POLITECNICO
DI TORINO

Master degree course in Physics of Complex Systems

Master Degree Thesis

The undirected Feedback Vertex Set problem

Cavity method approach

Supervisor
prof. Alfredo Braunstein
.....

Candidate
Luca Manfrin
.....

Academic year 2017 - 2018

This work is subject to the Creative Commons Licence

Summary

Several real-world problems can be mapped into combinatorial optimization problems defined on an undirected network structure (or graph in the discrete mathematics language). Generally such an undirected graph $G = (V, E)$ may contain an abundant number of cycles, or loops.

A feedback vertex set (FVS) is a subset S of the graph vertices V intersecting with each of these loops such that $G^{'} = (V^{'} = V \setminus S, E^{'} = E \cap (V^{'} \otimes V^{'}))$ forms a collection of tree components. Creating a FVS of cardinality approaching the global minimum value is an optimization problem in the non-deterministic polynomial-complete (NP-c) complexity class therefore it is believed to be unsolvable by an exact algorithm in a time interval bounded by a polynomial function of the vertices number in the graph N.

In this thesis I develop, mathematically and algorithmically, an approach to the optimal FVS problem on undirected graphs with long loops. Leaded by the cavity method based model applied in [2] by Zhou, we build a new one calling it binary single cycle (BSC) model. The idea is to relax the local constraints, used in [2] to choose the set $S^{'} \subset V$, to obtain the new graph $G^{'} = (V^{'} = V \setminus S^{'}, E^{'} = E \cap (V^{'} \otimes V^{'}))$ as a collection of trees and connected components with a single loop, called c-trees.

With respect to [2], our method has several advantages. First, a simplified statistical mechanics model with binary variables instead of Potts-type variables is applied. Second, we implement a direct derivation of the belief propagation (BP) equations in the $\beta \to \infty$, or zero temperature, limit employing a reinforcement method: a sort of soft decimation procedure permitting faster algorithmic performances. Third, the BSC technique allows us to obtain quasi-ground states configurations from a single convergence of the iterative equations instead of using other procedures that would require several convergences.

The equations devised to treat this problem are based on a distributed message-passing procedure, called Max-Sum (MS). Since the energy minimum could present degeneracy,a perturbative external field is added to the

system to break the symmetry: this way the output of our algorithm is only one of the most probable configurations.

We find that our algorithm computational time, $O\left(N(1+d\left[1+\lambda_0^{-1})\right]\right)$, and its low memory cost, $O(Nd)$, permit better performances with respect to the belief propagation-guided decimation (BPD) method applied in [2], the annealing technique of [4], or the Min-Sum process devised in [10].

Having tested this heuristic algorithm on single instances of sparse Erdös-Rényi (ER) random graph, we come to the conclusion that it behaves well with a reinforcement factor $\lambda_0 = 10^{-4}$: this way we obtain results with an acceptable error margin in reasonably short times. We tested it on various mean degree values comparing our results with those in [10]. Furthermore we have done more detailed studies for ER graphs with a degree $d = 3.5$: our algorithm nearly reaches the theoretically estimated value $\theta = 0.1782$ of the FVS nodes fraction in the limit of large nodes number N or of low reinforcement factor λ_0.

Acknowledgements

I would like to thank Professor A. Braunstein and Dr. L. Dall'Asta for their fundamental assistance and collaboration throughout this thesis and also Professor I. Biazzo, A. Pagnani, A. A. Gamba and Dr. G. Catania for their practical support during the work and moral relief in the coffee breaks.

I also would like to thank G. M. Ferro for all the table tennis matches lost, A. Somazzi for the cured meat boards he shared, M. Lovisetto for the exquisite songs he performed for us, P. Pavanelli for the wisdom he bestowed upon us, and last, but not least, all of my colleagues for all the experiences shared in these two years.

Finally I have to endlessly thank Lucia Scarlet, light of my life, my family and, particularly, my mother without whom all this work would not have been possible.

Contents

List of Tables

List of Figures

Chapter 1

General introduction

1.1 Strutural control problems

In the last decades, the scientific community was confronted with the rise of interest in the complex systems field due to the increase in awareness that the physical world is described by irregular, complex, and dynamical structures: being able to control complex systems internal state has then become a central topic.

In a simplified way, networks can be used to describe and analyse a large variety of complex structures, from social sciences to information technology or biological systems. While the ultimate goal to study complex systems is to control them, the interplay between the dynamics and network structures presents a tremendous challenge to our ability to formulate effective control strategies. However, once we can represent the system as a graph, we need to study network theory to figure out which elements need to be managed, through which control actions, and how to force the system toward a desired stationary state.

A network $G = (V, E)$, also called graph in the discrete mathematics language, consists of a set V of N entities called nodes, or vertices: each of the related pairs of nodes in set E is referred to as edge. In the context of complex systems studies several crucial questions about control can be answered observing the property modifications of a graph G when a subset S of its nodes is selected and treated in a specific way.

In all these applications it is reasonable to assign an energetic cost to the inclusion of a vertex in S. This way one faces a combinatorial optimization problem: the cost minimization of subset S under the constraint of its effect on the graph. Imposing a null cost, this becomes a trivial issue since each

vertex in S is selected randomly with some independent probability, as in classical percolation theory. Otherwise, we are really confronting a combinatorial optimization problem exhibiting two types of features: static ones, due to the combinatorial optimization aspect, and dynamic ones, due to the dynamical definition of the cost function itself.

Among the control framework problems, structure-based methods, otherwise called topological control methods, suggest how to choose the control set S relying simply on the network structure and, if necessary, some general assumptions about the type of dynamics. A specific case of topological control is the feedback vertex set (FVS) problem which studies how to choose a set S in a graph G such that, once removed, the remaining graph $G^{'} = (V^{'} = V \setminus S, E^{'} = E \cap (V^{'} \otimes V^{'}))$ is acyclic [1].

For example the FVS problem has wide practical applications in the field of computer science such as database management, combinatorial circuit design, and deadlock recovery in operating systems [2, 3].

It also has many potential uses in complex networks dynamic analysis: the probability of belonging to a near-optimal FVS can be useful to discern how important is a node for the system. Furthermore, knowing the FVS of a graph highly simplifies the study of its dynamics since it can be considered as a straightforward response problem of a cycle-free subsystem under the influence of the vertices in the FVS [4]. In the case of direct graphs it has been shown that solving the FVS problem is equivalent to identify a minimum set of driver nodes to fully control a complex nonlinear network [5, 6].

Another real-world application concern the optimal targets of attack [7]: how to delete a minimum number of nodes to break the network down into disconnected small components. For Erdös-Rényi (ER) sparse random graph ensembles, this problem is essentially equivalent to the minimum FVS problem. The optimal targets of attack is especially applied in network structures protection, surveillance and control of network dynamical processes, such as the spread of infective diseases, and also play a significant role in network information diffusion, such as viral marketing and network advertisement.

1.2 Introduction to the Feedback Vertex Set (FVS) problem for an undirected graph

We consider the problem of finding a FVS on an undirected and simple graph $G = (V, E)$ composed of a set V of N vertices, whose integer-valued indices are generally denoted with lower-case letters (i, j, k, l ...), and of $|E|$ edges,

each of which connects two different vertices such that $E = \{(i, j) : i, j \in V\}$. The graph is undirected since the edges have no intrinsic orientation. Self-edges, relating a vertex to itself, are not present and there is at most one edge between any pair of different vertices therefore the graph is defined as simple.

If there is an edge between vertex i and j, then they are referred to as neighbours: ∂i is denoted as the set of nodes connected to vertex i and the degree $d_i \equiv |\partial i|$ is its cardinality. We define a path in graph G as the sequence of edges

$$(i, j_1), (j_1, j_2), \ldots, (j_{n-1}, j_n), (j_n, k)$$

joining vertex i and k. A cycle is a closed path wherein a vertex can be reached starting from itself. A connected sub-graph of graph G that contains no cycles is called a tree.

A feedback vertex set (FVS) of graph G is a subset $S \subset V$ such that, if all the vertices of this set and the attached edges are removed from G, the remaining graph $G^{'} = (V^{'} = V \setminus S, E^{'} = E \cap (V^{'} \otimes V^{'}))$ has no cycles and simply is a collection of tree components, also referred to as forest. This means that for each cycle of the graph G at least one of its vertices is included in the set S. A FVS for a directed graph is similarly defined: such a set should contain at least one vertex for every directed cycle of the graph. A FVS is also referred to as a decycling set hence the fraction of nodes in S is called decycling fraction $\theta \doteq |S|/N$.

Constructing a FVS for a given graph using percolation is a rather easy task, however, for many practical purposes, it is convenient to construct a FVS containing as few vertices as possible. This optimization problem of constructing a minimum FVS is extremely non-trivial since cycles of all sizes need to be considered. Indeed the minimum FVS problem is among the first 21 problems shown to be nondeterministic polynomial-complete (NP-c) by Cook and Karp in the early 1970s in [8, 9]. Therefore it is generally believed to be unsolvable by a deterministic sequential algorithm, for all generic input graphs G, in a computing time bounded by a polynomial function of the number N of its vertices.

1.3 State of the art for the undirected FVS problem on random graphs

Since it is not possible to devise an exact algorithm constructing an optimal FVS for large, cycle-rich graphs instances due to its NP-c nature, it is important to develop an efficient heuristic algorithm able to achieve near-optimal cardinality for the FVS problem.

Such research topic is quite challenging: the major difficulty is that cycles are global objects of a graph so their existence cannot be implied by simply checking the neighbourhood of a single vertex or edge. This theoretical difficulty was solved in various articles by building a local set of constraints for each node or edge and devising heuristic iterative algorithms to enforce them until convergence.

There are two main way to implement such an algorithm: the first is based on the cavity method, formulating approximate message passing equations conveying to every variable the information about cycles existence, and the second is a simulated annealing process, which iteratively looks for the system minimum energy gradually decreasing the temperature.

In [2] H. J. Zhou develops a spin glass approach where he defines a local variable A_i on each vertex i taking values $\{0, i, j \in \partial i\}$ therefore each node can assume $d_i + 2$ different possible states. If $A_i = 0$ the vertex is defined as unoccupied otherwise it is occupied and called root node if $A_i = i$ while, for $A_i = j\partial i$, i is defined as child node with j its parent node.

