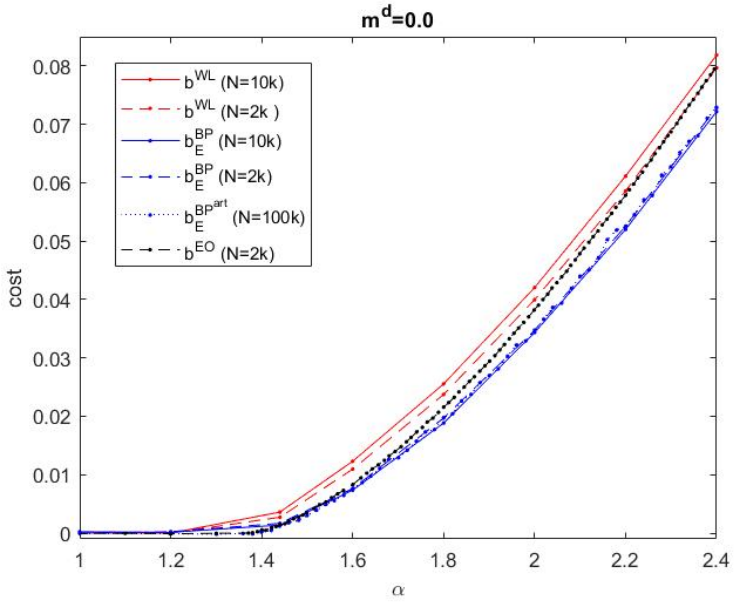

Figure 6.3: α dependence of the partition cost at $m^d = 0$ (for $\alpha = 1.00, 1.20, 1.44, 1.60, 1.80, 2.00, 2.20$ and 2.4). The b_E^{BP} and b^{WL} values were obtained averaging over 100 samples, while the $b_E^{BP^{art}}$ were obtained from Ref. [4] averaging over 2 samples. The b^{EO} values were also obtained from Ref. [4].

While over the α range the b_E^{BP} values remain roughly the same for different system sizes, the b^{WL} values tend to be bigger for bigger systems. Also, we see at $\alpha = 1.00$ and 1.20 that the b_E^{BP} costs computed for $m^d = 0$ start in values near zero and increase for $\alpha \geq 1.44$. This is in accordance with equation 5.1, as $1.20 < (\alpha_S(m = 0) = 1.3863)$, and therefore, we expect a partition cost of zero for this mean degrees, not just for $m^d = 0$ but for all magnetization values. Such an observation is done in figure 6.4 for 3 different magnetization values.

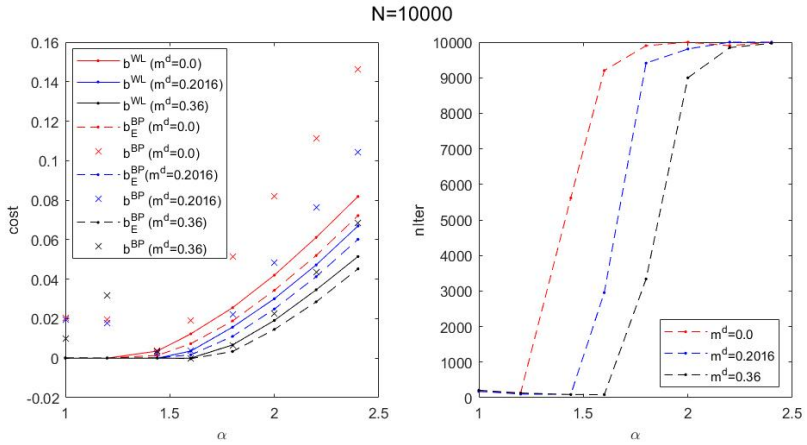

Figure 6.4: α dependence of the partition cost (for $\alpha = 1.00, 1.20, 1.44, 1.60, 1.80, 2.00, 2.20$ and 2.4). The b_E^{BP}, b^{BP} and b^{WL} values were obtained averaging over 100 samples of $N = 10000$ graphs at different magnetization values ($m^d = 0.0, 0.2016$ and 0.36). On the right is represented the mean number of iterations done by Algorithm 1 for a maxim number of 10000 iterations.

However, it must be noticed that the b^{BP} values do not confirm the expected behaviour for $\alpha < (\alpha_S(m = 0) = 1.3863)$, where the BP algorithm fails to give the optimal partitions. In any case, we cannot point out this problem as a convergence problem as from figure 6.4 it is observed that the mean degree α_C, at which the BP equations stop converging (known as the glass transition), is always superior to $\alpha_S(m = 0) = 1.3863$ for every value of magnetization m^d [4].

