Inference of Hyperparameters in Agent-based Dynamics

With applications to network inference in epidemics

Politecnico di Torino

Master degree in
Physics of Complex Systems

Candidate:
Federico Florio

Supervisor:
Prof. Alfredo Braunstein

Co-Supervisor:
Stefano Crotti

Academic Year 2023/2024

ABSTRACT

Dynamic processes on graphs are fundamental to modeling a wide range of real-world phenomena, including the spread of epidemics, information diffusion in social networks and neural cascades. In such cases, the parameters governing these processes are often unknown, and for an accurate description of the processes they must be deduced from observations that are often incomplete or even erroneous. This thesis addresses the challenge of inferring the governing parameters of discrete-time Markov processes on graphs using observations of the time series. A primary focus is placed on the notoriously difficult task of inferring the network topology itself from partial observations.

In the past, inference methods have been proposed for non-recurrent models, i.e. those in which the system cannot return to a previous state, which are simpler to address due to the small number of possible single-node trajectories. When considering recurrent models, this number grows exponentially with the time horizon, making their treatment substantially harder. This thesis introduces an innovative inference technique tailored for recurrent models, with a specific emphasis on epidemic spreading models.

Central to this approach are two key methodologies: Belief Propagation (BP), an algorithm that approximates the posterior probability distributions of the trajectories conditioned to the observations, and the Tensor Train approximation, which enables efficient representation and manipulation of multi-variable functions, essential for network inference.

The method was tested on the Susceptible-Infectious-Susceptible (SIS) model, showing successful reconstruction of the underlying network from limited data. Consequently, this thesis opens up new avenues for applications in diverse fields such as epidemiology, social dynamics, and computational neuroscience. The results represent a substantial advancement in the field of network inference for dynamic stochastic processes, offering a robust toolset for understanding complex systems governed by recurrent interactions.

Table of Contents

List of Abbreviations

AUC	Area Under the Curve. 29, 49
BP	Belief Propagation. 4, 6, 7, 10–12, 15, 18, 20, 33, 37, 39, 43, 44
EM	Expectation-Maximization. 3, 20
FNR	False Negative Rate. 49
FPR	False Positive Rate. 48, 49
GA	Gradient Ascent. 4, 8, 9, 20, 21
MPBP	Matrix-Product Belief Propagation. 4, 10, 17, 18, 20–22, 28, 33
MPS	Matrix-Product State. 12, 13, 16
ROC	Receiver Operating Characteristic. 28, 29, 31, 48, 49
SI	Susceptible-Infectious. 4, 5
SIS	Susceptible-Infectious-Susceptible. vi, 2, 4–6, 8, 10, 14, 19, 20, 25, 27, 30, 33–35, 42
SVD	Singular-Value Decomposition. 13, 14, 16, 17, 41, 42
TNR	True Negative Rate. 48
TPR	True Positive Rate. 48, 49

Mathematical Notation

$\mathbb{I}[a]$: indicator function for proposition a ($\mathbb{I}[a] = 1 \iff a$ is true and $\mathbb{I}[a] = 0 \iff a$ is false)

$\delta_{x,y}$: Kronecker's delta function ($\delta_{x,y} = \mathbb{I}[x = y]$)

$\otimes$: tensor product

$A \setminus a$: set A deprived of element a (slight abuse of notation for $A \setminus \{a\}$)

Graphs

$\mathscr{G} = (\mathscr{V}, \mathscr{E})$: graph $\mathscr{G}$ with set of nodes $\mathscr{V}$ and set of edges $\mathscr{E}$

$i \in \mathscr{G}$: node i belongs to graph $\mathscr{G}$ (slight abuse of notation for $i \in \mathscr{V}$)

$(ij) \in \mathscr{G}$: edge (ij) belongs to graph $\mathscr{G}$ (slight abuse of notation for $(ij) \in \mathscr{E}$). Edges are considered as undirected unless otherwise specified

∂i: neighbors of node i ($\partial i = \{a \in \mathscr{V} : (ai) \in \mathscr{E}\}$)

$-i$: set of all nodes except i ($-i = \mathscr{V} \setminus \{i\}$)

Probability

$\mathbb{P}[A]$: probability of event A

$\mathbb{P}[A, B]$: joint probability of events A and B

$\mathbb{P}[A|B]$: probability of event A conditioned to event B

$\mathbb{E}[x]$: expectation value of variable x (over the probability distribution $\mathbb{P}[x]$)

$\langle F \rangle_p$: expected value of function $F(x)$ over the probability distribution $p(x)$

$\mathrm{Var}[A]$: variance of event A (over the probability distribution $\mathbb{P}[A]$)

$\text{Cov}[A,B]$: covariance of events A and B (over the probability distribution $\mathbb{P}[A,B]$)

SIS model

x_i^t state of node i at time t

$\overline{x}_i$: trajectory of node i (set of states od node i at each time)

$\boldsymbol{x}_A^t$: state of nodes $a \in A$ at time t

λ_{ji}: infection probability from node j to node i

ρ_i: recovery probability for node i

α_i: auto-infection probability for node i

1 INTRODUCTION

Imagine an epidemic spreading through a population, or some sort of fake news traveling from mouth to mouth and reaching a big portion of society. Or maybe think about neurons in the brain firing one after the other and thus building up thought and reasoning. In all three cases, we are able to observe the outcomes of some process (the health state of some individuals, the people convinced of a fact, or the activation of some neurons) and we would like to deduce the underlying essential elements that characterize such a process. Either to prevent something from happening, to favour a process, or to study a phenomenon that we observe, it would be useful to be aware of the parameters that govern such a phenomenon. In particular, one would like to reconstruct the network of interactions among individual elements of a group, given some observation of a process they undergo. The problem of using data to determine unknown properties of a system (technically, the probability distribution from which the data has been drawn) is called *statistical inference*.

In recent years, the scientific community has put a great amount of effort into trying to solve the problem of inferring the topology of a graph given the observation of a dynamical process taking place on it [1].

The problem can be solved relatively easily if the whole process is observed, but things become more intricate when observations are scattered over time and between graph nodes. In real situations, this is indeed the most interesting case, as one can not hope to obtain full knowledge of the trajectory of a complex process and can almost always rely only on incomplete and even partially wrong information.

Particularly interesting are the epidemic diffusion models, through which one can repre-

sent epidemics, of course, but also opinion dynamics, communication between computers and economic systems [2–6]. For this reason, the present work is aimed at performing inference of networks given observations of epidemic cascades.
In the past, important breakthroughs were made for what concerns non-recurrent models, i.e. those models in which the system can not find itself in a certain state for more than one time [7], because they are substantially easier to treat. Nevertheless, real processes are often recurrent: as an example, in the SARS-CoV-2 pandemic, individuals could be infected several times after having recovered. Moreover, non-recurrent models pose another problem: in order to effectively perform inference, a certain number of independent cascades has to be observed, because, for each possible edge, a cascade provides only a little number of events that indicate its presence or absence. This is hardly ever possible. It is much more common to observe a single, long time trajectory generated by a recurrent model.
Therefore, in this thesis the Susceptible-Infectious-Susceptible (SIS) model is taken into consideration and inference is performed on the infection and recovery probabilities, which are the parameters of the model and from which information can be gathered on the underlying network.
Thus, the problem is to infer the parameters of the model, given some (partial) observations of the time series of the system.