The author enforces local constraints on each node variable through the edge factor

$$\begin{aligned} C_{ij}(A_i, A_j) \doteq \delta^0_{A_i}\delta^0_{A_j} + \delta^0_{A_i}\left(1 - \delta^0_{A_j} - \delta^i_{A_j}\right) + \delta^0_{A_j}\left(1 - \delta^0_{A_i} - \delta^j_{A_i}\right) \\ + \delta^j_{A_i}\left(1 - \delta^0_{A_j} - \delta^i_{A_j}\right) + \delta^i_{A_j}\left(1 - \delta^0_{A_i} - \delta^j_{A_i}\right) \end{aligned} \tag{1.1}$$

containing all the permitted configurations of states between neighbours such that only if $C_{ij}(A_i, A_j) = 1$ the edge $(i, j) \in E$ is defined as satisfied.

A microscopic configuration of the whole graph is defined as $\underline{A} \equiv \{A_1, A_2, \ldots, A_N\}$ and, if $\underline{A}$ satisfies all the edges in G, than it is referred to as a solution of this graph. Every occupied node in such solution constitutes a collection of trees and c-trees, where a c-tree is a connected component with a single cycle. Therefore, imposing the unoccupied state to one random node for each c-tree cycle, the FVS is obtained including in S each node i with state $A_i = 0$.

Finally H. J. Zhou creates a set of belief propagation (BP) equations and implements the algorithm using a belief propagation-guided decimation

(BPD) heuristic procedure. In this routine he computes the BP message equations iteratively a number T of times at whose end the probability of each node to be unoccupied is calculated. Then the fN vertices with the highest empty-probability are deleted from G and added to the FVS. The author further simplifies the graph by iteratively removing, without including them in the FVS, all the nodes with degree $d_i \leq 1$. He repeats this procedure until the graph is empty hence obtaining the FVS of G. This procedure requires $O(N(d+2)Tf^{-1})$ number of iterations.

In [4] S. M. Qin and H. J. Zhou formulate a simulated annealing local search (SALS) algorithm for the undirected FVS problem by constructing an ordered list L formed by $n \leq N$ vertices

$$L \equiv (v_1, v_2, \ldots, v_n) \tag{1.2}$$

and assign a local variable to each of them in the form of an integer rank $r_i \in [1, n]$.

Then they impose as local constraints the ranking condition

$$\sum_{j:j\in L, j\in\partial i} \Theta(r_i - r_j) \leq 1 \quad \forall i \in L \tag{1.3}$$

with

$$\Theta(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \leq 0 \end{cases} \tag{1.4}$$

such that a list L is defined legal if it contains at most one neighbour of lower rank for each node $i \in L$. This way the FVS is formed by all the remaining vertices of G not included in L.

Then the SALS algorithm is implemented by initializing the legal set L with one randomly chosen vertex of G. At each step a arbitrarly selected node i is added to L with probability 1 if L contains less than two neighbours of i, or with probability $\exp(-\Delta E/T)$ otherwise: $E(L) \equiv N - \sum_{i\in L} 1$ is the set L energy. The process is repeated $N \times N_t$ times during which the minimum FVS and its energy E_{min} are recorded. Then the temperature is reduced as $T \leftarrow \alpha T$, with $\alpha < 1$, and the procedure is repeated until E_{min} is the same for N_{fail} contiguous temperature values. This algorithm requires an exact number $[N_t N \ln_\alpha(T_f/T_0)]$ of iterations, with T_0 the initial temperature and T_f the one where the convergence is reached.

Finally, in [10] A. Braunstein, L. Dall'Asta, G. Semerjian, and L. Zdeborová realize an algorithm based on leaf removal-time variables: these are

defined as the time-step at which a node is deleted upon reaching degree $d_i \leq 1$, due to the elimination of all of its neighbours except one.

These local constraints are rephrased as a set of static equations:

$$t_i(S) = \begin{cases} 0 \quad \text{if} \quad i \in S \\ \phi_i\left(\{t_j\}_{j\in\partial i}\right) \quad \text{if} \quad i \in V \setminus S \end{cases} \tag{1.5}$$

with

$$\phi_i\left(\{t_j\}_{j\in\partial i}\right) = 1 + \max_2\left(\{t_j(S)\}_{j\in\partial i}\right) \tag{1.6}$$

where $\max_2$ denote the second largest argument.

They then elaborate a message passing method based on Min-Sum equations and successively implement an iterative algorithm using a reinforcement procedure to allow its convergence: the computational time is $O(NdT\lambda_0^{-1})$ where T is the maximum value that the variable t_i can assume and λ_0 is the reinforcement factor.

1.4 The Binary Single Cycle (BSC) model

Motivated by the desire to find an heuristic algorithm better performing than those found in literature, in this section we show an original and simple way of representing the global cycles through local constraints on all the edges of an undirected graph $G = (V, E)$.

From [2] we borrow the idea of introducing a direction on each edge in G and, adopting this graphical representation, we build our binary single cycle (BSC) model where each edge is characterized by a local binary spin variable $x_{i\to j} \equiv x_{ij} = \{-1,1\}$. So we can define a set of spins $\mathbf{x} \doteq \{x_{ij} : (i,j) \in E\}$ where $x_{ij} = -1$ denotes an incoming edge in i from j while $x_{ij} = 1$ is an outgoing edge from i to j. By definition $x_{ij} = -x_{ji}$.

Since we want to solve the optimal FVS as an energy optimization problem, we need to decide how to assign a cost to every node that we add to the FVS. This purpose is reached by conceiving a set of soft constraints on each node such that every vertex having more than one child must be added to the FVS and pays the energetic price. Such soft constraints can be rephrased on each node as an equation of the edge-states $\mathbf{x}_i = \{x_{ij} : j \in \partial i\}$ linking the node i with its neighbours ∂i:

$$A(\mathbf{x}_i) = \sum_{j\in\partial i}\left(\delta^1_{x_{ij}} \prod_{k\in\partial i\setminus j} \delta^{-1}_{x_{ik}}\right) + \prod_{j\in\partial i} \delta^{-1}_{x_{ij}} \tag{1.7}$$

where δ_a^b is the Kronecker defined as

$$\delta_a^b = \begin{cases} 1 & \text{if} \quad a = b \\ 0 & \text{if} \quad a \neq b \end{cases} \tag{1.8}$$

The first term in (1.7)

$$\sum_{j \in \partial i} \left(\delta_{x_{ij}}^{1} \prod_{k \in \partial i \setminus j} \delta_{x_{ik}}^{-1} \right)$$

accounts for the node i to be a parent of one of its neighbours, as all but one edges have to be incoming, while the second

$$\prod_{j \in \partial i} \delta_{x_{ij}}^{-1}$$

is associated to the node being a root, as all the edges have to be incoming. This way if $A(\mathbf{x}_i) = 1$ the node i abides by its local soft constraints and is graphically represented by a full circle. Otherwise, $A(\mathbf{x}_i) = 0$ represents a node which, unable to fit its constraints, is added to the FVS $S^{'}$ and is drawn as an empty circle.

So we call $\mathbf{x}$ a solution for G: for a single graph instance we have $2^{|E|}$ possible different solutions $\{\mathbf{x}\}$, where $|E| = Nd$ and $d = \langle d_i \rangle_{i \in V}$.

However, the FVS is still potentially incomplete: there could be some cycle in the remaining graph $G^{'} = (V^{'} = V \setminus S^{'}, E^{'} = E \cap (V^{'} \otimes V^{'}))$. As in [2], $G^{'}$ can be shown to be a collection of trees and c-trees, which are connected components with one loop. Then we still need to break the c-trees cycle by randomly deleting one of the loop vertices and adding it to the FVS to obtain a complete FVS S.

It is easy to show how disposing the orientation on the edges and deleting the nodes with more than one outgoing direction, produces at best c-tree components. We can consider the simple and generalizable case of a connected component formed by two single cycles joined by an edge as in Figure 1.2a. Firstly we consider only one loop, Figure 1.1a, and, imposing the direction on one edge, the soft constraints bring us to assign values to the remaining edge-variables of the cycle obtaining a directed one, as in Figure 1.1b.

Considering now the whole connected component as in Figure 1.2a, imposing a direction on the left loop the edge linking the two cycles must be directed toward the left loop. This way we have two possibilities: either

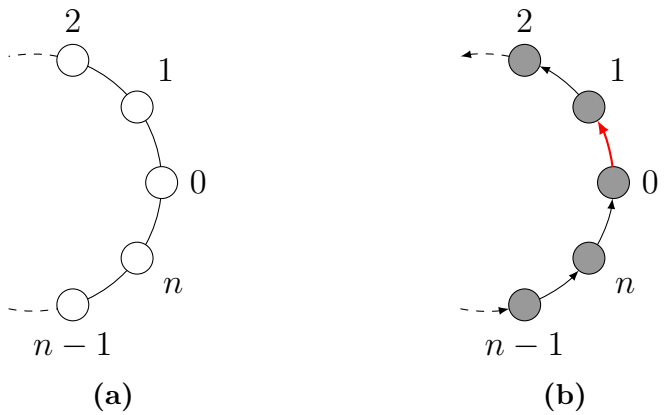


Figure 1.1: Representation of a generic undirected single cycle: (a) is the cycle before imposing a direction on its edges while in (b), having imposed $x_{01} = 1$, the other edge variables are consequence of the soft constraints.

$n+1$ has two outgoing edges, as in Figure 1.2b, or one of the other nodes of the right cycle will, as in Figure 1.2c. This is the reason why a connected component in $G^{'} = (V^{'} = V \setminus S^{'}, E^{'} = E \cap (V^{'} \otimes V^{'}))$ cannot contain more than one loop.

Our BSC model has two main characteristics: it contains only binary local variables and uses soft constraints. The first leads to a low memory cost and a fast algorithm since, for a sparse graph with mean degree d, the number of variables considered is only $O(Nd)$. Furthermore, the second permits an easy implementation of the equations since we don't need to exclude any possible edge-variables configuration.