In fact, from the results of figure 6.4 it was verified that for $m^d = 0.0$ we have $\alpha_C \approx 1.8$ and that for higher values of magnetization this value grows. In other words, for the same mean degree α the number of iterations decreases at higher magnetization values, so even if $\alpha \geq 1.80$ the BP partitioning may still converge for higher magnetization values. Moreover, for a given value of magnetization, the α value at which the partitioning costs start increasing matches the α value at which the number of iterations also increases from an almost constant number.

Another observation from both figure 6.3 and 6.4 is that b_E^{BP}, given by the BP algorithm, is a lower bond of the exact result [4], and so, it remains similar or lower than the results obtained by the EO and WL algorithms. For graphs with the same size $N = 20000$, the WL results tend to be close to the EO results, although higher (specially for smaller α values), and smaller than the b^{BP} values. From this we conclude about the better quality of the WL partitions on random graphs in comparison with the BP partitions.

However, do the WL results remain better than the BP results when applying decimation to the BP partitioning? From the results of figure 6.5 we see that, for two different random graphs, the BP with decimation actual cost ($b^{BP_{dec}}$) values vary around the WL results, being slightly better in some cases and slightly worst in others. What distinguish the two results, besides the more regular evolution of the WL costs over the magnetization spectrum, is the fact that the WL partitioning is the only algorithm that ensures the agreement between the actual magnetization of the determined partition and the desired magnetization. On the BP

decimation algorithm the certainty about this agreement increases as the number of spins that were fixed increases for smaller values of magnetization. At $m^d = 0$ all the spins have been fixed, while at higher values of m^d just a few spins had been fixed. Because of this, the BP decimation ran starting not at the highest values of magnetization but at $m^d = 0$, followed by progressively higher m^d values.

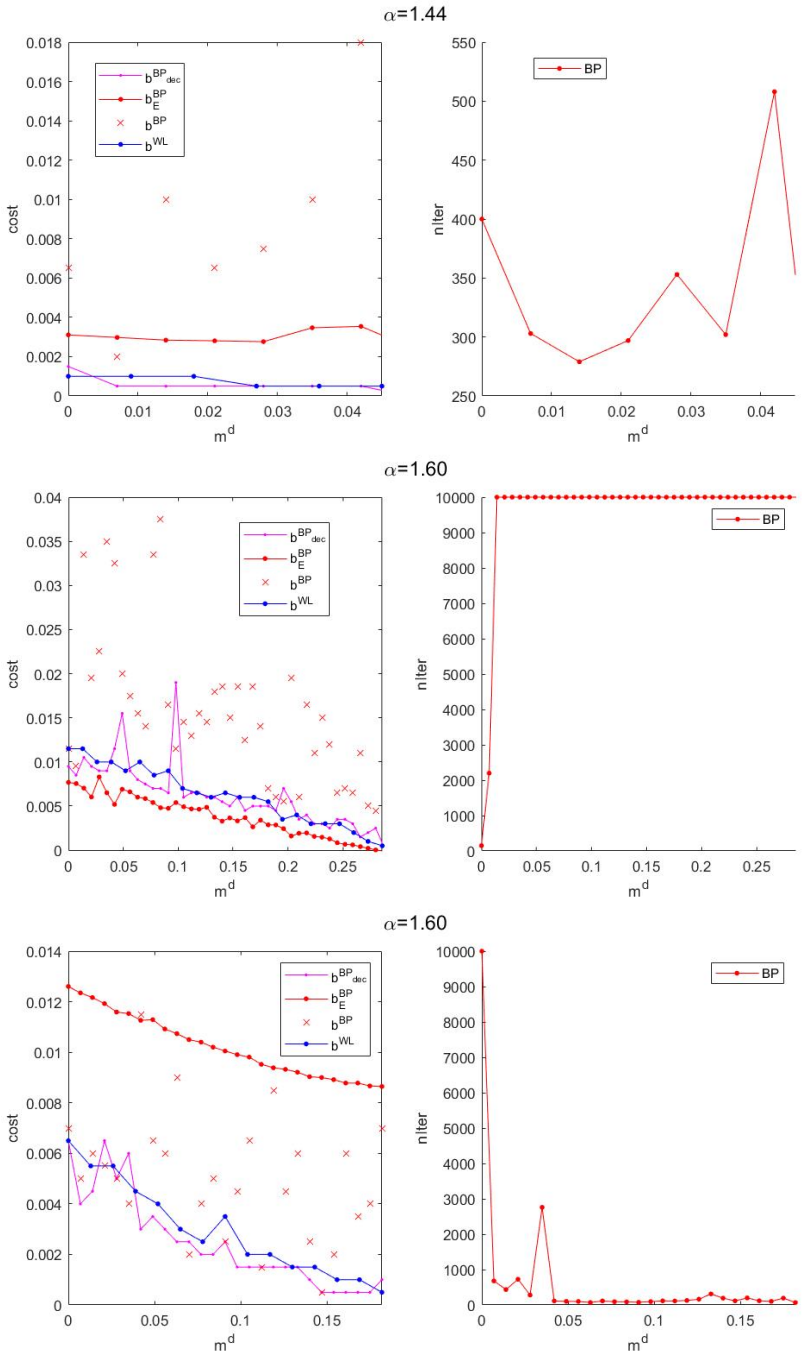

Figure 6.5: In each row are presented the results obtained for one different random graph with size $N = 2000$ and mean degree $\alpha = 1.44$ or $\alpha = 1.60$. The $b^{BP_{dec}}$ values were obtained applying the decimation technique to Algorithm 1. Note that the b^{WL} values were only represented if greater than zero. In the right column is presented the number of iterations done by Algorithm 1, for a maxim number of 10000 iterations.