1.1 THE MODEL

Mathematically, the inference procedure is based on Bayes' theorem. Indeed, one wants to maximize the likelihood of the parameters given the observations. By calling Θ the set of all parameters, the problem is to find:

$$\arg\max_{\Theta}\left\{\mathbb{P}[\Theta|\text{obs}]\right\} \tag{1.1}$$

Thanks to Bayes' theorem this can be rewritten as:

$$\arg\max_{\Theta}\left\{\mathbb{P}[\Theta|\text{obs}]\right\} = \arg\max_{\Theta}\left\{\frac{\mathbb{P}[\text{obs}|\Theta]\mathbb{P}[\Theta]}{\mathbb{P}[\text{obs}]}\right\} = \arg\max_{\Theta}\left\{\mathbb{P}[\text{obs}|\Theta]\mathbb{P}[\Theta]\right\} \tag{1.2}$$

where $\mathbb{P}[\Theta]$ is some prior probability distribution on the parameters. In the case of a uniform prior (that corresponds to no prior) the problem reduces to finding:

$$\arg\max_{\Theta}\left\{\mathbb{P}[\text{obs}|\Theta]\right\} = \arg\max_{\Theta}\left\{\mathscr{L}(\Theta)\right\} \tag{1.3}$$

where the likelihood function $\mathscr{L}(\Theta) := \mathbb{P}[\text{obs}|\Theta]$ has been defined. This task is non-trivial because the likelihood is untractable with standard methods.
In order to tackle the problem, the likelihood can be rewritten in a statistical-mechanical language, by first conditioning on all the possible trajectories the system can cover:

$$\mathscr{L}(\Theta) = \sum_{\text{traj}} \mathbb{P}[\text{obs}|\text{traj}]\mathbb{P}[\text{traj}|\Theta] \tag{1.4}$$

It is clear that here the first probability term represents the compatibility between the trajectory traj and the observations, while the second one represents the underlying model itself.
The terms of the summation can be written as Boltzmann weights, thus recognizing the expression of a partition function:

$$\mathscr{L}(\Theta) = \sum_{\text{traj}} e^{-H(\text{traj})} = Z(\text{obs}, \Theta) \tag{1.5}$$

Finally, by defining a free energy $F(\Theta) = -\log Z(\text{obs}, \Theta)$, one has recast the initial maximum-likelihood problem in a minimum-free-energy one:

$$\arg\max_{\Theta} \mathscr{L}(\Theta) = \arg\min_{\Theta} F(\Theta) \tag{1.6}$$

The optimization problem can now be solved with an alternation of calculation of the free energy and optimization of the parameters.
This procedure is not completely new: the idea of alternating steps of distribution reconstruction and steps of parameter optimization is proper of Expectation-Maximization, a statistical technique used to find maximum likelihood estimates of parameters in a statistical model [8]. The following chapters will delve into the details of all the techniques and models exploited to obtain a working method.

2 Prerequisites

For the accurate description of the methods used in the course of the thesis work, some background knowledge is necessary. This chapter is aimed at providing the reader with the essential tools needed to fully understand the following work. It will inspect the SIS model, the BP algorithm and the formalism of Tensor Trains, the essential elements giving rise to MPBP. In the end, a brief introduction to Gradient Ascent will be provided.

2.1 The SIS Model

When modelling epidemics, one usually employs compartmental models. These consist of dividing a population into a few compartments, each representing a status of individuals with respect to the disease. Then, having specified some initial conditions (i.e. the compartment each individual belongs to at the initial time), at any time between the initial one $t = 0$ and the final one $t = T$, individuals will have a probability of changing state, that depends on their current state and on that of the other individuals. The simplest method one can come up with is the Susceptible-Infectious model. Individuals are represented by nodes of a graph and they can be in two states: Susceptible (S) and Infectious (I). An edge (ji) connecting node j to node i means that the former can infect the latter, if conditions permit[1]. More precisely, if at time t node j is Infectious and node i is Susceptible, there is a probability λ_{ji} that node i will become Infectious at time $t+1$. The natural generalization of this model consists in allowing the infectious nodes to

[1]Notice that here edges are considered as directed

recover, giving rise to the Susceptible-Infectious-Susceptible model. According to it, at time t, if node i is infectious, there is a probability ρ_i that it will be Susceptible at time $t+1$.
Additionally, a small auto-infection probability α_i can be introduced. This is particularly useful when performing simulations, in order to avoid trajectories that stop before the final time. Indeed, the SIS model has an absorbing state, consisting in all the nodes being Susceptible.
In mathematical terms, calling $w_i^{t+1}\left(x_i^{t+1}, x_i^t, \boldsymbol{x}_{\partial i}^t\right) := \mathbb{P}\left[x_i^{t+1} | x_i^t, \boldsymbol{x}_{\partial i}^t\right]$ the transition probability for node i, the SIS model can be framed as follows:

$$\begin{aligned}
&w_i^{t+1}\left(x_i^{t+1}=S, x_i^t=I, \boldsymbol{x}_{\partial i}^t\right)=\rho_i \\
&w_i^{t+1}\left(x_i^{t+1}=I, x_i^t=I, \boldsymbol{x}_{\partial i}^t\right)=1-\rho_i \\
&w_i^{t+1}\left(x_i^{t+1}=S, x_i^t=S, \boldsymbol{x}_{\partial i}^t\right)=\left(1-\alpha_i\right)\left(\prod_{j\in\partial i}\left(1-\lambda_{ji}\mathbb{I}\left[x_j^t=I\right]\right)\right) \\
&w_i^{t+1}\left(x_i^{t+1}=S, x_i^t=S, \boldsymbol{x}_{\partial i}^t\right)=1-\left(1-\alpha_i\right)\left(\prod_{j\in\partial i}\left(1-\lambda_{ji}\mathbb{I}\left[x_j^t=I\right]\right)\right)
\end{aligned} \tag{2.1}$$

These, together with the initial probabilities $w_i^0\left(x_i^0\right) := \mathbb{P}\left[x_i^0\right]$, give the probability of a trajectory of a single node:

$$\Phi_i\left(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}\right) := \frac{1}{Z_i}\mathbb{P}\left[\overline{x}_i | \overline{\boldsymbol{x}}_{\partial i}\right] = w_i^0\left(x_i^0\right)\prod_{t=1}^{T} w_i^t\left(x_i^t, x_i^{t-1}, \boldsymbol{x}_{\partial i}^{t-1}\right) \tag{2.2}$$

Then, the probability for a trajectory of the whole system is:

$$\Phi(\overline{\boldsymbol{x}}) := \mathbb{P}\left[\overline{\boldsymbol{x}}\right] = \frac{1}{Z}\prod_{i\in\mathcal{V}}\Phi_i\left(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}\right) = \prod_{i\in\mathcal{V}}\left(w_i^0\left(x_i^0\right)\prod_{t=1}^{T} w_i^t\left(x_i^t, x_i^{t-1}, \boldsymbol{x}_{\partial i}^{t-1}\right)\right) \tag{2.3}$$

It is worth to notice a huge difference between the SI model and the SIS model. The former is non-recurrent, namely a node can not switch back to a state once it has gone away from it. This implies that the trajectory of a node can be represented by just one integer number in $[0, T]$, that is the time at which it goes from the S state to the I state. The latter, instead, is a recurrent model, which means that a node, after having been Infected, can become Susceptible, then Infected again and so on. For this reason, the number of possible trajectories of a single individual is exponential in T, thus making the model much more difficult to treat.

2.2 THE BELIEF PROPAGATION ALGORITHM

In applications of probability theory, it is often required to compute single-variable marginals of a multivariate distribution. For modest numbers of variables this is easy, but it becomes computationally unfeasible as soon as the number of variables grows. As an example, consider a probability distribution $\mathbb{P}[\boldsymbol{x} = (x_1, \dots, x_n)]$ depending on $n = 100$ binary variables x_i. If one wanted to calculate the i^{th} marginal the naïve way, this would mean to perform the calculation $\mathbb{P}[x_i] = \sum_{\boldsymbol{x}' : x'_i = x_i} \mathbb{P}[\boldsymbol{x}']$, requiring to evaluate $\mathbb{P}[\boldsymbol{x}']$ at 2^{99} different values of $\boldsymbol{x}'$, a prohibitive amount of calculations.

If $\mathbb{P}[\boldsymbol{x}]$ factorizes conveniently[2], instead, the probability distribution can be represented by means of a factor graph (see Appendix A) and, if the latter has a suitable topology, the computation can be performed in an easier and more efficient way by means of Belief Propagation (BP) [9].

The BP algorithm, also called Sum-product message passing, consists of an exchange of functions (actually called "messages") between variable nodes and factor nodes.