Now we can define the energy of a configuration $\mathbf{x}$ as

$$E(\mathbf{x}) = \sum_{i=1}^{N} [1 - A(\mathbf{x}_i)] \tag{1.9}$$

using $A(\mathbf{x}_i)$ defined in (1.7). This way the partition function of our BSC model is

$$Z(\beta) = \sum_{\mathbf{x}} e^{-\beta E(\mathbf{x})} \tag{1.10}$$

where the sum over $\mathbf{x} = \{X_{ij} = x_{ij}\}_{(i,j)\in E}$ is a sum over all the possible configurations and β is the inverse temperature. The joint probability mass function is

$$p(\mathbf{x}) = \frac{1}{Z(\beta)} e^{-\beta E(\mathbf{x})} \tag{1.11}$$

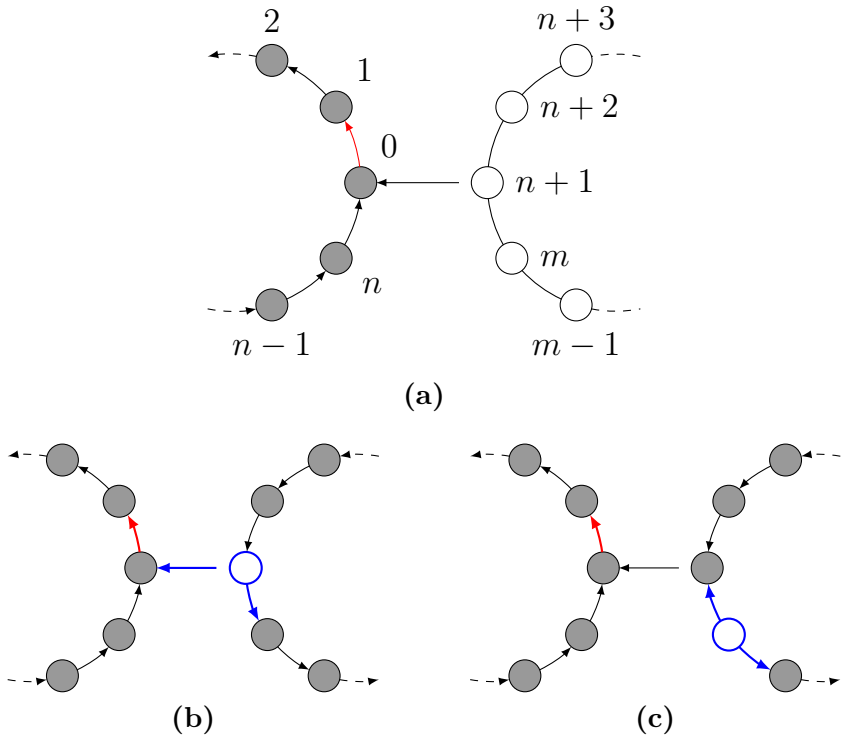


Figure 1.2: Representation of a connected component composed of two cycles linked by an edge. In all figures the red edge represents the one imposed first: in (a) we enforce the soft constraints on 0 by setting $x_{0,n+1} = -1$ consequently, in (b) and (c), we show the two possible ways of imposing the constraints on the remaining edges: the empty node is the one unable to satisfy them.

and the free energy is related to the partition function through

$$F(\beta) = \frac{1}{\beta} \ln Z(\beta) \tag{1.12}$$

The exact computations of the partition function in equation (1.10) for an arbitrary graph remains an NP-hard problem. However, if G is a sparse random graph, we can compute it in the thermodynamic limit using the cavity method.

Chapter 2

Cavity method implementation

2.1 BSC model Belief Propagation (BP) algorithm on factor graphs

Cavity methods [11, 12], also known as message-passing algorithms, are a practical and powerful way to solve problems involving probabilistic inference using graphical models. Two different objectives can be distinguished when applying the cavity method: the first is to find marginal probabilities for some subset of the system nodes, implementing the belief propagation (BP) technique, while the second is to find one of the system most probable global states, otherwise said optimal, using another procedure called Max-Sum (MS) or Min-Sum equivalently, simply depending of the sign definitions.

Since the MS view, which is the one we are interested in, corresponds to the BP approach imposing an infinite energetic cost, or zero temperature, we start presenting the BP equations devised for our BSC problem.

In the BP algorithm [13, 14, 12], and generally in all cavity methods, messages are sent from one node to each of its neighbours and vice versa. The equations we obtain from this messages are exact when the graphical model is a tree otherwise the presence of cycles cause BP to give approximate results. Since BP is a local procedure it should perform well whenever the underlying graph is 'locally' a tree but, even in this case, the algorithm could not converge or, in presence of long-range correlations, could perform poorly.

We introduce the BP routine on graphical models defined in terms of factor graphs: a bipartite graph expressing the factorization structure in the joint

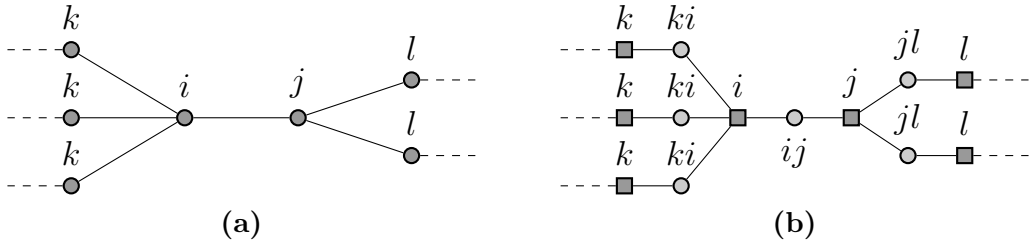


Figure 2.1: This figure illustrates the equivalence between a generic graph (a) and its corresponding factor graph (b).

probability mass function

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{i \in V} \psi_i(\mathbf{x}_i) \tag{2.1}$$

in terms of a variable node set and of a factor node set. In our BSC model, the factor nodes are the original graph nodes, on whom the soft constraints act, while the variable nodes are the edges, as is visible in Figure 2.1. Indeed, in equation (2.1) i is an index labelling the node factors $\psi_1, \psi_2, ...\psi_N$, graphically represented with a circle, and $\mathbf{x}_i = \{x_{ij} : j \in \partial i\}$ includes the edge-variable nodes in the neighbourhood of i, graphically represented with a square as in Figure 2.1b. We assume that the functions $\psi_i(\mathbf{x}_i)$ are non-negative and finite such that $p(\mathbf{x})$ is a well-defined probability distribution, with Z its generic normalization constant.

To focus on the problem of computing the edge-variable nodes marginal

$$p_{(ij)}(x_{ij}) = \sum_{\mathbf{x} \setminus x_{ij}} p(\mathbf{x}) \tag{2.2}$$

we need to introduce the messages between variable nodes and their neighbouring factor nodes and vice versa. Our BSC model has the advantage that every edge-variable node is always connected only to two factor nodes: the message from an edge-variable node to one of its two factor neighbours $n_{ij \to j}(x_{ij})$ is then simply the message coming from the other factor node $m_{i \to ij}(x_{ij})$, so

$$n_{ij \to j}(x_{ij}) \propto m_{i \to ij}(x_{ij}) \tag{2.3}$$

This way the message equation going from a factor node to one of its neighbouring edge-variable nodes $m_{i \to ij}(x_{ij})$ can be easily written as

$$m_{i \to ij}(x_{ij}) \propto \sum_{\mathbf{x}_i \setminus x_{ij}} \left[\psi_i(\mathbf{x}_i) \prod_{ki \in \partial i \setminus ij} m_{k \to ki}(x_{ki}) \right] \tag{2.4}$$

where $\mathbf{x}_i = \{x_{ij} : j \in \partial i\}$ and $\sum_{\mathbf{x}_i \setminus x_{ij}}$ is the sum over all the possible configurations of the $\mathbf{x}_i \setminus x_{ij}$ edge-variables. The factor graph notation becomes useful later on, in section 2.1.2, when we insert an external field to break the system symmetry. Anyway, the notation can be simplified by considering only the variable nodes labels: this way the message equation becomes

$$m_{ij}(x_{ij}) = \frac{1}{z_{ij}} \sum_{\mathbf{x}_i \setminus x_{ij}} \left[\psi_i(\mathbf{x}_i) \prod_{k \in \partial i \setminus j} m_{ki}(x_{ki}) \right] \tag{2.5}$$

with the normalization factor z_{ij} simply defined as

$$z_{ij} = m_{ij}(-1) + m_{ij}(1) \tag{2.6}$$

This BP message (2.5) is also called 'sum-product' due to its form.

Finally we build the believes $b_{(ij)}(x_{ij})$, which are the BP approximation to the exact marginal probabilities function $p_{(ij)}(x_{ij})$. The belief equations are defined as

$$b_{(i,j)}(x_{ij}) \propto m_{ij}(x_{ij}) m_{ji}(x_{ji} = -x_{ij}) \tag{2.7}$$

where the proportionality symbol $\propto$ point out that one must still normalize this quantity. So the BP algorithm is created by iterating the message-update equations (2.5) until they converge, or until they meet other requirements, after which the beliefs can be read off from (2.7). Thus, (2.5) and (2.7) characterise our BP algorithm.

To compute explicitly the message update equations we need to define the constraint function on the factor nodes. This has to determine when a node has to pay the energetic price for not abiding to the soft constraints, hence having the form

$$\begin{aligned} \psi_i(\mathbf{x}_i) &\doteq A(\mathbf{x}_i) + e^{-\mu} \left(1 - A(\mathbf{x}_i)\right) \\ &= e^{-\mu} + (1 - e^{-\mu}) \left[\sum_{j \in \partial i} \left(\delta^1_{x_{ij}} \prod_{k \in \partial i \setminus j} \delta^{-1}_{x_{ik}} \right) + \prod_{j \in \partial i} \delta^{-1}_{x_{ij}} \right] \end{aligned} \tag{2.8}$$

where we used the definition of $A(\mathbf{x}_i)$ in (1.7). With this equation we see that the joint probability functions in equation (2.1) and (1.11) coincide for identical configurations $\mathbf{x}$, hence

$$\psi_i(\mathbf{x}_i) \equiv e^{-\beta E_i(\mathbf{x}_i)} \tag{2.9}$$

If we look at this equation with $A(\mathbf{x}_i) = 0$ we obtain the equality $\exp(-\mu) = \exp(-\beta)$ so the cost we are imposing has the physical meaning of an inverse temperature.