It must also be noticed that the WL partitioning can be much faster than the BP decimation. In fact, the b^{WL} values computed in figure 6.5 were obtained at least 26 times faster than the $b^{BP_{dec}}$ values. This is understandable as in order to obtain the BP decimation results it takes $2n_-$ repetitions of Algorithm 1 if $n_- < n_+$ or $2n_+ - 1$ repetitions otherwise.

Moreover, for the first two graphs presented in figure 6.5, we once again verify the lower bond characteristic of the b_E^{BP} values, even when the algorithm has reached the maximum number of iterations without converging (as seen for the graph in the middle row of figure 6.5). However, for some other graphs (as for the last one presented in figure 6.5), the lower bond characteristic of the b_E^{BP} values was not verified, probably because of the lack of convergence to the desired magnetization value. It is possible that the same problem has been responsible for negative values of b_E^{BP} computed for some other graphs that were not presented in figure 6.5. In any case, this topic still requires further study.

All these observations bring up the limitations of the BP partitioning with no decimation, particularly when we are interested in using it as a problem solver on a particular graph.

6.2 Results on regular random graphs

Some results of the BP partitioning with decimation were presented in [4], but only for random regular graphs. Therefore, in order to validate our implementation of this algorithm we proceed by applying it to several random regular graphs.

In figure 6.6 it is possible to observe the similarity of results between our implementation and the Ref. [4] results, being our $b^{BP_{dec}}$ results similar or slightly smaller than the $b^{BP_{dec}^{art}}$ values.

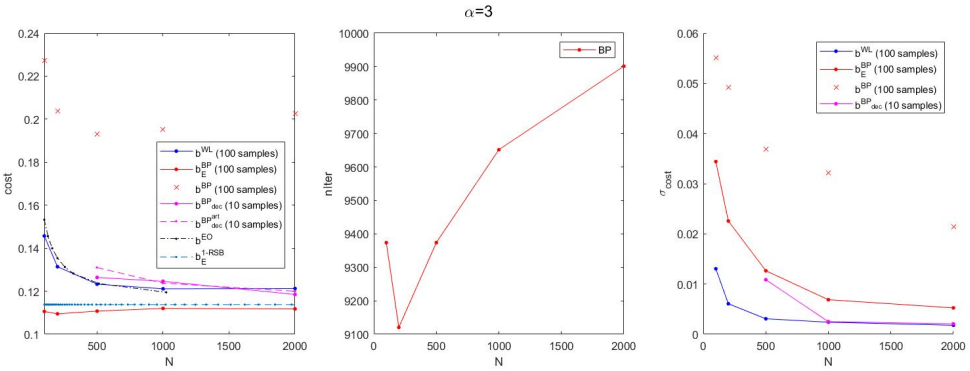

Figure 6.6: On the left it is represented the partitioning cost at $m^d = 0$, for different values of N ($N = 100, 200, 500, 1000$ and 2000), averaged over random regular graphs with $\alpha = 3$. The $b^{BP_{dec}^{art}}$ and b^{EO} values were obtained from Ref. [4], the last ones using the 1-Replica symmetry breaking algorithm, which like the BP partitioning, provides energy estimated partitioning costs. On the middle of the figure is represented the mean number of iterations done by Algorithm 1, for a maximum number of 10000 iterations. On the right the standard deviation is presented for the set of samples for the partition costs obtain for each system size.

Overall, for $N \geq 500$, both the $b^{BP_{dec}}$, $b^{BP_{dec}^{art}}$, b^{WL} and b^{EO} values are very close, being the WL and EO results particularly similar for $N \leq 1000$, where they are actually the smallest

actual partitioning costs.

We see that all algorithms present smaller standard deviation for higher values of N, being the WL partitioning the one with the smaller standard deviation over all the N values. Even though, the standard deviation of the $b^{BP_{dec}}$ values approaches the standard deviation of the b^{WL} for bigger systems.

Evaluating the efficiency of the BP partitioning with no decimation on regular graphs we observe that, although still being b_E^{BP} a lower bound for b^{WL}, b^{EO} and $b^{BP_{dec}^{art}}$, the corresponding b^{BP} values are much higher than all the other results presented.