More precisely, given a variable node i and a factor node a such that f_a depends on x_i, the message from i to a is:

$$\mu_{i \to a}(x_i) = \frac{1}{z_{i \to a}} \prod_{b \in \partial i \setminus \{a\}} \mu_{b \to i}(x_i) \tag{2.4}$$

while the message from a to i is:

$$\mu_{a \to i}(x_i) = \frac{1}{z_{a \to i}} \sum_{\boldsymbol{x}_{a \setminus i}} f_a(\boldsymbol{x}_a) \prod_{j \in \partial a \setminus \{i\}} \mu_{j \to a}(x_j) \tag{2.5}$$

The one above is a system of equations for the messages and from the solution one can compute the beliefs:

$$b_i(x_i) = \frac{1}{z_i} \prod_{a \in \partial i} \mu_{a \to i}(x_i) \tag{2.6}$$

A derivation of the above formulae can be found in Appendix B for the case of an acyclic factor graph. Moreover, factor beliefs can be defined:

$$b_a(\boldsymbol{x}_a) = \frac{1}{z_a} f_a(\boldsymbol{x}_a) \prod_{i \in \partial a} \mu_{i \to a}(x_i) \tag{2.7}$$

A particular case is represented by the factor graph being acyclic. In that case, whenever

[2]Think about the SIS model in section 2.1, in which the probability distribution for a whole trajectory is factorized over the nodes, albeit containing some interactions.

there is a factor leaf a connected to a variable node i the update rule gives (notice that in this case $\boldsymbol{x}_a = x_i$):

$$\mu_{a \to i}(x_i) \propto f_a(x_i) \tag{2.8}$$

and similarly for a variable leaf i connected to a factor a:

$$\mu_{i \to a}(x_i) \sim \text{Uniform} \tag{2.9}$$

This means that on an acyclic factor graph, the system in Equation 2.4 and 2.5 can be solved starting from the leaves and proceeding towards the route of the graph. In this case the beliefs are exactly equal to the marginals $b_i(x_i) = \mathbb{P}[x_i]$.

When the factor graph contains cycles, the system can be considered as a fixed point problem. Through iterations and successive updates, one can compute an approximation to the solution and thus to the marginals [10]. The accuracy of the approximation depends on the topology of the graph: in general, BP fails to give good results if the graph contains short loops, although in the past several methods have been developed to compute more and more precise corrections [10–13].

2.3 TENSOR TRAINS

Tensor trains, as the simple form of tensor networks, are a mathematical tool that was first developed and is mostly used in the field of quantum physics, to parametrize variational quantum states [14]. Actually, their applications can be much more diverse and varied [15]. In the present work, tensor trains will be used to parametrize multivariate probability distributions, in order to render them more tractable.

Given n discrete variables $\boldsymbol{x} = x_1, \dots x_n$, the probability distribution $p(\boldsymbol{x})$ written as a tensor train has the form:

$$p(\boldsymbol{x}) = A^1(x_1)A^2(x_2)\dots A^n(x_n) \tag{2.10}$$

where each one of the $A^i(x_i)$ is a matrix $A^i(x_i) \in \mathbb{R}^{d_i \times d_{i+1}}$, whose entries all depend on the value of x_i. The quantities d_i are called the *bond dimensions* of the matrices and this condition automatically ensures that the matrix products are consistently defined. In order $p(\boldsymbol{x})$ to be a scalar, it must be $d_1 = d_{n+1} = 1$, i.e. $A^1(x_1)$ is a row vector and $A^n(x_n)$ is a column vector.

The constraint of the x_i's being discrete serves to ensure that the matrices are representable. Indeed, they can be thought as tensors $A^i \in \mathbb{R}^{d_i \times d_{i+1} \times q_i}$, where q_i is the number of values that x_i can assume.

This representation should remind the form of a distribution of independent variables[3]; in that case it would be:

$$p(\boldsymbol{x}) = p_1(x_1)p_2(x_2)\dots p_n(x_n) \tag{2.11}$$

However, when writing the function as a tensor train, it is not a product of scalar functions. Indeed, if the matrices all have 1 row and 1 column, the form 2.11 is recovered, but for general dimensions of matrices, the expression 2.10 is a mixture of mean-fields, thus being much more general. The final form, recovered after all the matrix products, is much more complex and much more versatile in reproducing whatever was the initial probability distribution. Actually, it can be shown that tensor trains can represent any function, with big enough matrices [16].
At the same time, however, the form is factorized, albeit in a product of matrices, and it will be shown that this allows to perform rather tedious calculations in a convenient manner. In the case of the SIS model, for example, it will be very convenient to represent the trajectory probabilities $\Phi_i(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i})$ in a matrix product state, factorized over different times.

2.4 GRADIENT ASCENT

Gradient Ascent is probably the most famous algorithm to perform numerical optimization; it is particularly useful when one wants to find the maximum of a function with several inputs, as analytical methods become useless in that case. For this reason it is vastly used in the field of Machine Learning and Neural Networks, where functions of thousands or millions of parameters have to be optimized.
For a better visualization, however, it is useful to think about a function $F(x, y)$ of just two variables and its graph, as shown in Figure 2.1.
The idea behind the GA algorithm is to “climb up the graph" of the function, in order to reach a peak. This can be done by realizing that the gradient $\nabla F(x, y)$ evaluated at any point, represents the direction of steepest ascent up the profile of the function. Then, one could try to start at a random point and take a small step in the direction of the gradient, then repeat this procedure; this is actually “climbing" the graph of the function, as if it were a hill.
In a more algorithmic language, Gradient Ascent could be framed as follows:

[3]Or, in the language of a statistical physicist, a mean-field distribution

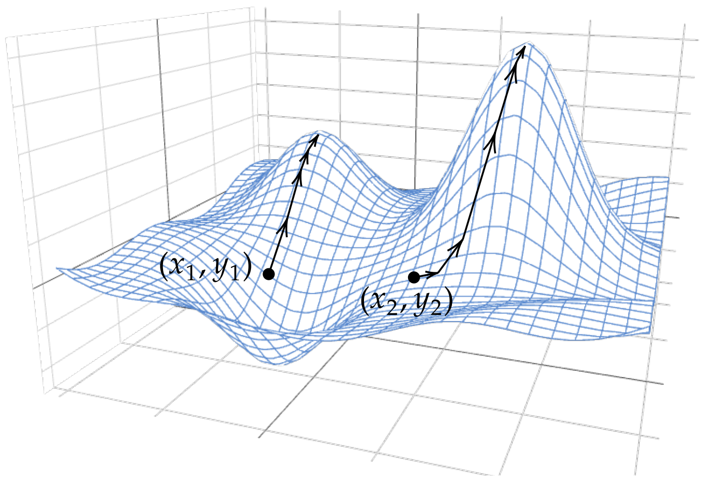

FIGURE 2.1. Gradient ascent applied to a 2-dimensional function. When starting from (x_1, y_1) the algorithm does not converge to the global maximum, while it does when starting from (x_2, y_2).

```
Algorithm: Gradient Ascent
    Take $\boldsymbol{x}_0$ at random
    while $\|\boldsymbol{x}_n - \boldsymbol{x}_{n-1}\| > \varepsilon$ (or until a max number of iterations is reached) do
        Calculate $\nabla F(\boldsymbol{x})$
        Update $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \eta \nabla F(\boldsymbol{x})$
    end
```

The threshold ε and the learning rate η are parameters of the algorithm, together with the maximum number of iterations. In particular, the learning rate is crucial in determining the behaviour of the algorithm: if it is too small, the ascent can be slow, if it is too large, the steps can be too long and the algorithm can overshoot, never reaching a convergence.

To address these issues, several variants of GA have been designed, usually modifying the update step to make it faster or more robust. Among them, notable ones are momentum-based methods [17], Adam [18] and Adagrad [19]. Even simple ones can be used in which fixed-length steps are taken in the direction of the gradient. A list of variants and methods can be found in [20].