Replacing $\psi_i(\mathbf{x}_i)$ with its explicit form we can rewrite (2.5) as the sum of three parts:

$$\text{(i)}\ e^{-\mu} \prod_{k\in\partial i\setminus j} \left[\sum_{x_{ki}} m_{ki}(x_{ki})\right] = e^{-\mu} \prod_{k\in\partial i\setminus j} [m_{ki}(-1) + m_{ki}(1)]$$

$$\begin{aligned}\text{(ii)}\ & (1-e^{-\mu}) \sum_{\mathbf{x}_i\setminus x_{ij}} \left[\sum_{l\in\partial i} \left(\delta^1_{x_{il}} \prod_{k\in\partial i\setminus l} \delta^{-1}_{x_{ik}} \prod_{k\in\partial i\setminus j} m_{ki}(x_{ki})\right)\right] \\ & = (1-e^{-\mu}) \left\{\delta^1_{x_{ij}} \left[\prod_{k\in\partial i\setminus j} m_{ki}(1)\right] + \delta^{-1}_{x_{ij}} \left[\sum_{k\in\partial i\setminus j} \frac{m_{ki}(-1)}{m_{ki}(1)}\right] \left[\prod_{k\in\partial i\setminus j} m_{ki}(1)\right]\right\}\end{aligned}$$

$$\text{(iii)}\ (1-e^{-\mu}) \sum_{\mathbf{x}_i\setminus x_{ij}} \left[\prod_{l\in\partial i} \delta^{-1}_{x_{il}} \prod_{k\in\partial i\setminus j} m_{ki}(x_{ki})\right] = (1-e^{-\mu}) \delta^{-1}_{x_{ij}} \left[\prod_{k\in\partial i\setminus j} m_{ki}(1)\right]$$

where in (i) we used

$$\sum_{\mathbf{x}_i\setminus x_{ij}} \left[\prod_{k\in\partial i\setminus j} m_{ki}(x_{ki})\right] = \prod_{k\in\partial i\setminus j} \left[\sum_{x_{ki}} m_{ki}(x_{ki})\right]$$

Then we can compute the two message update equations for $x_{ij} = \{-1,1\}$:

$$\begin{cases} m_{ij}(-1) \propto e^{-\mu} \prod_{k\in\partial i\setminus j} \left[1 + \frac{m_{ki}(-1)}{m_{ki}(1)}\right] + (1-e^{-\mu}) \left[\sum_{l\in\partial i\setminus j} \frac{m_{li}(-1)}{m_{li}(1)} + 1\right] \\ m_{ij}(1) \propto e^{-\mu} \prod_{k\in\partial i\setminus j} \left[1 + \frac{m_{ki}(-1)}{m_{ki}(1)}\right] + (1-e^{-\mu}) \end{cases} \tag{2.10}$$

We use the proportionality symbol $\propto$ to emphasize that a factor

$$\frac{1}{z_{ij}} \prod_{k\in\partial i\setminus j} m_{ki}(1)$$

has been neglected in both the above.

2.1.1 BP implementation equations for BSC

Due to the BSC model having binary variables, we can efficiently gather the two message equations in a single variable one containing all the information

about the edge-variable state x_{ij}. So we define the new variable

$$\begin{aligned} q_{ij} &\doteq \frac{m_{ij}(-1)}{m_{ij}(1)} \\ &= 1 + \frac{(1-e^{-\mu}) \sum_{l \in \partial i \setminus j} q_{li}}{e^{-\mu} \prod_{k \in \partial i \setminus j} (1+q_{ki}) + (1-e^{-\mu})} \end{aligned} \tag{2.11}$$

The $=$ here is due to the deletion of the prefactor

$$\frac{1}{z_{ij}} \prod_{k \in \partial i \setminus j} m_{ki}(1)$$

in the fraction.

Finally, using the message normalization

$$m_{ij}(-1) + m_{ij}(1) = 1$$

we can re-write m_{ij} as function of q_{ij}:

$$\begin{cases} m_{ij}(-1) = \dfrac{q_{ij}}{1+q_{ij}} \\ m_{ij}(1) = \dfrac{1}{1+q_{ij}} \end{cases} \tag{2.12}$$

2.1.2 Local perturbative external field

The BP implementation has a symmetry problem: when dealing with multiple cycles the algorithm could be unable to converge or lead to a trivial configuration where all the believes are 0.5.

This problem can be solved by breaking the symmetry with a local perturbative external field $\epsilon_{(ij)}(x_{ij})$: this way we always obtain only one of possible configurations. To include this field we add an ulterior factor node connected to each edge-variable in the factor graph, as in Figure 2.2b.

This brings back the distinction of two types of messages in the factor graph: the factor to edge-variable message

$$n_{ij \to j}(x_{ij}) \propto \epsilon_{(ij)}(x_{ij}) m_{i \to ij}(x_{ij}) \tag{2.13}$$

and its inverse

$$m_{i \to ij}(x_{ij}) \propto \sum_{\mathbf{x}_i \setminus x_{ij}} \left[\psi_i(\mathbf{x}_i) \prod_{ki \in \partial i \setminus ij} n_{ki \to i}(x_{ki}) \right] \tag{2.14}$$

(a) **(b)**

Figure 2.2: In this figure we can see the factor graph before (a) and after (b) the introduction of the perturbative external field $\epsilon_{(ij)}(x_{ij})$.

To simplify again the notation, we call the edge-variable to factor message $m_{ij}^{post}(x_{ij})$, since it is the one after the field-factor, such that

$$m_{ij}^{post}(x_{ij}) \equiv \epsilon_{(ij)}(x_{ij}) \sum_{\mathbf{x}_i \setminus x_{ij}} \left[\psi_i(\mathbf{x}_i) \prod_{k \in \partial i \setminus j} m_{ki}^{post}(x_{ki}) \right] \tag{2.15}$$

This way, using a new definition of the field-factor

$$\epsilon_{(ij)}(x_{ij}) = e^{\mu x_{ij} \hat{\epsilon}_{ij}/2}, \quad \text{s.t.} \quad \hat{\epsilon}_{ij} = -\hat{\epsilon}_{ji} \tag{2.16}$$

that will be handy in the next part, the single variable q_{ij} can be easily re-written as function of $m_{ij}^{post}(x_{ij})$ as

$$q_{ij} = e^{-\mu \hat{\epsilon}_{ij}} \left[1 + \frac{\sum_{l \in \partial i \setminus j} q_{li}}{\frac{e^{-\mu}}{(1-e^{-\mu})} \prod_{k \in \partial i \setminus j} (1 + q_{ki}) + 1} \right] \tag{2.17}$$

Finally the believes can then be exactly computed as:

$$\begin{aligned} b_{(ij)}(x_{ij}) &= \frac{m_{ij}^{post}(x_{ij})\ m_{ji}^{post}(-x_{ij})\ e^{-\mu x_{ij} \hat{\epsilon}_{ij}/2}}{m_{ij}^{post}(x_{ij}) m_{ji}^{post}(-x_{ij}) e^{-\mu x_{ij} \hat{\epsilon}_{ij}/2} + m_{ij}^{post}(-x_{ij}) m_{ji}^{post}(x_{ij}) e^{\mu x_{ij} \hat{\epsilon}_{ij}/2}} \\ &= \left[1 + \left(\frac{q_{ij}}{\hat{q}_{ji}} \right)^{x_{ij}} e^{\mu x_{ij} \epsilon_{ij}} \right]^{-1} \end{aligned} \tag{2.18}$$

2.2 BSC model Max-Sum (MS) algorithm on factor graphs

Message-passing algorithms are not limited to computing marginals: as anticipated, we could want to find a configuration maximizing a given joint

probability distribution $p(\mathbf{x})$. This task is carried out by the MS algorithm [12].

In this new context the role of marginal probabilities is played by the max-marginals

$$M_{ij}(x_{ij}^*) = \max_{\mathbf{x}} \{p(\mathbf{x}) : x_{ij} = x_{ij}^* \forall (i,j) \in E\} \tag{2.19}$$

When the max-marginals are non-degenerate, for each $(i,j) \in E$, there exists an x_{ij}^* such that $M_{ij}(x_{ij}^*) > M_{ij}(x_{ij})$ for any $x_{ij} \neq x_{ij}^*$. The unique maximizing configuration is then given by $\mathbf{x}^* = \{x_{ij}^*\}_{(ij)\in E}$.

The message-passing update rules leading to the computation of max-marginals can be straightforwardly deduced from the BP ones: it is sufficient to replace sums with maximizations. This yields the following max-product update rules:

$$n_{ij\to j}(x_{ij}) \propto e^{\mu x_{ij}\hat{\epsilon}_{ij}/2} m_{i\to ij}(x_{ij}) \tag{2.20}$$

and

$$m_{i\to ij}(x_{ij}) \propto \max_{\mathbf{x}_i \setminus x_{ij}} \left\{ \psi_i(\mathbf{x}_i) \prod_{ki \in \partial i \setminus ij} n_{ki\to i}(x_{ki}) \right\} \tag{2.21}$$

The max-product messages admit a similar interpretation as in the case of BP: $n_{ij\to j}(x_{ij})$ is an estimate of the max-marginal of variable x_{ij} with respect to the modified graphical model where factor node j is removed from the graph and, analogously $m_{i\to ij}(x_{ij})$ is the max-marginal when all factors in $\partial ij \setminus i$ have been removed. Hence also this method belongs to the 'cavity' family.

To use a simplified notation we can again rewrite the message $n_{ij\to j}(x_{ij})$ as

$$m_{ij}^{post} \equiv e^{\mu x_{ij}\hat{\epsilon}_{ij}/2} \max_{\mathbf{x}_i \setminus x_{ij}} \left\{ \psi_i(\mathbf{x}_i) \prod_{k \in \partial i \setminus j} m_{ki}^{post}(x_{ki}) \right\} \tag{2.22}$$

An estimate of the max-marginals is obtained as in the BP form:

$$b_{(ij)}(x_{ij}) \propto m_{ij}^{post}(x_{ij}) m_{ji}^{post}(x_{ji}) e^{-\mu x_{ij}\hat{\epsilon}_{ij}/2} \tag{2.23}$$

To find the optimal configuration of a distribution factorizing as equation (2.1), we develop an alternative formulation based on minimizing the energy cost function. To do so we impose a change of variable for the message function such that

$$m_{ij}^{post}(x_{ij}) \doteq e^{\mu h_{ij}(x_{ij})} \tag{2.24}$$

and

$$\psi_i(\mathbf{x}_i) \doteq e^{\mu H_i(\mathbf{x}_i)} \tag{2.25}$$

Since in statistical mechanics we have $p(x_{ij}) = \exp^{-\beta E_{ij}(x_{ij})}$ and we saw with (2.8) that $\mu \equiv \beta$, the new variables above defined are energy opposite. Furthermore we can develop the new set of messages

$$\hat{h}_{ij}(x_{ij}) \propto \frac{\hat{\epsilon}_{ij}}{2} + \max_{\mathbf{x}_i \setminus x_{ij}} \left\{ H_i(\mathbf{x}_i) + \sum_{bk \in \partial i \setminus j} \hat{h}_{ki}(x_{ki}) \right\} \tag{2.26}$$

exploiting the exponential function properties. Due to its form, this equation is called Max-Sum (MS).