Moreover, for 100 samples of $\alpha = 3$, the possibility of occurring convergence problems increases for higher values of N, as the number of iterations approaches the maximum. The same observation is done in figure 6.7 for $N = 2000$, being that the effects of the lack of convergence seem to worsen for higher degrees α.

For comparison, results from the 1-Replica symmetry breaking (1-RSB) method from Ref. [4] were also shown in figure 6.6, providing a more accurate cost lower bound than the BP partitioning. Still, given that the BP and 1-RSB results are relatively close to each other for $\alpha = 3$, solving the BP partitioning may be preferable to solve the often complex 1-RSB equations.

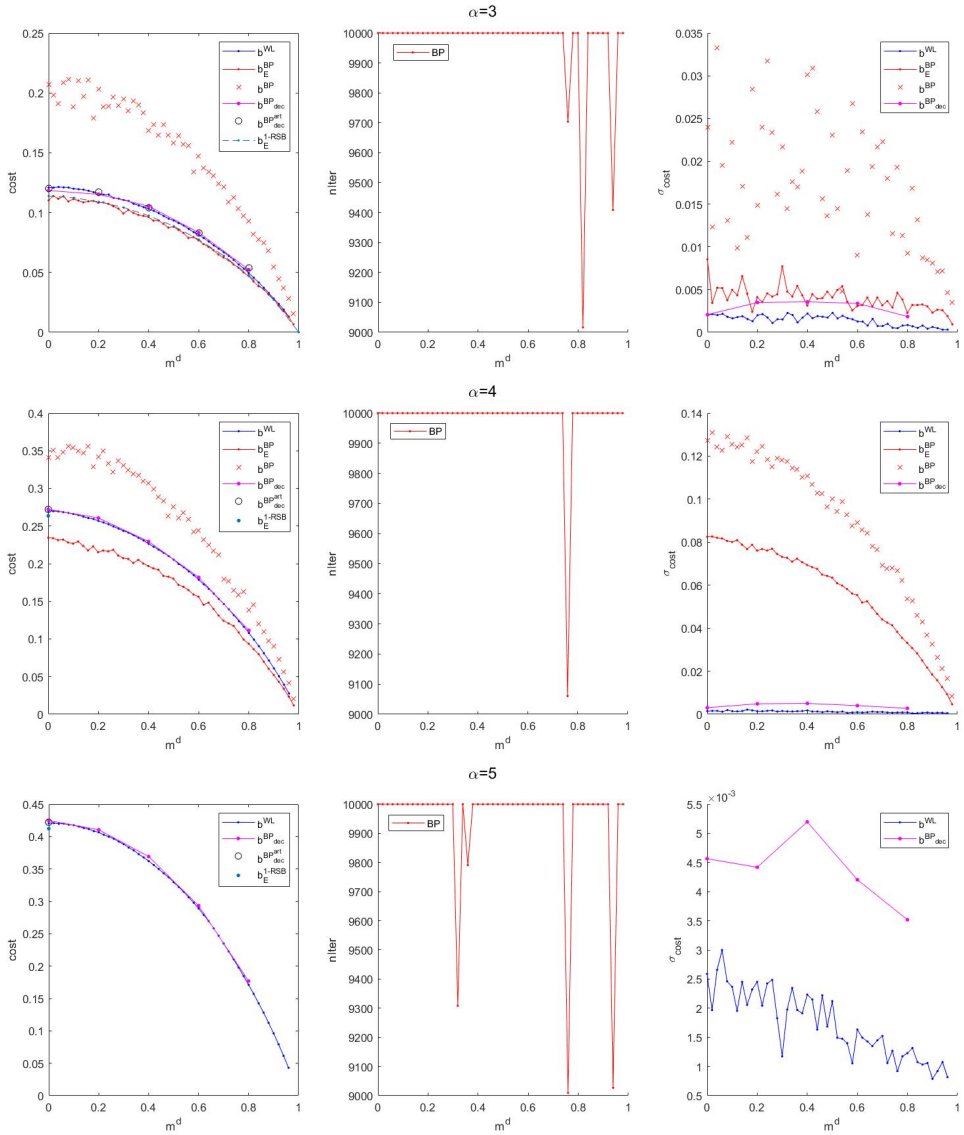

Figure 6.7: On the left column it is represented the dependence of the partitioning cost over the magnetization range, resulting from the average over 10 random regular graphs with size $N = 2000$. The $b^{BP^{art}_{dec}}$ and the b_E^{1-RSB} values were obtained from Ref. [4], the last ones using the 1-Replica symmetry breaking algorithm for $N = 30000$. On the middle column it is represented the mean number of iterations done by Algorithm 1, for a maximum number of 10000 iterations. On the left column it is presented the standard deviation of the 10 samples for the partition costs obtain for each magnetization value. For $\alpha = 5$ the cost results of the BP partitioning with no decimation were not represented as they diverge to much from the other results (probably because of convergence problems), lacking proper meaning.