3 Matrix-Product Belief Propagation

As seen, the Belief Propagation algorithm can be used to compute the marginals of a probability distribution written in factorized form. However, in some cases, this is computationally unfeasible because the support of the individual factors is too large. As an example, consider a dynamical system for binary variables (spins) which can take value $+1$ or -1 at each time instant: for T times, a single-node trajectory has 2^T possible values, so the computation becomes prohibitive even for moderately large times.
To address this issue, a variant of the algorithm was proposed in [21] that relies on the tensor train representation of probability distributions, and it was called Matrix-Product Belief Propagation. In the following, the method will be analyzed in detail, taking the SIS model as a working example.

3.1 Building the Factor Graph

The first tool that is needed for Belief Propagation is a factor graph. For an SIS model defined over a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$[1], a reasonable choice is to take the trajectories $\overline{x}_i$ as variables and the probabilities of single trajectories $\Phi_i(\overline{x}_i, \overline{x}_{\partial i})$ as factors. This actually allows for the inclusion of reweighting terms ϕ_i^t for the trajectory:

$$\Phi_i(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}) = \frac{1}{Z_i} w_i^0(x_i^0) \prod_{t=1}^{T} w_i^t(x_i^t, x_i^{t-1}, \boldsymbol{x}_{\partial i}^{t-1}) \prod_{t=0}^{T} \phi_i^t(x_i^t) := \frac{1}{Z_i} \prod_{t=0}^{T} f_i^t(x_i^t, x_i^{t-1}, \boldsymbol{x}_{\partial i}^{t-1}) \tag{3.1}$$

[1]This is the graph on which the model is defined, not to be confused with the factor graph

These terms are particularly useful when studying large deviations of the system, i.e. trajectories that have a very low probability of being observed. Practically, the ϕ_i^t's can encode the observations of the true trajectory that one wants to take into account when calculating the marginals[2].

Constructing the factor graph this way, there would be one factor node for each variable node and each factor would be connected, additionally to its own variable i, to all the variables $j \in \partial i$ neighbours of i in the original graph $\mathcal{G}$. This would introduce numerous loops that would have a negative impact on the performances of Belief Propagation (Figure 3.1, top right).

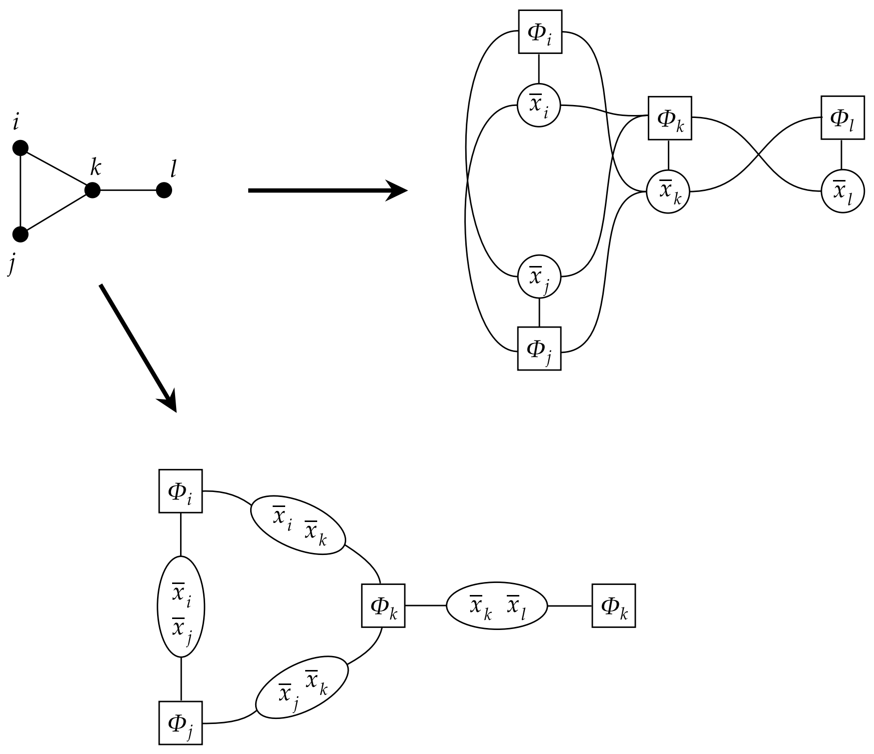

FIGURE 3.1. From the same starting graph $\mathcal{G}$ (top left), one can construct a naïve factor graph with many loops (top right) or a more refined factor graph that does not introduce additional loops (bottom).

Instead of doing this, as explored in [22], one can consider the dual factor graph, in which pairs of neighbouring nodes' trajectories (i.e. the edges of $\mathcal{G}$) are taken as variable

[2]For example, if one observes that node 7 is Infectious at time 4, to take this into account it is sufficient to set $\phi_7^4(x) = \delta_{x,I}$

nodes and the factor nodes are the nodes of $\mathcal{G}$. In this case, factors must also enforce consistency among the pairs of trajectories (e.g. in Figure 3.1, the factor Φ_i must ensure that the trajectory $\overline{x}_i$ in the node below is the same as that in the node on the right). This is easy to realize in practice.
As it is clear from Figure 3.1, the dual factor graph built this way has the same topology as the original one.
Moreover, some simplifications can be made to the BP equations. Given Equation 2.4, one can notice that the message from a variable to a factor is nothing else than the message incoming in the variable from the other side:

$$\mu_{(\overline{x}_i\overline{x}_j)\to\Phi_i} \propto \mu_{\Phi_j\to(\overline{x}_i\overline{x}_j)} \tag{3.2}$$

For this reason, it is possible to consider just messages from one factor to another one and to call them $\mu_{i\to j}$. The update equations become:

$$\mu_{i\to j}\left(\overline{x}_i,\overline{x}_j\right) = \frac{1}{z_{i\to j}} \sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}} \Phi_i\left(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}\right) \prod_{k\in\partial i\setminus j} \mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right) \tag{3.3}$$

and the beliefs are:

$$b_i\left(\overline{x}_i\right) = \frac{1}{z_i} \sum_{\overline{x}_i} \sum_{\overline{\boldsymbol{x}}_{\partial i}} \Phi_i\left(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}\right) \prod_{k\in\partial i} \mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right) \tag{3.4}$$

As previously stated, an exact representation of the messages would require an exponentially large space in the maximum time T, hence the tensor-train approximation is employed.

3.2 MESSAGES AS MATRIX-PRODUCT STATES

Since an exact representation of the messages is unfeasible, they are represented as Matrix-Product States (or Tensor Trains):

$$\mu_{i\to j}\left(\overline{x}_i,\overline{x}_j\right) \propto \prod_{t=0}^{T} A^t_{i\to j}\left(x^t_i,x^t_j\right) \tag{3.5}$$

As usual, this means that, for each $\left(x^t_i,x^t_j\right)$, $A^t_{i\to j}$ is a real-valued matrix. Equivalently, $A^t_{i\to j}\left(x^t_i,x^t_j\right)$ can be thought of as a 4-dimensional tensor, with two axes that are the ones of the matrix and the other two corresponding to x^t_i and x^t_j.
Now, one has to check whether messages remain parametrized as MPSs when passing

through the update equations. Substituting Equation 3.5 into Equation 3.3:

$$\mu_{i\to j}\left(\overline{x}_i,\overline{x}_j\right) \propto \sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}} \Phi_i\left(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}\right) \prod_{k\in\partial i\setminus j}\prod_{t=0}^{T} A^t_{k\to i}\left(x^t_k,x^t_i\right) \propto \tag{3.6}$$

$$\propto \prod_{t=0}^{T} \sum_{\left\{x^t_k\right\}_{k\in\partial i\setminus j}} f^{t+1}_i\left(x^{t+1}_i,x^t_i,\boldsymbol{x}^t_{\partial i}\right) \bigotimes_{k\in\partial i\setminus j} A^t_{k\to i}\left(x^t_k,x^t_i\right) \propto \tag{3.7}$$

$$\propto \prod_{t=0}^{T} B^t_{i\to j}\left(x^{t+1}_i,x^t_i,x^t_j\right) \tag{3.8}$$

where $B^t_{i\to j}\left(x^{t+1}_i,x^t_i,x^t_j\right) := \sum_{\left\{x^t_k\right\}_{k\in\partial i\setminus j}} f^{t+1}_i\left(x^{t+1}_i,x^t_i,\boldsymbol{x}^t_{\partial i}\right) \otimes_{k\in\partial i\setminus j} A^t_{k\to i}\left(x^t_k,x^t_i\right)$. The symbol $\otimes$ represents a tensor product: given two vector spaces U and V, it maps a pair $(u\in U, v\in V)$ into an element of $U\otimes V$, denoted as $u\otimes v$.[3]

There is a problem: if incoming tensors all have bond dimension M, the resulting one has bond dimension $M^{|\partial i|-1}$ (when reshaped to actually be a matrix), so the tensors' dimensions grow at each iteration of the update equations.