2.3 MS version of BP equations

Here we implement the MS messages starting from the BP ones obtained in section 2.1.2. From (2.24) we obtain

$$h_{ij}(x_{ij}) \propto \frac{x_{ij}\hat{\epsilon}_{ij}}{2} + \frac{1}{\mu} \ln \left[m^*_{ij}(x_{ij}) \right] \tag{2.27}$$

Now we can re-write the message equations (2.10) as function of the new variable $h_{ij}(x_{ij})$:

$$\begin{cases} h_{ij}(-1) \propto -\dfrac{\hat{\epsilon}_{ij}}{2} + \dfrac{1}{\mu} \ln \left\{ e^{-\mu} \displaystyle\prod_{k \in \partial i \setminus j} \left(1 + e^{\mu [h_{ki}(-1) - h_{ki}(1)]} \right) + \right. \\ \qquad \left. + (1 - e^{-\mu}) \left[\displaystyle\sum_{l \in \partial i \setminus j} e^{\mu [h_{li}(-1) - h_{li}(1)]} + 1 \right] \right\} \\ h_{ij}(1) \propto \dfrac{\hat{\epsilon}_{ij}}{2} + \dfrac{1}{\mu} \ln \left\{ e^{-\mu} \displaystyle\prod_{k \in \partial i \setminus j} \left(1 + e^{\mu [h_{ki}(-1) - h_{ki}(1)]} \right) + (1 - e^{-\mu}) \right\} \end{cases} \tag{2.28}$$

To obtain the same form of MS message in (2.26) above, we can now apply the the saddle-point method

$$\lim_{\mu \to \infty} \frac{1}{\mu} \ln \left(\sum_{i=1}^{n} e^{\mu f_i(x_i)} \right) = \max \left\{ f_i(x_i) \right\}_{i=1}^{n}$$

to (2.28) in the limit $\mu \to \infty$, obtaining

$$\begin{cases} h_{ij}(-1) \propto -\dfrac{\hat{\epsilon}_{ij}}{2} + \max\left\{ \displaystyle\sum_{k\in\partial i\setminus j} \max\left\{0; h_{ki}(-1) - h_{ki}(1)\right\} - 1; \right. \\ \qquad\qquad \left. \displaystyle\max_{l\in\partial i\setminus j} \left\{0; h_{li}(-1) - h_{li}(1)\right\} \right\} \\ h_{ij}(1) \propto \dfrac{\hat{\epsilon}_{ij}}{2} + \max\left\{ \displaystyle\sum_{k\in\partial i\setminus j} \max\left\{0; h_{ki}(-1) - h_{ki}(1)\right\} - 1; 0 \right\} \end{cases} \tag{2.29}$$

2.3.1 MS implementation equations for BSC

To implement the optimized algorithm using a single variable equation, we define

$$\hat{h}_{ij} \doteq h_{ij}(-1) - h_{ij}(1) \tag{2.30}$$

and with further computations we obtain:

$$\begin{aligned} \hat{h}_{ij} = &- \hat{\epsilon}_{ij} + \max\left\{ \sum_{k\in\partial i\setminus j} \max\left\{0; \hat{h}_{ki}\right\} - 1; \max_{l\in\partial i\setminus j} \left\{\hat{h}_{li}\right\}; 0 \right\} - \\ &- \max\left\{ \sum_{k\in\partial i\setminus j} \max\left\{0; \hat{h}_{ki}\right\} - 1; 0 \right\} \end{aligned} \tag{2.31}$$

Here, as already stated for (2.17), the $=$ sign is due to a simplification of the additive prefactors of $h_{ij}(x_{ij})$ in (2.30).

Due to the above equation form, is easy to show that

$$\hat{h}_{ij} + \hat{\epsilon}_{ij} \geq 0 \tag{2.32}$$

meaning $\hat{h}_{ij} \in [-\hat{\epsilon}_{ij}, +\infty)$.

Considering the case where $\partial i \setminus j$ is an empty set, we define

$$\max\{\emptyset\} = -\infty$$

hence, for a leaf in the graph, $\hat{h}_{ij} = -\hat{\epsilon}_{ij}$.

Finally we can derive the belief of the edge (i, j), like in (2.18), as function of $\hat{h}_{ij}$:

$$b_{(ij)}(x_{ij}) = \frac{1}{1 + e^{\mu x_{ij}\left(\hat{h}_{ij} - \hat{h}_{ji} + \hat{\epsilon}_{ij}\right)}} \sim \begin{cases} 0, & \text{if } \hat{b}_{ij}x_{ij} > 0 \\ 1, & \text{if } \hat{b}_{ij}x_{ij} < 0 \end{cases} \tag{2.33}$$

where we define

$$\hat{b}_{ij} \doteq \hat{h}_{ij} - \hat{h}_{ji} + \hat{\epsilon}_{ij} \tag{2.34}$$

This new variable contains all the informations about the state of the edge (i, j), and $\hat{b}_{ij} = -\hat{b}_{ji}$.

In (2.33) above, the solutions represent max-marginals and not probabilities hence $b_{(ij)}(x_{ij}) \sim 1/2$ is not feasible since, breaking the symmetry, we are forcing the system in one of its energy minima.

2.4 BSC model algorithmic implementation with MS

We implement the MS algorithm for our BSC model as a Python 3.6 function in the following way:

(i) we input the graph G as an adjacency list and build a matrix $\hat{\mathbf{h}}$ for the messages and another one for the external field, with the same form. The former is initialized to zero while the latter is randomly filled such that $\hat{\epsilon}_{ij} = -\hat{\epsilon}_{ji}$, with values in a range $[-g/|E|, g/|E|]$ where $0 < g < 1$ is an input variable of the function;

(ii) in a *while* loop we iterate the MS equation (2.31), using a reinforcement method until convergence. as in [10];

(iii) a new graph is then created deleting the nodes having more than one child and including them in the FVS S, obtaining a collection of tree and c-tree components. Finally, in this new graph, for each c-tree we add to S a vertex of the cycle detected using the depth first search (DFS) method.

The complete code is reported in Appendix A.

The iterative procedure in (ii) is designed with a parallel update method meaning that, when a new $\hat{h}_{ij}$ value is computed, it is immediately replaced inside the matrix $\hat{\mathbf{h}}$ and the edge-variable sequence we follow is randomized to prevent periodic oscillations of the message values. This iteration is implemented in an optimized way with two nested *for* loops allowing to compute the new message values in a number of steps linear with N. Finally, the reinforcement technique we apply to accelerate the algorithmic convergence, is based on the same principles described in the SI Appendix of [10].

2.4.1 The reinforcement method

In the BP algorithm a reinforcement, forcing the messages toward a solution, can be applied since the equations converge to the same stationary states. For the MS algorithm, on the other hand, this reinforcement method has only been validated through empirical results.

The concept is very similar to the learning coefficient used in machine learning: each time-step we modify the perturbative external field on each edge using the value assumed by the same edge-variable during the previous time-step. This way we can force the system toward one of its near-optimal states faster. How quickly the *whyle* loop converges is imposed using a time-step dependent variable λ_t, that we call reinforcement variable: the faster it is the less accurate the result will be.

We implement the reinforcement on $\hat{\epsilon}_{ij}$ as

$$\hat{\epsilon}_{ij}^{t+1} \doteq \hat{\epsilon}_{ij}^{t} - \lambda_t \hat{b}_{ij}^{t} \tag{2.35}$$

where λ_t is considered growing linearly with the time-step t as $\lambda_t \doteq \lambda_0\, t$, and $\hat{b}_{ij}$, defined in (2.34). The $-$ sign in the above equation is due to the form of $\hat{h}_{ij}$ in (2.31): in fact, replacing (2.35) in (2.31), in the limit $t \to \infty$ leads to:

$$\hat{h}_{ij}^{t+1} \simeq \lambda_0\; t\; \hat{b}_{ij}^{t} \tag{2.36}$$

From this equation we can also understand that, as the algorithm proceed, the edge-variable diverges. For this reason we have to impose, as convergence condition, that the sign of $\hat{h}_{ij}$ for each edge $(i,j) \in E$ does not change for a certain number of consecutive iterations. From (2.33) this condition results equivalent to require that the direction on each edge remains invariant for a certain number of consecutive time-steps.

Both, the reinforcement factor λ_0 and the number of consecutive iterations needed to reach convergence, are variables that have to be specified at the beginning when using this BSC functional implementation.

2.4.2 Computational time

After explaining how the algorithm works, we can now estimate its computational time.

In the *while* loop in (ii) we have two nested for-loops: the outer one is on all the graph nodes while the inner one in on all of its neighbours so, given a graph with mean degree d, the computational time is $O(Nd)$.

Due to the reinforcement method we expect the whole MS algorithm to converge in $O(\lambda_0^{-1})$ time-steps as worst case: the reinforced equations admit a solution $< +\infty$ only if the reinforcement $\lambda_t = \lambda_0\ t \leq 1$.

Finally the cost of searching and deleting one node of every remaining single cycle in (iii) is $O(N + |E|)$.

Using $|E| = Nd$, the algorithm total computational time is then

$$O\left(N\left[1 + d(\lambda_0^{-1} + 1)\right]\right) \tag{2.37}$$

Compared to the implementations developed in [2], [4], and [10], our gain a factor $O(T\lambda_0/f)$, $O(N_t\lambda_0 \ln_\alpha(T_f/T_0)/d)$, and $O(T)$ respectively if tested on the same ER random graphs.

Chapter 3

Results

We try our algorithm on Erdös-Renyi sparse random graphs. For sparse random networks it is well known that short loops of length $L \ll \log N$ are very rare [15, 16, 17]. In such networks the small connected components are mostly trees, while each giant component includes a finite fraction of all the nodes and an exponential number of long loops. As already argued in section 2.1, we expect that our BP approximation is sufficiently correct in the thermodynamic limit thanks to the ER random graph instances being locally trees.

In our experiments we always use a number of consecutive equal configurations fixed to 10, and $g = 10^{-4}$. Both variables aim to bound the message values from diverging too fast, when a high number of iterations is reached, without affecting the speed of convergence.