In figure 6.7, once again, we confirm for $\alpha = 3$ the agreement between our $b^{BP_{dec}}$ values and the $b^{BP^{art}_{dec}}$ values of Ref. [4], which are very close to the WL results over all the magnetization range considered. Note that in Ref. [4], for $\alpha = 4$ and 5, only results of $b^{BP^{art}_{dec}}$ and b_E^{1-RSB} for $m^d = 0$ were presented, and they agree with our $b^{BP_{dec}}$ and b^{WL} values computed for the full range of m^d values.

As expected, the partitioning costs computed with any of the algorithms for $m^d < 1$ are

non-zero, and with the exception of the BP decimation, in which there is a fixation of spins, all the algorithms tend to present smaller standard deviations for bigger magnetization values. This is understandable as there are more different ways of performing a balanced partitioning over a regular graph than a partitioning where one of the groups is small and the other large. However, note that if we were dealing with larger regular graphs it would be expected that the division of different large samples at the same magnetization would yield approximately the same partitioning cost.

We also observe that the WL partitioning is the algorithm that provides the smaller σ_{cost} values, and therefore, it provides the most stable results across different samples. Moreover, the WL results computed for figure 6.7 were obtained at least 29 times faster than the results of the BP partitioning with decimation.

In figure 6.7, for $\alpha = 3$ and 4, we observe that the partition costs b^{BP} are never smaller than either the WL or the BP decimation partitioning costs and that, once again, the lower bound characteristic of the b_E^{BP} values is verified. However, while for $\alpha = 3$ the BP lower bound is very close to the 1-RSB lower bound, for $\alpha = 4$ the BP lower bound at $m^d = 0$ is lower and more distant from the 1-RSB's. When it comes to $\alpha = 5$ the b_E^{BP} values were found to be much higher than all the other algorithms partitioning costs, probably because of the worsening of the convergence problems.

Also, from $\alpha = 3$ to $\alpha = 4$, there is an increase of the σ_{cost} values for b_E^{BP} and b^{BP}, and for $\alpha = 5$ this values are even higher. Given this, for $\alpha = 5$ the BP partitioning only gives meaningful results if computed with a decimation technique.

From table 6.1, at all degrees α considered we have confirmed the lower bound characteristic of the 1-RSB costs, which are lower than the WL and BP decimation actual costs. Also, while for $\alpha = 3$ the WL results are higher than the BP decimation results, for $\alpha \geq 4$ the WL results are actually slightly lower. So, while we can agree with the observation that the optimal partitioning costs must be found somewhere between the b_E^{1-RSB} and $b^{BP_{dec}}$ values, as it is stated in Ref. [4], we can be more concrete and say that for $\alpha \geq 4$ the exact optimal partitioning costs must be found somewhere between the b_E^{1-RSB} and b^{WL} values. In any case, the difference between the WL and BP decimation results for a balanced partitioning is less than 0.002 for any of the degrees considered.

α	b_E^{1-RSB}	b^{WL}	$b^{BP_{dec}^{art}}$	$b^{BP_{dec}}$
3	0.113 846	0.1209	0.1180(3)	0.1185
4	0.263 527	0.2701	0.272(1)	0.2722
5	0.412 398	0.4206	0.422(2)	0.4248

Table 6.1: Partition cost values at $m^d = 0$ for random regular graphs with different degrees α. The $b^{BP_{dec}^{art}}$, $b^{BP_{dec}}$ and b^{WL} values were obtained from a set of 10 graphs of size $N = 2000$, while the b_E^{1-RSB} solutions were obtained from Ref. [4] using the 1-Replica symmetry breaking algorithm for $N = 30000$. The $b^{BP_{dec}^{art}}$ were also obtained from Ref. [4].

CHAPTER 7

Conclusion

In this thesis some of the Belief Propagation algorithms proposed in Ref. [4] were implemented. Furthermore, the Monte Carlo algorithm of Wang-Landau was adapted to simulate the Ising model on random graphs and applied, for the first time, to the problem of graph bipartitioning.

The results obtained in both cases were found to be in good agreement with those reported in Ref. [4]. Particularly, when averaging over a set of Erdos-Rényi random graphs, was verified the expected cost lower bound characteristic of the b_E^{BP} results, which were lower than the b^{BP}, b^{WL} and the known b^{EO} actual costs. However, when considering a particular graph or even when averaging over random regular graphs of $\alpha = 5$, there were some situations where the b_E^{BP} costs did not represent a cost lower bound, probably because of the lack of convergence to the desired magnetization.

In fact, while in many cases the BP algorithm can be used to estimate a lower bound to the ground state energy, and therefore, the lower possible partitioning cost b_E^{BP}, it is not as successful in determining the graph partition with either that ground state energy or the desired magnetization. The BP actual partitioning costs b^{BP} tend to be considerably higher than the optimal b_E^{BP} estimated cost. To obtain reasonable actual costs it is necessary to apply a decimation technique to the BP algorithm, as proposed in Ref. [4].