To solve this and to recover the form 3.5, similarly to [23], two sweeps of Singular-Value Decomposition (see Appendix C) are performed.

The first sweep happens from left to right (i.e. from $t=0$ to $t=T$). For each time, the tensor $B^t_{i\to j}\left(x^{t+1}_i,x^t_i,x^t_j\right)$ is decomposed[4]:

$$B^t_{i\to j}\left(x^{t+1}_i,x^t_i,x^t_j\right) = C^t_{i\to j}\left(x^t_i,x^t_j\right)\Lambda^t\left[V^t\left(x^{t+1}_i\right)\right]^\dagger \tag{3.9}$$

and then one matrix is retained, while the others are incorporated into the next one:

$$B^{t+1}_{i\to j}\left(x^{t+1}_i,x^t_i,x^t_j\right) \leftarrow \Lambda^t\left[V^t\left(x^{t+1}_i\right)\right]^\dagger B^{t+1}_{i\to j}\left(x^{t+1}_i,x^t_i,x^t_j\right) \tag{3.10}$$

Reminding that, in tensor trains, the last matrix is actually a column vector, one realizes that the sequence of $C^t_{i\to j}$ matrices obtained after this sweep is in the form of the ansatz of Equation 3.5.

The second sweep is aimed at reducing the size of the matrices, discarding the smallest singular values. The fact that, during truncation, the MPS is in "mixed canonical form" ensures that the global truncation error is minimal [24]. If the bond dimension of incoming matrices was M, the truncation goes as follows (this time the sweep is from $t=T$ to $t=0$ and x^t_i and x^t_j are incorporated in the column index):

$$C^t_{i\to j}\left(x^t_i,x^t_j\right) = U^t\Lambda^t A^t_{i\to j}\left(x^t_i,x^t_j\right) \tag{3.11}$$

[3]If u and v are matrices, for instance, the element $u\otimes v$ will be a tensor with four indices.

[4]Notice that, as discussed in Appendix C, the variables x^t_i and x^t_j are incorporated in the row index, while x^{t+1}_i is incorporated in the column index.

After truncation, that reduces the size of $A^t_{i\to j}$, U^t and Λ^t, the latter matrices are brought to the left, while $A^t_{i\to j}$ is retained:

$$C^{t-1}_{i\to j}\left(x^{t-1}_i, x^{t-1}_j\right) \leftarrow C^{t-1}_{i\to j}\left(x^{t-1}_i, x^{t-1}_j\right) \tilde{\Lambda}^t \tilde{U}^t \tag{3.12}$$

As before, given that the first tensor is a row vector, the final form of the tensor train is the same as in Equation 3.5 and all the matrices have the same dimension.
As a final side note, it is worth mentioning two possible problems.
When performing SVD truncation, tensor trains are not guaranteed to maintain the properties of a probability distribution; it is possible, as an example, that the truncated tensor train has negative values for some inputs. This problem is solved by increasing the bond dimension, making the tensor train approximation more precise in reconstructing the original probability distribution. Moreover, for long tensor trains, underflow errors are very likely to occur[5]. To address this issue, the matrices $C^t_{i\to j}$ and $A^t_{i\to j}$ can be normalized at each step of SVD, storing the normalization in a separate variable.

3.3 EFFICIENT MPBP COMPUTATIONS

As pointed out in [25], the computational cost to perform the sweeps of SVD is $\mathcal{O}\left(M^{2z-1}\right)$ for a node with z neighbours and matrices with bond dimension M. This exponential dependence on the degree is problematic, as even in the case of sparse graphs some high-degree nodes are often present (this is the case, for example, of Erdős-Rényi random graphs). Fortunately, for a wide class of models, including epidemic ones, a recursive and efficient scheme can be devised that reduces the computational cost to $\mathcal{O}\left(M^6\right)$.
This can be applied to models for which the transition probabilities $w_i\left(x^{t+1}_i, x^t_i, \boldsymbol{x}^t_{\partial i}\right)$ depend on $\boldsymbol{x}_{\partial i}$ only through some auxiliary variables which summarize the aggregate interaction with the neighbours. For the SIS model, one can write the probability of an infection being transmitted by some neighbour of i to i itself through auxiliary variables $\overline{y}_{\partial i}$ (here y^t_A with $A \in \partial i$ is the probability that at least one among the nodes $k \in A$ transmits the disease to i at time t):

$$w^{t+1}_i\left(x^{t+1}_i, \boldsymbol{x}^t_{\partial i}, x^t_i\right) = \mathbb{P}\left[x^{t+1}_i | \boldsymbol{x}^t_{\partial i}, x^t_i\right] = \sum_{y^t_{\partial i}} \mathbb{P}\left[x^{t+1}_i | y^t_{\partial i}, x^t_i\right] \mathbb{P}\left[y^t_{\partial i} | \boldsymbol{x}^t_{\partial i}, x^t_i\right] \tag{3.13}$$

Notice that $y^t_{\partial i}$ is an aggregated quantity, taking into account the cumulative effect of all the nodes $k \in \partial i$. It is different from the set $\{y^t_k\}_{k\in\partial i}$.

[5]Because many matrices with entries smaller than one are multiplied the one for the other.

For such y quantities it holds[6]:

$$\mathbb{P}\left[y^t_{A\cup B}|y^t_A,y^t_B,x^t_i\right]=\mathbb{I}\left[y^t_{A\cup B}=I\right]\mathbb{P}\left[y^t_A=I\vee y^t_B=I\right]+\mathbb{I}\left[y^t_{A\cup B}=S\right]\mathbb{P}\left[y^t_A=S\wedge y^t_B=S\right] \tag{3.14}$$

so that one can write:

$$w^{t+1}_i\left(x^{t+1}_i,\boldsymbol{x}^t_{\partial i},x^t_i\right)=\sum_{\left\{y^t_k\right\}_{k\in\partial i}}\mathbb{P}\left[x^{t+1}_i|\left\{y^t_k\right\}_{k\in\partial i},x^t_i\right]\prod_{k\in\partial i}\mathbb{P}\left[y^t_k|x^t_k,x^t_i\right] \tag{3.15}$$

with

$$\begin{aligned}\mathbb{P}\left[x^{t+1}_i=S|\left\{y^t_k\right\}_{k\in\partial i},x^t_i\right]&=\mathbb{P}\left[x^{t+1}_i=S|\gamma\left(\left\{y^t_k\right\}_{k\in\partial i}\right),x^t_i\right]=\\&=(1-\alpha_i)\delta\left(y^t_{\partial i},S\right)\mathbb{I}\left[x^t_i=S\right]+\rho_i\mathbb{I}\left[x^t_i=I\right]\end{aligned} \tag{3.16}$$

$$\begin{aligned}\mathbb{P}\left[x^{t+1}_i=I|\left\{y^t_k\right\}_{k\in\partial i},x^t_i\right]&=\mathbb{P}\left[x^{t+1}_i=I|\gamma\left(\left\{y^t_k\right\}_{k\in\partial i}\right),x^t_i\right]=\\&=\left(\alpha_i+\delta\left(y^t_{\partial i},I\right)-\alpha_i\delta\left(y^t_{\partial i},I\right)\right)\mathbb{I}\left[x^t_i=S\right]+\left(1-\rho_i\right)\mathbb{I}\left[x^t_i=I\right]\end{aligned} \tag{3.17}$$

and

$$\mathbb{P}\left[y^t_k|x^t_k,x^t_i\right]=\mathbb{I}\left[x^t_k=I\right]\left(\lambda_{ki}\mathbb{I}\left[y^t_k=I\right]+(1-\lambda_{ki})\mathbb{I}\left[y^t_k=S\right]\right)+\mathbb{I}\left[x^t_k=S\right]\mathbb{I}\left[x^t_k=S\right] \tag{3.18}$$