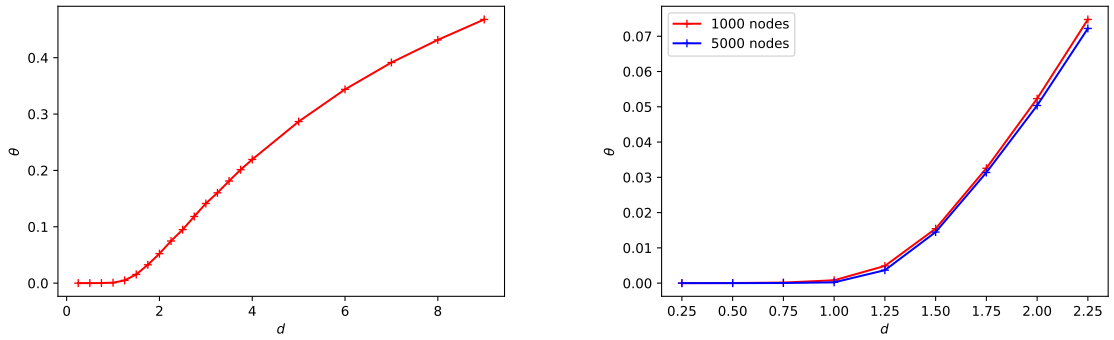
3.1 Phase transition in the mean degree d

First of all we confronted the decycling fraction

$$\theta \doteq \frac{|S|}{N}$$

at various mean degree values. This comparison should allows us to verify the correctness of the algorithm we implemented since a trivial second-order phase transition at $d = 1$ is expected: if $d = Np < 1$ the graph will almost surely have no connected components of cardinality greater than $O(\log N)$. As we are treating ER random graphs, the cycles length grows as $O(\log N)$ so, for $d < 1$, on average the connected components does not contain cycles.

Indeed this is the result shown in Figure 3.1.



(a) **(b)**

Figure 3.1: The figure represents how θ behaves with the degree change. In (a) we can observe a large d range for $N = 1000$ while in (b) there is the particular around $d = 1$ for $N = 1000$n and $N = 5000$. To obtain these data we use a reinforcement factor $\lambda_0 = 10^{-4}$.

3.2 Algorithmic behaviour changing the reinforcement factor λ_0

We now consider the behaviour of our algorithm for different reinforcement factor values λ_0. This study has the objective of understanding if the decycling fraction converges to the theoretic results reported in Table (3.1), obtained in [10].

d	$\theta^{\mathbf{MS}}_{\mathbf{decycling}}(\mathbf{d})$
1.5	0.0135
2.5	0.0936
3.5	0.1782
5	0.2823

Table 3.1: The (1RSB) cavity predictions for the decycling number $\theta^{MS}_{decycling}(d)$ of Erdös-Renyi random graphs of average degree d: the values have been reached by the MS algorithm on graphs of size $N = 10^7$ nodes [10].

We generally expect the θ values to be more precise with $N \to \infty$, since our approximation is done in the thermodynamic limit, and with $\lambda_0 \to 0$, as the reinforcement is a forcing factor on the convergence time which could prevent the algorithm from reaching the global energy minimum.

A short test is done on ER graphs with all the mean degree values $d \doteq \langle d_i \rangle$ in Table 3.1: we fit them with

$$f(\lambda_0) = a + \log_{10}(\lambda_0^b + c) \quad \text{with} \quad a, b, c > 0$$

and the results are presented in Figure 3.2. Their asymptotic values are computed with $f(\lambda_0) \sim a \log_{10} c$ as $\lambda_0 \to 0$ and all of them approach the corresponding theoretic limit in Table 3.1: from the numerical results we obtained $\theta \sim 0.01302$ for $d = 1.5$, $\theta \sim 0.09305$ for $d = 1.5$, $\theta \sim 0.17834$ for $d = 1.5$, and $\theta \sim 0.28343$ for $d = 5.0$. As expected the fraction becomes more precise when $\lambda_0 \to 0$.

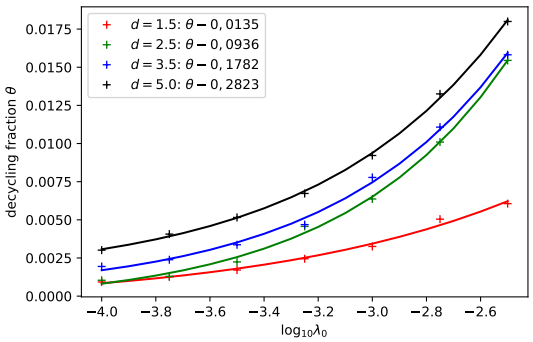


Figure 3.2: In this figure we report the behaviour of the total decycling fraction θ against $\log_1 0\lambda_0$ for all the mean degree values in Table 3.1.

To do an extensive study on this behaviour, we fix the ER graphs degree at $d = 3.5$ and we work on networks with $N = 1000$ and 5000. As we previously anticipated, for the same λ_0 the θ values at $N = 5000$ are more precise then those at $N = 1000$ like we see in Figure 3.3.

The fitted data are used it to plot Figure 3.4a: this exhibits the percentage distance of θ from its asymptotic value. Since the percentage distance at $\lambda_0 = 10^{-4}$ is under the 1% and the number of iterations required to reach convergence are reasonably small, as is visible in Figure 3.4b, we decide to fix $\lambda_0 = 10^{-4}$ for the following applications

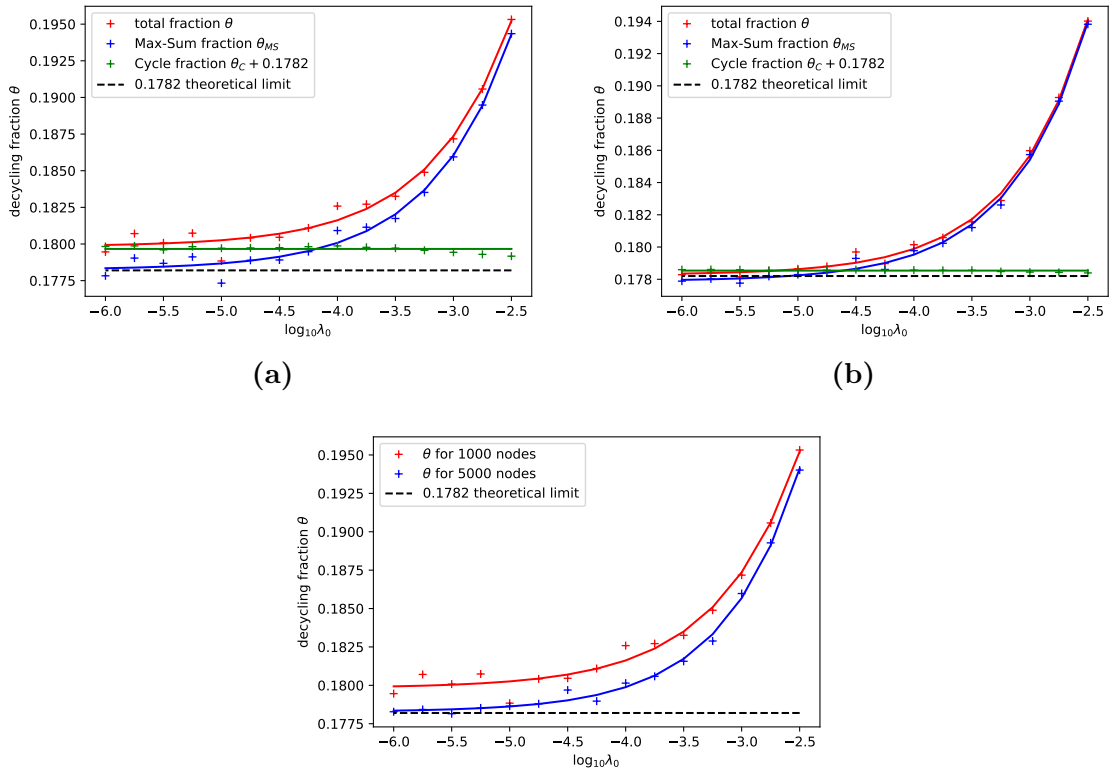
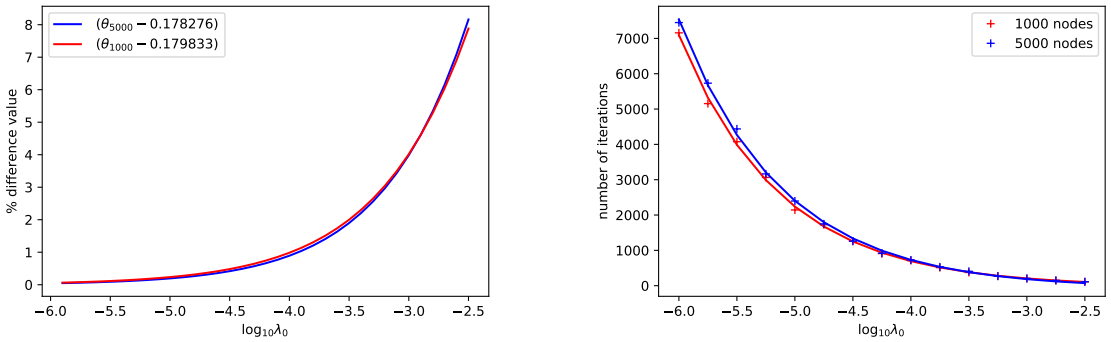


Figure 3.3: Algorithmic results on the decycling fraction θ against the logarithmic scale reinforcement factor λ_0 of Erdös-Renyi random graphs with degree $d = 3.5$. Both figures (a) and (b) represent the node fraction deletion through the MS algorithm (θ_{MS} blue line), c-trees single cycles (θ_C green line), and their sum (θ red line) with respect to the theoretical value 0.1782: (a) corresponds to the variation for a $N = 1000$ graph while (b) is a $N = 5000$ nodes. Finally (c) represent the comparison between the total fraction θ for the two cases of $N = 1000$ and $N = 5000$.

3.3 Algorithmic behaviour changing the number of nodes N

In this section we keep fixed the reinforcement factor to $\lambda_0 = 10^{-4}$ and the algorithm is analysed for an increasing number of nodes N. We again use ER random graphs with a degree $d = 3.5$ and $N = 3^c$, with the c values reported in Figure 3.5.



(a) **(b)**

Figure 3.4: In (a) the figure shows the percentage distance of the total fraction θ from its fitted asymptotic value, for $N = 1000$ and $N = 5000$, with respect to $\log_1 0\lambda_0$. In (b) we represent, on the same abscissa values, the mean iterations number needed to the algorithm to reach convergence.

As expected, θ approaches the theoretic value as N increases: in both images of Figure 3.5 this is easily visible. We fitted the data using

$$f(N) = \frac{a}{N^b} + c$$

with $a, b, c > 0$.