From the similarity between the b^{WL} and $b^{BP_{dec}}$ results for arbitrary asymmetric partition sizes (not considered in Ref. [4] for $\alpha = 4$ and 5), and of them with the known b^{EO} results (which were often still slightly lower), was possible to conclude about the reasonable quality of the BP decimation and WL results. Actually, it was already stated in Ref. [4] that the exact optimal partitioning cost was expected to be found somewhere between the b_E^{1-RSB} and the $b^{BP_{dec}}$ cost values. From this work it was found that on random regular graphs with $\alpha = 4$ or 5 we have $b^{WL} \lesssim b^{BP_{dec}}$, so we can more precisely say that the exact optimal partitioning cost can be found somewhere between b_E^{1-RSB} and b^{WL}. However, for $\alpha = 3$ we have $b^{WL} \gtrsim b^{BP_{dec}}$ and it remains to be studied the behaviour for other α values. In fact, higher degree graphs are still a big challenge to bipartitioning algorithms.

While the WL algorithm presented a longer runtime when compared with the BP algorithm with no decimation, when compared with the BP decimation approach it was actually faster. From this we conclude that the WL partitioning algorithm can be a good alternative to the BP partitioning, specially in bipartitioning finding, where the goal is to quickly determine the nodes that belong to each of the groups without compromising the quality of the solution. However, it must be notice that the WL results presented in this work for Erdos-Rényi random graphs were obtained considering only the giant component, thanks to theoretical considerations.

In this work was explored the performance of the WL and BP partitioning algorithms on bipartitioning problems, but it remains to further explore its performance on k-partitioning problems, where the graphs are divided into k groups of fixed size. One simple way of doing this would be to repeatedly apply the algorithms to the found groups until a total of k groups are found. However, there are reasons to believe that it would be possible to adapt the algorithms so the k groups could be found simultaneously.

References

[1] M. E. J. Newman, *Networks: an introduction*. Oxford; New York: Oxford University Press, 2010, ISBN: 9780199206650 0199206651.

[2] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. London: Prentice Hall, 1974.

[3] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," 2015. arXiv: 1311.3144 [cs.DS].

[4] P. Šulc and L. Zdeborová, "Belief propagation for graph partitioning," *Journal of Physics A: Mathematical and Theoretical*, vol. 43, no. 28, p. 285 003, Jun. 2010, ISSN: 1751-8121.

[5] "Undirected graphical models," 2014. [Online]. Available: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-438-algorithms-for-inference-fall-2014/lecture-notes/MIT6_438F14_Lec3.pdf.

[6] J. Pearl, "Bayesian networks: A model of self-activated memory for evidential reasoning," in *Proc. of Cognitive Science Society (CSS-7)*, 1985.

[7] J. Gu and A. Mohammed, "Mixed graph representation and mixed graph isomorphism," Nov. 2017.

[8] "Discrete mathematics for computer scientists mathematicians (2nd ed.)," *The Florida State University Department of Mathematics and Computer Science*, J. L. Mott, A. Kandel, and T. P. Baker, Eds., 1986.

[9] M. Mezard and A. Montanari, "Information, physics, and computation," *Foundations of Physics - FOUND PHYS*, vol. 26, pp. 291–320, Feb. 2009. DOI: 10.1093/acprof:oso/9780198570837.001.0001.

[10] J. Yedidia, W. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," in *Exploring Artificial Intelligence in the New Millennium*, G. Lakemeyer and B. Nebel, Eds., Morgan Kaufmann Publishers, Jan. 2003, ch. 8, pp. 239–236, ISBN: 1-55860-811-7. [Online]. Available: https://www.merl.com/publications/TR2001-22.

[11] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001. DOI: 10.1109/18.910572.

[12] M. Welling and Y. W. Teh, "Linear response algorithms for approximate inference in graphical models," *Neural Comput.*, vol. 16, no. 1, pp. 197–221, 2004. [Online]. Available: http://dblp.uni-trier.de/db/journals/neco/neco16.html#WellingT04.

[13] Wolfram. "Coding theory." (), [Online]. Available: https://mathworld.wolfram.com/CodingTheory.html.

[14] N. Aydin. "Introduction to coding theory and cryptography." (), [Online]. Available: https://www2.kenyon.edu/Depts/Math/Aydin/Teach/Sp05/392/Intro.pdf.

[15] R. Kindermann and L. Snell, *Markov random fields and their applications*. American Mathematical Society, 1980, vol. 1, ISBN: 978-0-8218-5001-5. DOI: http://dx.doi.org/10.1090/conm/001.

[16] S. G. Brush, "History of the lenz-ising model," *Rev. Mod. Phys.*, vol. 39, no. 4, pp. 883–893, Oct. 1967. DOI: 10.1103/RevModPhys.39.883.