Equation 3.14 means that the y's of disjoint index sets are conditionally independent, given the corresponding x's. At this point, the BP update equations' computation can be simplified by means of the auxiliary variables[7]:

$$\begin{aligned}\mu_{i\to j}\left(\overline{x}_i,\overline{x}_j\right)&=\sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}}w^0_i\left(x^0_i\right)\prod_{t=0}^{T-1}w^{t+1}_i\left(x^{t+1}_i,\boldsymbol{x}^t_{\partial i},x^t_i\right)\prod_{k\in\partial i\setminus j}\mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right)=\\&=\sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}}\sum_{\overline{y}_{\partial i\setminus j}}w^0_i\left(x^0_i\right)\prod_{t=0}^{T-1}\left(\mathbb{P}\left[x^{t+1}_i|y^t_{\partial i},x^t_i\right]\mathbb{P}\left[y^t_{\partial i}|x^t_k,x^t_i\right]\right)\prod_{k\in\partial i\setminus j}\mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right)=\\&=\sum_{\overline{y}_{\partial i\setminus j}}\left(w^0_i\left(x^0_i\right)\prod_{t=0}^{T-1}\mathbb{P}\left[x^{t+1}_i|y^t_{\partial i},x^t_i\right]\right)\sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}}\prod_{t=0}^{T-1}\mathbb{P}\left[y^t_{\partial i}|x^t_k,x^t_i\right]\prod_{k\in\partial i\setminus j}\mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right)=\\&=\sum_{\overline{y}_{\partial i\setminus j}}\left(w^0_i\left(x^0_i\right)\prod_{t=0}^{T-1}\mathbb{P}\left[x^{t+1}_i|y^t_{\partial i},x^t_i\right]\right)\tilde{\mu}_{\partial i\setminus j\to i}\left(\overline{y}_{\partial i\setminus j},\overline{x}_i\right)\end{aligned} \tag{3.19}$$

[6]Notice that the following is actually a map $\gamma:\mathbb{P}\left[y^t_A,y^t_B\right]\mapsto\mathbb{P}\left[y^t_{A\cup B}\right]$.

[7]Here observations are neglected. They can be reintroduced at any time simply by multiplying them at the appropriate positions.

where $\tilde{\mu}_{\partial i\setminus j\to i}\left(\overline{y}_{\partial i\setminus j},\overline{x}_i\right)=\sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}}\prod_{t=0}^{T-1}\mathbb{P}\left[y^t_{\partial i}|x^t_k,x^t_i\right]\prod_{k\in\partial i\setminus j}\mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right)$ and it can be calculated by means of a recursion:

$$\begin{cases}\tilde{\mu}_{A\cup B\to i}\left(\overline{y}_{A\cup B},\overline{x}_i\right)=\sum\limits_{\overline{y}_A,\overline{y}_B}\prod\limits_{t=0}^{T-1}\mathbb{P}\left[y^t_{A\cup B}|y^t_A,y^t_B,x^t_i\right]\tilde{\mu}_{A\to i}\left(\overline{y}_A,x_i\right)\tilde{\mu}_{B\to i}\left(\overline{y}_B,x_i\right)\\ \tilde{\mu}_{\{k\}\to i}\left(\overline{y}_k,\overline{x}_i\right)=\sum\limits_{\overline{x}_k}\prod\limits_{t=0}^{T-1}\mathbb{P}\left[y^t_k|x^t_k,x^t_i\right]\mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right)\\ \tilde{\mu}_{\emptyset\to i}\left(\overline{y}_\emptyset,\overline{x}_i\right)\propto 1\end{cases}\tag{3.20}$$

The last thing that is left to show is how to perform efficiently the recursive calculation in Equation 3.20.

By writing the messages $\tilde{\mu}_{A\cup B\to i}$ as MPSs, the recursion for each matrix individually is:

$$\tilde{A}^t_{A\cup B\to i}\left(y^t_{A\cup B},x^t_i\right)=\sum_{y^t_A}\sum_{y^t_B}\mathbb{P}\left[y^t_{A\cup B}|y^t_A,y^t_B,x^t_i\right]\tilde{A}^t_{A\to i}\left(y^t_A,x^t_i\right)\otimes\tilde{A}^t_{B\to i}\left(y^t_B,x^t_i\right)\tag{3.21}$$

that, written in components, is:

$$\begin{aligned}&\left[\tilde{A}^t_{A\cup B\to i}\left(y^t_{A\cup B},x^t_i\right)\right]_{(a^t,b^t),(a^{t+1},b^{t+1})}=\\&\qquad=\sum_{y^t_A}\sum_{y^t_B}\mathbb{P}\left[y^t_{A\cup B}|y^t_A,y^t_B,x^t_i\right]\left[\tilde{A}^t_{A\to i}\left(y^t_A,x^t_i\right)\right]_{a^t,a^{t+1}}\left[\tilde{A}^t_{B\to i}\left(y^t_B,x^t_i\right)\right]_{b^t,b^{t+1}}\end{aligned}\tag{3.22}$$

The matrices on the left-hand side have a size double than that of the matrices on the right-hand side, so one has to perform an SVD truncation to restore the correct dimensions. The truncation is performed on $\tilde{A}^t_{A\cup B\to i}\left(y^t_{A\cup B},x^t_i\right)$ reshaped to have $\left(a^t,b^t\right)$ as row index and $\left(a^{t+1},b^{t+1},y^t_{\partial i\setminus j},x^t_i\right)$ as column index; given that incoming matrices have bond dimension M, it has a cost $\mathcal{O}\left(qYM^6\right)$, where Y is the number of values that $y^t_{\partial i\setminus j}$ can assume[8]. Therefore, if Y depends at most polynomially on $|\partial i|$, the exponential dependence on the degree can be avoided.

In conclusion, the following algorithm can compute messages from node i to its neighbours, with the function `AggrMsg` that is used to aggregate two messages together (here $\leq j$ means $\{k\in\partial i : k\leq j\}$):

[8]As the computational cost to perform SVD on a matrix of size $m\times n$ is $\min\{mn^2,m^2n\}$

```
Function AggrMsg($\tilde{\mu}_{A\to i}(\overline{y}_A, x_i)$, $\tilde{\mu}_{B\to i}(\overline{y}_B, x_i)$):
    for $t \in 1:T$ do
        Take $\tilde{A}^t_{A\to i}(y^t_A, x^t_i)$ and $\tilde{A}^t_{B\to i}(y^t_B, x^t_i)$
        Compute $\tilde{A}^t_{A\cup B\to i}(y^t_{A\cup B}, x^t_i)$ with 3.21
        Take $\mathbb{P}[y^t_A|\boldsymbol{x}^t_A, x^t_i]$ and $\mathbb{P}[y^t_B|\boldsymbol{x}^t_B, x^t_i]$
        Compute $\mathbb{P}[y^t_{A\cup B}|y^t_A, y^t_B, x^t_i]$ with 3.14
    end
    SVD sweep from left to right (3.9, 3.10)
    SVD sweep from right to left with truncation (3.11, 3.12)
    Compute $\tilde{\mu}_{A\cup B\to i}(\overline{y}_{A\cup B}, \overline{x}_i)$ with 3.20
    return $\tilde{\mu}_{A\cup B\to i}(\overline{y}_{A\cup B}, \overline{x}_i)$
end
```