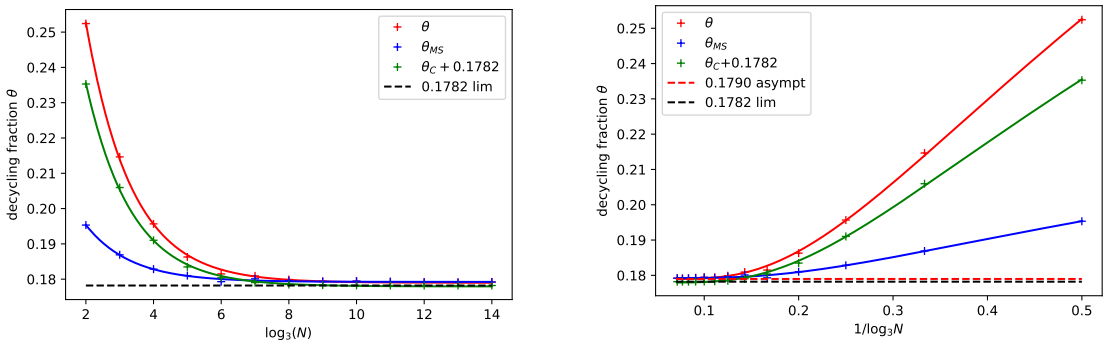
From this set of data we also see how the fraction of nodes deleted in the c-tree components $\theta_C \to 0$ as N increases. This has to be expected since the cycles length is $l_C \geq \log N$ and, with a number n_C of c-trees:

$$n_C \log N \leq n_C l_C \leq N \Longrightarrow \frac{n_C}{N} \leq \frac{1}{\log N} \to 0 \quad \text{as} \quad N \to \infty \tag{3.1}$$

Finally, as is reported in Figure 3.6, we can confirm that the computing time increases linearly with N.

3.4 Confront with the exact algorithm

At last the decycling fraction results of the BSC algorithm are compared with those obtained running the exact algorithm in Appendix B. Since the latter is exponentially slow with N we have only used graphs with a small number of nodes.



(a) **(b)**

Figure 3.5: Both figures represent the behaviour of the decycling fractions compared to N. In (a) on the abscissa we use the $\log_3 N$ while in (b) $1/\log_3 N$. The red line represents the total decycling fraction, the blue one the node fraction deleted using MS, and the green one the fraction erased from the c-tree components.

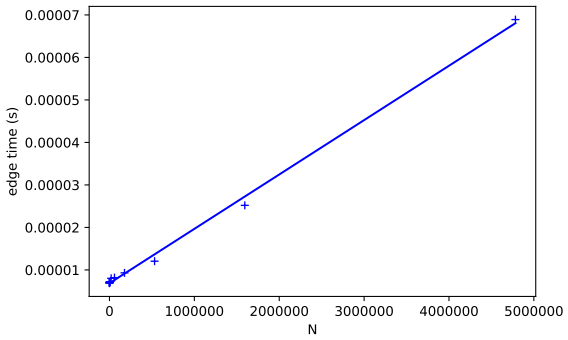
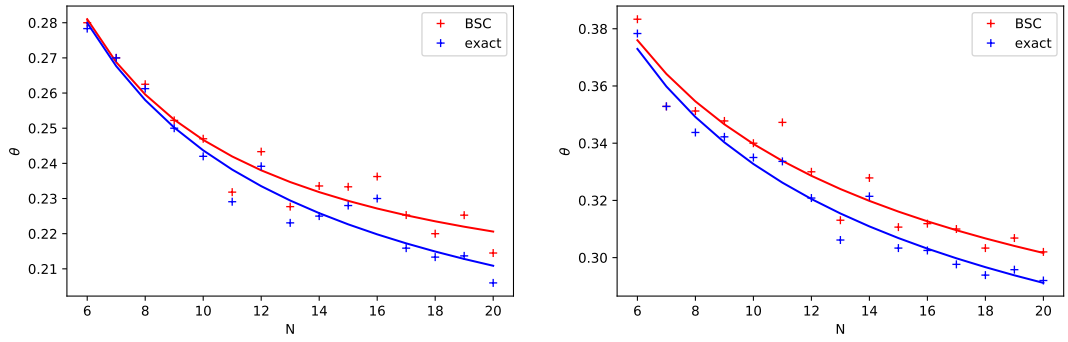


Figure 3.6: In figure is shown the computational time growth, expressed in seconds, for a single edge in the graph with respect to N.

In Figure 3.7 the results for $d = 3.4$ and $d = 4.5$ are reported: for both of them we can observe how the two data sets diverge as N increases. This is the inverse of our expectations, since our algorithm should be right in the thermodynamic limit but we believe that this behaviour is due to the N values being too small in this study to constitute a sufficient statistics.



(a) **(b)**

Figure 3.7: In figure is shown the mean cardinality growth of FVS S, with respect to N, of our BSG model (red) against the exact algorithm (blue) for a mean degree $d = 3.5$ in (a) and $d = 4.5$ in (b).

Chapter 4

Conclusions

In this thesis, we constructed a binary single cycle (BSC) spin glass model to solve the optimal feedback vertex set (FVS) problem on undirected graphs. The model was implemented using Max-Sum (MS) equations to explore the zero temperature limit of this problem, and a reinforcement method was applied to the algorithm to accelerate its convergence. This implementation was tested on single instances of sparse Erdös-Renyi (ER) random graphs. Our numerical results of the decycling number θ show that the MS implementation of our BSC model is able to construct nearly optimal feedback vertex sets for large N values and low reinforcement factor λ_0.

The data also show how the algorithmic computational time increases linearly with the number N of the graph nodes: this confirms that our BSC model has a better performance than the works in literature [2, 4, 10]. Although these results meet our predictions, the numerical time related statistics obtained for different reinforcement values λ_0 were affected by the parallel computational simulations we operated and did not reach our expectations. To solve this problem we suggest to improve the algorithmic time computation by not using absolute time references as calculator-clock related functions.

To realize broader statistics on our BSC implementation, in the future it would be appropriate to further develop the tests done on different degrees of ER random graphs and try it on real world networks, where our approximation could result more crude.

Having other available time it would also been interesting to compare our data with those obtained from the algorithms in literature [2, 4, 10].

Furthermore the FVS problem on direct graph has been shown to be important [5, 6]. We treated it with the spin glass model presented in [18],

implementing the equations using MS method. This algorithm was tested on single instances of ER random digraphs build such that the mean incoming and outgoing degree were the same. Fitting the data, we found that for a mean outgoing (or incoming) degree $d_{out} = 3.5$ the total decycling fraction approaches the same theoretic value $\theta_{th} = 0.1782$ of the ER undirected graphs with total mean degree $d = 3.5$. Even if this behaviour has not been yet understood from the theoretical point of view, if the results will be confirmed in future works, our BSC model on undirected graphs could be used to efficiently extrapolate predictions on the decycling number of this particular case of ER random digraphs.

Appendices

Appendix A

Below we present the complete BSG algorithmic implementation. The code has been developed in Python 3.6 as a function taking the inputs explained in Section 2.4: the undirected graph G as an adjacency list (referred to as `graph`), the factor f (referred to as `factor`), the reinforcement factor λ_0 (referred to as `lambdA_0`), and the number of consecutive equal configurations (referred to as `max_stable_config`). The function gives in output: the number of iterations, the total number of nodes in FVS, the nodes obtained through MS iteration, the nodes due to the decycling of c-trees, the time (in seconds) to compute one edge-variable, the set of nodes obtained through MS, and the set of nodes due to decycling of the c-trees.