[17] J. Taylor, "The ising model," [Online]. Available: https://math.la.asu.edu/~jtaylor/teaching/Spring2016/STP421/lectures/Ising.pdf.

[18] H. Zhang, G. Kamath, J. Kulkarni, and Z. S. Wu, "Privately learning markov random fields," 2020. arXiv: 2002.09463 [cs.DS].

[19] P. Eastman, "Introduction to statistical mechanics," *Reviews of Modern Physics*, [Online]. Available: https://web.stanford.edu/~peastman/statmech/index.html.

[20] A. Decelle and F. Krzakala, "Belief-propagation-guided monte-carlo sampling," *Physical Review B*, vol. 89, no. 21, Jun. 2014. DOI: 10.1103/physrevb.89.214421. [Online]. Available: https://doi.org/10.1103%2Fphysrevb.89.214421.

[21] N. Offeddu, M. Thielmann, and M. Ollikainen, "Computational statistical physics lecture notes," *ETH Zurich*, [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/baug/ifb/ifb-dam/homepage-IfB/Education/msc_courses/msc_computational-statphys/documents/dipse.pdf.

[22] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, pp. 183–233, Jan. 1999. DOI: 10.1023/A:1007665907178.

[23] F. Wang and D. P. Landau, "Efficient, multiple-range random walk algorithm to calculate the density of states," *Physical Review Letters*, vol. 86, no. 10, pp. 2050–2053, Mar. 2001, ISSN: 1079-7114. DOI: 10.1103/physrevlett.86.2050. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.86.2050.

[24] J. Lopes, "Métodos de monte carlo para sistemas com desordem e interacção de longo alcance," Phd Thesis, Universidade do Porto, Jun. 2006, pp. 73–76.

[25] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proceedings AAAI Natinal Conference on AI*, Pittsburgh, PA, 1982, pp. 133–136.

[26] A. Lipowski, A. L. Ferreira, D. Lipowska, and K. Gontarek, "Phase transitions in ising models on directed networks," *Phys. Rev. E*, vol. 92, p. 052 811, 5 Nov. 2015.

[27] S. Jo, J. Yoo, and U. Kang, "Fast and scalable distributed loopy belief propagation on real-world graphs," *WSDM '18: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 297–305, Feb. 2018. DOI: 10.1145/3159652.3159722.

[28] X. Li, Y. Pang, C. Zhao, Y. Liu, and Q. Dong, "A new multi-level algorithm for balanced partition problem on large scale directed graphs," *Advances in Aeronautics*, vol. 3, no. 1, 23, p. 23, Dec. 2021. DOI: 10.1186/s42774-021-00074-x.

[29] S. Boettcher and A. Percus, "Extremal optimization for graph partitioning," *CoRR*, vol. cond-mat/0104214, Jul. 2001. DOI: 10.1103/PhysRevE.64.026114.

[30] S. Boettcher, "Extremal optimization of graph partitioning at the percolation threshold," *Journal of Physics A: Mathematical and General*, vol. 32, no. 28, pp. 5201–5211, Jan. 1999. DOI: 10.1088/0305-4470/32/28/302.

[31] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes, "Critical phenomena in complex networks," *Reviews of Modern Physics*, vol. 80, no. 4, pp. 1275–1335, Oct. 2008. DOI: 10.1103/revmodphys.80.1275. [Online]. Available: https://doi.org/10.1103%2Frevmodphys.80.1275.

Appendix

Algorithm 1 Belief Propagation partitioning algorithm for a given graph and desired value of magnetization m^d.

Define the parameter $memory$, the maximum number of iterations $nIterMax$ and the minimum value of convergence eps,
Randomly initialize the external magnetic field H and the messages $\{h^0_{i\to j}\}$,
$nIter \leftarrow 0$, $conv \leftarrow 1.0$,
while $((nIter < nIterMax)$ and $(conv > eps))$ **do**
    $nIter \leftarrow nIter + 1$, $conv \leftarrow 0$,
    $\{h^1_{i\to j}\} \leftarrow 0$,
    for $i = 1$ *to* N **do**
        $convergence \leftarrow 0.0$,
        From the messages $h^0_{i\to j}$ compute h_i,
        for $j = \partial i$ **do**
            From $h^0_{i\to j}$ and the updated h_i compute the message $h^1_{i\to j}$,
            $convergence \leftarrow convergence + |h^1_{i\to j} - h^0_{i\to j}|$,
            $h^1_{i\to j} \leftarrow memory \times h^0_{i\to j} + (1 - memory) \times h^1_{i\to j}$,
        end for
        if $\alpha_i > 0$ **then**
            $convergence \leftarrow convergence/\alpha_i$,
        end if
        $conv \leftarrow conv + convergence$,
    end for
    $conv \leftarrow conv/N$,
    sort(h_i),
    $H \leftarrow -h_i[N(1 - m^d)/2]$,
    $\{h^0_{i\to j}\} \leftarrow \{h^1_{i\to j}\}$,
end while
Compute h_i,
$partition \leftarrow 0$
$n_+ \leftarrow 0$, $n_- \leftarrow 0$, $n_0 \leftarrow 0$,
for $i = 1$ *to* N **do**
    if $((H + h_i) > 0)$ **then**
        $partition[i] \leftarrow 1$,
        $n_+ \leftarrow n_+ + 1$
    else if $((H + h_i) < 0)$ **then**
        $partition[i] \leftarrow -1$,
        $n_- \leftarrow n_- + 1$
    else
        $n_0 \leftarrow n_0 + 1$
    end if
end for
$m \leftarrow (n_+ - n_-)/N$
Compute the partitioning cost b^{BP} from the spin values register in $partition$,
Compute the Bethe estimate of the ground-state energy E for the found configuration with magnetization m,
Compute the partitioning cost b^{BP}_E from the energy value computed,
    return $partition$, $b^{BP}(m^d)$ and $b^{BP}_E(m^d)$.