Algorithm: MPBP message updates

```
$\tilde{\mu}_{\emptyset\to i}(\overline{y}_{\emptyset}, \overline{x}_i) = 1$
for $k \in \partial i$ do
    Compute $\tilde{\mu}_{\{k\}\to i}(\overline{y}_k, \overline{x}_i)$ with 3.20
    Compute $\mathbb{P}[y^t_k|x^t_k, x^t_i]$ for $t \in 1:T$ with 3.18
end
for $j \in \partial i$ (in increasing order) do
    Take $\tilde{\mu}_{\le j-1\to i}(\overline{y}_{\le j-1}, \overline{x}_i)$ and $\tilde{\mu}_{\{j\}\to i}(\overline{y}_{\{j\}}, \overline{x}_i)$
    $\tilde{\mu}_{\le j\to i}(\overline{y}_{\le j}, \overline{x}_i)$ = AggrMsg($\tilde{\mu}_{\le j-1\to i}(\overline{y}_{\le j-1}, \overline{x}_i)$, $\tilde{\mu}_{\{j\}\to i}(\overline{y}_{\{j\}}, \overline{x}_i)$)
end
for $j \in \partial i$ (in decreasing order) do
    Take $\tilde{\mu}_{\ge j+1\to i}(\overline{y}_{\ge j+1}, \overline{x}_i)$ and $\tilde{\mu}_{\{j\}\to i}(\overline{y}_{\{j\}}, \overline{x}_i)$
    $\tilde{\mu}_{\ge j\to i}(\overline{y}_{\ge j}, \overline{x}_i)$ = AggrMsg($\tilde{\mu}_{\ge j+1\to i}(\overline{y}_{\ge j+1}, \overline{x}_i)$, $\tilde{\mu}_{\{j\}\to i}(\overline{y}_{\{j\}}, \overline{x}_i)$)
end
for $j \in \partial i$ do
    Take $\tilde{\mu}_{\le j-1\to i}(\overline{y}_{\le j-1}, \overline{x}_i)$ and $\tilde{\mu}_{\ge j+1\to i}(\overline{y}_{\ge j+1}, \overline{x}_i)$
    $\tilde{\mu}_{\partial i\setminus j\to i}(\overline{y}_{\partial i\setminus j}, \overline{x}_i)$ = AggrMsg($\tilde{\mu}_{\le j-1\to i}(\overline{y}_{\le j-1}, \overline{x}_i)$, $\tilde{\mu}_{\ge j+1\to i}(\overline{y}_{\ge j+1}, \overline{x}_i)$)
    Compute $\mu_{i\to j}(\overline{x}_i, \overline{x}_j)$ with 3.19
end
```

3.4 BETHE FREE ENERGY

The joint probability distribution can be written in Boltzmann form (see Appendix D) as:

$$\mathbb{P}\left[\overline{\boldsymbol{x}}\right] = \frac{1}{Z} e^{-H(\overline{\boldsymbol{x}})} \tag{3.23}$$

and the factorized form of the probability distribution allows writing explicitly the Hamiltonian:

$$H(\overline{\boldsymbol{x}}) = \sum_{i \in \mathcal{V}} \log \Phi_i\left(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}\right) \tag{3.24}$$

As seen in section 3.1, the factor graph for MPBP has one factor for each node of $\mathcal{G}$ and one variable for each node of $\mathcal{G}$. At each point of the iterations of the BP equations, given a set of messages $\{\mu_{(i,j)\to i}, \mu_{i\to(i,j)}\}_{(i,j)\in\mathcal{E}, i\in\mathcal{V}}$, the Bethe free energy (the BP approximation of the real free energy) is:

$$F_{\text{Bethe}} = -\sum_{i\in\mathcal{V}} \log(z_i) - \sum_{(i,j)\in\mathcal{E}} -\log\left(z_{(i,j)}\right) - \sum_{i\in\mathcal{V}} \sum_{j\in\partial i} \log\left(z_{(i,j)\to i}\right) \tag{3.25}$$

With the normalizations that can be written as:

$$z_i = \sum_{\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}} \Phi_i\left(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}\right) \prod_{k\in\partial i} \mu_{k\to i}\left(\overline{x_k}, \overline{x}_i\right) \tag{3.26}$$

$$z_{(i,j)} = \sum_{\overline{x}_i, \overline{x}_j} \mu_{i\to j}\left(\overline{x}_i, \overline{x}_j\right) \mu_{j\to i}\left(\overline{x}_j, \overline{x}_i\right) \tag{3.27}$$

$$z_{(i,j)\to i} = 1 \tag{3.28}$$

By recalling the normalizations for messages from one factor to another 3.3:

$$z_{i\to j} := \sum_{\overline{x}_i, \overline{x}_j} \sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}} \Phi_i\left(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}\right) \prod_{k\in\partial i} \mu_{k\to i}\left(\overline{x}_k, \overline{x}_i\right) \tag{3.29}$$

it can be noticed that[9] $z_{(i,j)} = \frac{z_i}{z_{i\to j}}$, so the Bethe free energy can be rewritten as:

$$\begin{aligned} F_{\text{Bethe}} &= -\sum_{i\in\mathcal{V}} \log(z_i) + \sum_{(i,j)\in\mathcal{E}} \log\left(\frac{z_i}{z_{i\to j}}\right) = \\ &= -\sum_{i\in\mathcal{V}} \log(z_i) + \frac{1}{2} \sum_{i\in\mathcal{V}} \sum_{j\in\partial i} \left(\log(z_i) - \log\left(z_{i\to j}\right)\right) = \\ &= \sum_{i\in\mathcal{V}} \left[\left(\frac{|\partial i|}{2} - 1\right) \log(z_i) - \frac{1}{2} \sum_{j\in\partial i} \log\left(z_{i\to j}\right)\right] \end{aligned} \tag{3.30}$$

The latter quantities are computed at each iteration of MPBP, because, as discussed in section 3.2, tensor trains are always normalized during the calculations.

[9]From Equation 3.3 one has $z_{i\to j}\mu_{i\to j}\left(\overline{x}_i, \overline{x}_j\right) = \sum_{\overline{\boldsymbol{x}}_{\partial i\setminus j}} \Phi_i\left(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}\right) \prod_{k\in\partial i} \mu_{k\to i}\left(\overline{x}_k, \overline{x}_i\right)$, so, having in mind Equation 3.26 one can write $z_i = \sum_{\overline{x}_i, \overline{x}_j} z_{i\to j}\mu_{i\to j}\left(\overline{x}_i, \overline{x}_j\right) \mu_{j\to i}\left(\overline{x}_j, \overline{x}_i\right) = z_{i\to j} z_{(i,j)}$

4 Inference Procedure

Now it is time to put together all the previously explored models and techniques to describe how inference of hyperparameters can be performed starting from the observation of a dynamical process. As always, the SIS model will serve as a working example.

The specific problem, then, is to observe (some of) the time series of an epidemic diffusion and to reconstruct, based on such observations, the network of contacts and the infection probabilities.

First, a high-level formulation of the method will be provided and afterwards, all the detailed calculations and algorithms will be described.