Implementation of BSG model with MS equations on Python 3.6

```
# Defining the Max-Sum as a function
def MS_HJun_alg(graph, factor, alfa, lambdA_0, max_stable_config):
    start_time=time.time() # computational time for each edge

    #    INITIALIZE MESSAGE VECTOR
    initial_value=0.
    #initialize h_hat
    h_hat=[[y*0.+initial_value for y in x] for x in graph]

    #    INITIALIZE CORRESPONDANCE VECTOR
    # q(i->j) and q(j->i) corresponding positions
    correspondance=[list() for _ in range(len(graph))]
    for i in range(len(graph)):
        for j in range(len(graph[i])):
            for k in range(len(graph[graph[i][j]])):
                if i==graph[graph[i][j]][k]:
                    correspondance[i].append(k)
                    break
```

```
            # we can interrupt it since we consider simple graphs:..
            # ..no self loops and no multi-edges

#   INITIALIZE THE EXTERNAL FIELD VECTOR
# Count the number of edges
edges=0
for i in range(len(graph)):
    for j in range(len(graph[i])):
        edges+=1
# since using the adjacent matrix we're duoble counting every edge
edges=int(edges/2.)
# external field range
epsilon=1.*factor/edges
# Assign a rand value of field in [-epsilon,epsilon]..
# ..to each edge s.t. epsilon(ij)=-epsilon(ji)
field=[[0. for j in range(len(graph[i]))]for i in range(len(graph))]
for i in range(len(graph)):
    for j in range(len(graph[i])):
        if i<graph[i][j]:
            field[i][j]=(np.random.rand()*2-1)*epsilon
            field[graph[i][j]][correspondance[i][j]]=-field[i][j]

# REINFORCED MS
loop=0 # iterations
# number of consecutive cycles the..
# ..configiration remained the same
stable_config=0
time_law=lambda t: t # defining how lambdA_0 changes with time
while stable_config<max_stable_config:
    #if loop%1.e3==0. and loop!=0: # we can print the loops
    #   print loop
    loop+=1
    flag=1 # flag for the stable config.
    # shuffling the order of edges
    gr_len=np.arange(len(graph))
    np.random.shuffle(gr_len)
    for i in gr_len:
        # We can distinguish the LEAF case from the others
        if len(graph[i])==1:
            sign=np.sign(h_hat[i][0]
                         -h_hat[graph[i][0]][correspondance[i][0]]
                         +field[i][0])
            h_hat[i][0]=alfa*h_hat[i][0]-(1.-alfa)*field[i][0]
            if sign!=np.sign(h_hat[i][0]
                         -h_hat[graph[i][0]][correspondance[i][0]]
                         +field[i][0]):
```

```
            flag=0
        else:
            max1_h=max2_h=float('-inf')
            sum_max=0.
            for j in range(len(graph[i])):
                # look for max h_hat in the neighborhood
                if (max2_h<h_hat[graph[i][j]][correspondance[i][j]]):
                    if (max1_h<=(h_hat[graph[i][j]]
                            [correspondance[i][j]])):
                        max2_h,max1_h=(max1_h,
                            h_hat[graph[i][j]][correspondance[i][j]]))
                    else:
                        max2_h=(h_hat[graph[i][j]]
                            [correspondance[i][j]])
            # we create the sum of max of all neighbours and we'll..
            # ..subtract the max of the neigh. we'll be looking at
                sum_max+=max(0.,(h_hat[graph[i][j]]
                     correspondance[i][j]]))
                gr_sublen=np.arange(len(graph[i]))
                np.random.shuffle(gr_sublen)
                for j in gr_sublen:
                    sign=np.sign(h_hat[i][j]-(h_hat[graph[i][j]]
                            [correspondance[i][j]])+field[i][j])
                    h_hat[i][j]=(alfa*h_hat[i][j]
                           + (1.-alfa)*(-field[i][j]+max(sum_max
                           -max(0.,(h_hat[graph[i][j]]
                               [correspondance[i][j]])-1.,
                              max1_h if max1_h!=(h_hat[graph[i][j]]
                               [correspondance[i][j]])else max2_h,0.)
                           -max(sum_max-max(0.,(h_hat[graph[i][j]]
                               [correspondance[i][j]])-1.,0.)
                           ))
                    if sign!=np.sign(h_hat[i][j]-
                            h_hat[graph[i][j]][correspondance[i][j]]
                            +field[i][j]):
                        flag=0
        for i in range(len(graph)): # MS reinforcement implementation
            for j in range(len(graph[i])):
                if graph[i][j]>i:
                    field[i][j]-=(lambdA_0*time_law(loop)*(h_hat[i][j]
                            -h_hat[graph[i][j]][correspondance[i][j]]
                            +field[i][j]))
                    field[graph[i][j]][correspondance[i][j]]=-field[i][j]
        if flag==1:
            stable_config+=1
        else:
```

```
            stable_config=0

#   CREATE THE LIST OF BELIEVES & DELETE NODES
# Create list of believes b(+1) from i to j
b={}
for i in range(len(graph)):
    for j in graph[i]:
        if (((i,j) not in b) and ((j,i) not in b)):
            b[i,j]=0.
# Update of the believes list b(+1)
out_edge=[0]*len(graph) # counts how many edges exit a node
for i in range(len(graph)):
    for j in range(len(graph[i])):
        if (i,graph[i][j]) in b:
            if (h_hat[i][j]-h_hat[graph[i][j]][correspondance[i][j]]
                  +field[i][j]>0):
                b[i,graph[i][j]]=0.
                out_edge[graph[i][j]]+=1
                elif (h_hat[i][j]
                      -h_hat[graph[i][j]][correspondance[i][j]]
                      +field[i][j]<0):
                    b[i,graph[i][j]]=1.
                    out_edge[i]+=1

# print which nodes have >1 outgoing edge
deleted_nodes_MS=[]
for i in range(len(graph)):
    if out_edge[i]>1:
        deleted_nodes_MS.append(i)

#   DELETE NODES AND STUDY DISCONNECTED SUBGRAPHS
# Delete MS nodes
new_graph=[[]for _ in graph]
for i in range(len(graph)):
    if i not in deleted_nodes_MS:
        for j in range(len(graph[i])):
            if graph[i][j] not in deleted_nodes_MS:
                new_graph[i].append(graph[i][j])

# Depth First Search
def dfs_visit(graph, node, found_cycle, prev_node, marked,
    deleted_nodes_Cycle):
        if found_cycle[0]==1:
            return
        marked[node]=1
        for neigh in graph[node]:
```

```
            if marked[neigh]==1 and neigh!=prev_node:
                if found_cycle[0]==0:
                    deleted_nodes_Cycle.append(node)
                found_cycle[0]=1
                return
            elif marked[neigh]==0:
                dfs_visit(graph,neigh,found_cycle,node,marked,
                          deleted_nodes_Cycle)

    # Find a SIMPLE cycle in a CONNECTED component
    def cycle_finder(graph,marked,deleted_nodes_Cycle):
        found_cycle=[0]
            for node in range(len(graph)):
                if marked[node]==0:
                    dfs_visit(graph,node,found_cycle,node,marked,
                              deleted_nodes_Cycle)
                if found_cycle[0]==1:
                    break
        return marked,deleted_nodes_Cycle

    # Find how many simple cycles we have in the subgraphs
    connected_components=0
    marked=[0 for _ in range(len(new_graph))]
    deleted_nodes_Cycle=[]
    for i in range(len(new_graph)):
        if marked[i]==0:
            connected_components+=1
            marked,deleted_nodes_Cycle=cycle_finder(new_graph,marked,
                    deleted_nodes_Cycle)
            marked=[x*2 for x in marked]
            # this way we won't find cycles from unmarked nodes..
            # ..of connected components only partially marked

    return (loop,len(deleted_nodes_MS)+len(deleted_nodes_Cycle),
             len(deleted_nodes_MS),len(deleted_nodes_Cycle),
             (time.time()-start_time)/(2.*edges*loop),
             deleted_nodes_MS, deleted_nodes_Cycle )
```

Appendix B

Here is presented the algorithm code to compute exactly the FVS problem. This code has been developed in Python 3.6 as a function taking as input only the undirected graph G (referred to as `graph`) and giving as output a list of lists: each of the inner lists is a possible FVS of G.

Implementation of exact FVS solution on Python 3.6

```
def long_decycling(graph):
    N=len(graph)
    found_tot=0 # flag for finding an fvs
    stop=[0]
    fvs=[]
    counter=[-1]*N
    counter[0]+=1
    j=1
    while stop[0]==0:
        nodes=[]
        for _ in range(j):
            nodes.append(counter[_])
            new_graph=nodes_del(graph,nodes)
            found=cycle_detector(new_graph)
            if found==0:
                found_tot=1
                fvs.append(nodes)
        # to look for all fvs of the same cardinality
        if found_tot==1 and counter[j-1]==N-j:
            break
        counter[0]+=1
        if counter[0]>=N: # create new potential fvs
            counter_growth(counter,1,N,stop)
            counter[0]=counter[1]+1
            if -1 in counter:
                j=counter.index(-1)
            else:
                j=len(counter)
    return fvs

def nodes_del(graph,nodes):
    new_graph=[[] for _ in graph]
    for i in range(len(graph)):
        if i not in nodes:
            for j in range(len(graph[i])):
                if graph[i][j] not in nodes:
                    new_graph[i].append(graph[i][j])
    return new_graph

# we look for cycles in a (potentially disconnected) graph
def cycle_detector(graph):
    found_cycle=[0]
    marked=[0 for _ in range(len(graph))]
    for node in range(len(graph)):
```

```
        if marked[node]==0:
            dfs_visit(graph,node,found_cycle,node,marked)
        if found_cycle[0]==1:
            break
    return found_cycle[0]

def dfs_visit(graph, node, found_cycle, prev_node, marked):
    if found_cycle[0]==1:
        return
    marked[node]=1
    for neigh in graph[node]:
        if marked[neigh]==1 and neigh!=prev_node:
            found_cycle[0]=1
            return
        elif marked[neigh]==0:
            dfs_visit(graph,neigh,found_cycle,node,marked)

def counter_growth(counter,i,N,stop):
    counter[i]+=1
    if counter[i]>=N-i and i<N-1:
        counter_growth(counter,i+1,N,stop)
        counter[i]=counter[i+1]+1
    if counter[i]>=N-i and i==N-1:
        stop[0]=1
    return counter,stop
```

Bibliography

[1] J. G. T. Zañudo, G. Yang, and R. Albert, *Structre-based control of complex networks with nonlinear dynamics*, PNAS **114**, 28 (2017)

[2] H. J. Zhou, *Spin glass approach to the feedback vertex set problem*, Eur. Phys. J. B **86**, 455 (2013)

[3] S. M. Qin, Y. Zeng, and H. J. Zhou, *Spin-glass phase transitions and minimum energy of the random feedback vertex set problem*, Phys. Rev. **94**, 022146 (2016)

[4] S. M. Qin, H. J. Zhou, *Solving the undirected feedback vertex set problem by local search*, Eur. Phys. J. B **87**, 273 (2014)

[5] B. Fiedler, A. Mochizuki, G. Kurosawa, D. Saito, *Dynamics and control at feedback vertex sets. I: Informative and determining nodes in regulatory networks*, J. Dyn. Diff, Equat. **25**: 563-604 (2013)

[6] A. Mochizuki, B. Fiedler, G. Kurosawa, D. Saito, *Dynamics and control at feedback vertex sets. II: A faithful monitor to determine the diversity of molecular activities in regulatory networks*, J. Th. Bio. **335**: 130-146 (2013)

[7] S.Mugishan and H. J. Zhou, *Identifying optimal targets of network attack by belief propagation*, Phys. Rev. **94**, 012305 (2016)

[8] S.A. Cook, The complexity of theorem-proving procedures, in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, (New York, ACM, 1971), pp. 151–158

[9] R.M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*, (New York Plenum Press, 1972), pp. 85–103

[10] A. Braunstein, L. Dall'Asta, G. Semerjian, and L. Zdeborová, *Network dismantling*, PNAS **113**, 44 (2016)

[11] M. Mezard, G. Parisi, *The Bethe lattice spin glass revisited*, Eur. Phys. J. B **20**: 217-233 (2001)

[12] M. Mézard, A. Montanari, *Information, Physics and Computation*, (London, Oxord Univ Press, 2009)

[13] J. S. Yedidia, W. T. Freeman, and Y.Weiss ,*Constructing Free-Energy Approximations and Generalizing Belief Propagation Algorithms*, IEEE Trans. Inf. Th. **51**, 7 (2005)

[14] J. S. Yedidia, W. T. Freeman, and Y.Weiss *understanding Belief Propagation and its Generalizations*, NIPS-13 (2000)

[15] E. Marinari and R. Monasson, *Circuits in random graphs: from local trees to global loops*, J. Stat. Mech. (2004) P09004.

[16] E. Marinari, R. Monasson, and G. Semerjian, *An algorithm for counting circuits: Application to real-world and random graphs*, Europhys. Lett. **73**, 8 (2005).

[17] G. Bianconi and M. Marsili, *Loops of any size and hamilton cycles in random scale-free networks*, J. Stat. Mech. (2005) P06005.

[18] H. J. Zhou, *A spin glass approach to the directed feedback vertex set problem*, J. Stat. Mech., 073303 (2016)