Algorithm 2 Wang-Landau partitioning algorithm for a given graph and desired value of magnetization m^d.

1: Define dm, $intm$ and n_{MC},
2: $n_f^{max} \leftarrow 0.0$
3: Compute the number of edges n_{edges},
4: $m_i \leftarrow m^d$,
5: Under the name *partition* registers the nodes of the initial random configuration with the n_+ positive spin values and the n_- spin values that agree with m_i.
6: Compute the energy E_i of the initial configuration.
7: $f \leftarrow 1.0$,
8: $w \leftarrow 0$,
9: $w(E_i, m^d) \leftarrow f$,
10: $E_{min} \leftarrow E_i$, $E_{max} \leftarrow E_i$, $E_0 \leftarrow E_{min}$,
11: $iff \leftarrow 1$,
12: Run the Wang-Landau algorithm using the function of Algorithm 3 and update the values *partition*, n_+, n_-, m_i, w, E_i, E_{min}, E_{max} and E_0,
13: $iff \leftarrow 0$,
14: Rerun the function of Algorithm 3 and update the values *partition*, n_+, n_-, m_i, w, E_i, E_{min}, E_{max} and E_0,
15: From the configuration *partition* compute the partition cost b^{WL}.
return *partition* and $b^{WL}(m^d)$.

Algorithm 3 Wang-Landau function

function WANGLANDAU($intm, dm, n_{MC}, n_{edges}, partition, n_+, n_-, m_i, n_f^{max}, f, w, E_i, E_{min}, E_{max}, E_0, iff$)
  $h \leftarrow 0$, $n_f \leftarrow 0$, $iflag \leftarrow 1$,
  for $step = 1$ *to* n_{MC} **do**
    for $is = 1$ *to* N **do**
      if ($step$ % $intm == 0$ *or* ($n_+ == 0$ *or* $n_+ == N$)) **then**
        Randomly choose a spin i to flip and compute E_r and m_r,
      else
        Randomly choose a pair of spins to interchange and compute E_r and m_r,
      end if
      if ($|m_r - m^d| \leq dm$ **then**
        Compute the probability $p_A(i \rightarrow r)$ of accepting the new configuration,
        Generate a random number rn between 0 and 1,
        if ($rn \leq p_A(i \rightarrow r)$) **then**
          Update n_+, n_-, E_{min} and E_{max} according to the proposed configuration,
          Reset the counter n_f to zero if the values of E_{min} or E_{max} had change,
          $m_i \leftarrow m_r$, $E_i \leftarrow E_r$
        end if
        if ($m_i == m^d$) *and* ($E_i \leq E_0$) **then**
          $E_0 \leftarrow E_i$
          The system configuration is saved under the name $partition$,
        end if
      end if
      if ($iff == 1$) **then**
        if ($f < 1.0^{-8}$) **then**
          Stop the function.
        else
          $w(E_i, m_i) \leftarrow w(E_i, m_i) + f$,
          if ($iflag == 1$ *and* $E_i == E_{max}$) **then**
            $n_f \leftarrow n_f + 1$, $iflag \leftarrow -1$,
          end if
          if ($iflag == -1$ *and* $energia == emin$) **then**
            $n_f \leftarrow n_f + 1$, $iflag \leftarrow 1$,
          end if
          if ($n_f == n_f^{max}$) **then**
            $f \leftarrow f/2.0$, $n_f \leftarrow 0$,
          end if
        end if
      end if
    end for
    $h(E_i, m_i) \leftarrow h(E_i, m_i) + 1$,
  end for
  $h(E_i, m_i) \leftarrow h(E_i, m_i)/n_{MC}$,
  return $partition$, n_+, n_-, m_i, w, E_i, E_{min}, E_{max} and E_0.