4.1 High-level Description

As the network is not known, as well as the parameters, a fully-connected graph is considered, with each undirected edge and each node having their value of infection probability and recovery probability, respectively. The problem of inferring the network is thus incorporated into inferring the value of the probabilities, and then a threshold can be set to discriminate between the edges that are present and those that are absent. The whole method, based on the Bayesian framework described in section 1.1, can be reduced to the iteration of two alternated steps, in which the parameters are updated. The steps are repeated until convergence of the parameters or until a maximum number of iterations is reached. Suppose that at a certain iteration n the infection and recovery probabilities are $\left\{\lambda_{ij}^{(n)}\right\}_{(i,j)\in\mathscr{E}}$ and $\left\{\rho_{i}^{(n)}\right\}_{i\in\mathscr{V}}$. The first step consists in an iter-

ation of MPBP, in which the likelihood of the parameters $\mathscr{L}\left(\Theta^{(n)}\right)$ is calculated, with $\Theta^{(n)} = \left\{\lambda_{ij}^{(n)}\right\}_{(i,j)\in\mathscr{E}} \cup \left\{\rho_i^{(n)}\right\}_{i\in\mathscr{V}}$.
In the second step the parameters are updated by means of gradient ascent: the derivatives of the likelihood with respect to each individual parameter $\frac{\partial}{\partial\lambda_{ij}^{(n)}}\mathscr{L}\left(\Theta^{(n)}\right)$ and $\frac{\partial}{\partial\rho_i^{(n)}}\mathscr{L}\left(\Theta^{(n)}\right)$ are computed and the parameters themselves are updated according to some GA rule, for example, in the most naïve way, $\lambda_{ij}^{(n+1)} = \lambda_{ij}^{(n)} + \varepsilon\frac{\partial}{\partial\lambda_{ij}^{(n)}}\mathscr{L}\left(\Theta^{(n)}\right)$ and $\rho_i^{(n+1)} = \rho_i^{(n)} + \varepsilon\frac{\partial}{\partial\rho_i^{(n)}}\mathscr{L}\left(\Theta^{(n)}\right)$.
Here, partial derivatives of the likelihood are computed, while in general one should calculate the total derivative $\frac{\mathrm{d}}{\mathrm{d}\Theta^{(n)}}\mathscr{L}\left(\Theta^{(n)},\boldsymbol{\mu}\right) = \frac{\partial}{\partial\Theta^{(n)}}\mathscr{L}\left(\Theta^{(n)},\boldsymbol{\mu}\right) + \sum_\mu \frac{\partial}{\partial\mu}\mathscr{L}\left(\Theta^{(n)},\boldsymbol{\mu}\right)\frac{\mathrm{d}\mu}{\mathrm{d}\Theta^{(n)}}$, where μ are the messages. However, in the end, after all its iterations, the method should get to a fixed point of the messages and parameters, and at the fixed point for the messages the term $\frac{\partial}{\partial\mu}\mathscr{L}\left(\Theta^{(n)},\mu\right)$ vanishes. Because of this, it is reasonable to neglect the message-dependent term of the derivative in the first place and only consider the partial derivatives with respect to the parameters.
In the end, after N iterations, the method gives two outputs: the parameters $\Theta^{(N)}$ and the likelihood $\mathscr{L}\left(\Theta^{(N)}\right) = \mathbb{P}\left[\text{obs}|\Theta^{(n)}\right]$, from which the individual probabilities for the state of each node at any time can be derived.
For numerical stability one does not use the likelihood itself, but rather its logarithm $\log\mathscr{L}(\Theta) = -F(\Theta)$ and the result does not change as the logarithm is a monotonously increasing function.
An alternative to this procedure could be to run MPBP until convergence of the messages and then run GA until convergence. This would be exactly an approximated version of Expectation-Maximization, approximated because the loopy version of BP cannot provide the exact marginals. However, it is believed that making MPBP converge with the inexact parameters, or making GA converge with the inexact messages is a waste of time and it is much more efficient to perform the steps in an alternated fashion.

4.2 INITIALIZATION

First, in order to work properly, all the needed quantities must be defined. As seen in chapter 2, the SIS model is defined by means of a graph $\mathscr{G}_{\text{sis}} = (\mathscr{V}_{\text{sis}}, \mathscr{E}_{\text{sis}})$ and some parameters, namely the infection probabilities $\Lambda_{\text{sis}} = \left\{\lambda_{ij}\right\}_{(i,j)\in\mathscr{E}_{\text{sis}}}$ and the recovery probabilities $\mathrm{P}_{\text{sis}} = \left\{\rho_i\right\}_{i\in\mathscr{V}_{\text{sis}}}$.
With these and knowing the initial-time probabilities for each node in the graph, one

can simulate a trajectory and take some observations $\{\phi_i^t\}$ of the state of some nodes i at some times t.
Then one has to set up the framework to perform MPBP. The parameters, the object of the inference, are hidden, together with the graph $\mathscr{G}_{\text{sis}}$. A new graph $\mathscr{G} = (\mathscr{V} = \mathscr{V}_{\text{sis}}, \mathscr{E})$ is considered together with new parameters $\Lambda^{(0)}$ and $\mathrm{P}^{(0)}$ initialized in some random way[1] and, starting from this, a factor graph is created as done in section 3.1.
The observations are inserted into the model as reweighting factors for the probabilities of single node trajectories $\Phi_i(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i})$, as described in section 3.1.
In this framework, the graph and parameters $\mathscr{G}_{\text{sis}}$, Λ_{sis} and P_{sis} represent the ground truth, the true parameters and the true graph, while $\Lambda^{(n)}$ and $\mathrm{P}^{(n)}$ are the parameters to be updated and those that one tries to make the most similar possible to Λ_{sis} and P_{sis} by means of the inference procedure.

In the following, all the described procedures are to be performed iteratively, with parameters $\Lambda^{(n)}$ and $\mathrm{P}^{(n)}$ to be updated until convergence.

4.3 RECONSTRUCTION STEP

In the reconstruction step, the messages of MPBP are updated by means of the procedure described in section 3.3 and using the parameters $\Lambda^{(n)}$ and $\mathrm{P}^{(n)}$ in all computations. Besides the messages themselves, it is important at this stage to compute the normalizations z_i, which will be useful in the following.

4.4 CALCULATION OF DERIVATIVES

In order to perform Gradient Ascent, the derivatives of the log-likelihood are needed with respect to the parameters of the model:

$$\frac{\partial}{\partial \lambda_{ij}^{(n)}} \log \mathscr{L}\left(\Theta^{(n)}\right) \qquad \frac{\partial}{\partial \rho_i^{(n)}} \log \mathscr{L}\left(\Theta^{(n)}\right) \tag{4.1}$$

Actually, as described in section 1.1, the problem is translated into a variational one, with a variational free energy (here represented by the Bethe free energy) being minimized.

[1]The graph $\mathscr{G}$ can be whatever, as long as $\mathscr{V} = \mathscr{V}_{\text{sis}}$. In case of complete ignorance of the system, it can be a complete one, but it can also be the original graph with some added edges (false positives) that the inference procedure attempts to remove. In any case, it should contain more edges than the original one, because the inference method can find false positives but not false negatives.

Thus, by calling $p(\overline{\boldsymbol{x}})$ the original probability distribution for the system trajectories, obtained with the true parameters, and $q(\overline{\boldsymbol{x}})$ the one obtained by the parameters $\Lambda^{(n)}$ and $\mathrm{P}^{(n)}$ and with the messages not yet converged, the free energy can be written as:

$$F[q,\Theta] = \langle H_\Theta \rangle_q - S(q) \tag{4.2}$$

Where S is the entropy of the distribution q and H_Θ is the energy, i.e. the logarithm of the Boltzmann weights (see Appendix D). At this level, it is clear how the inference problem consists of two optimizations: The iteration of MPBP provides the most accurate messages to describe the distribution *with the current parameters*, while the gradient ascent step adjusts the parameters *according to the current messages*.
It is clear that only the first part of the free energy depends explicitly on the parameters Θ, so the entropic contribution can be ignored when computing the derivatives.
In the case of the Bethe free energy 3.4 the energetic contribution is represented by the part with the $\log z_i$'s, while the entropic part contains the normalizations of the messages $\log z_{(i,j)\to i}$ and of the two-variable nodes $\log z_{(i,j)}$.
Then the sought quantities become:

$$\frac{\partial}{\partial \lambda_{ij}^{(n)}} \sum_{i\in\mathcal{V}} \log(z_i) \qquad \frac{\partial}{\partial \rho_i^{(n)}} \sum_{i\in\mathcal{V}} \log(z_i) \tag{4.3}$$

First, it can be noticed that each derivative involves only one $\log(z_i)$ term. Indeed, the parameter ρ_i is present only in the term $\log(z_i)$, as it describes the recovery probability of node i, while the parameter λ_{ij} is present only in the term $\log\left(z_j\right)$, as it is associated of a change in the state of node j.
In general, each term $\log(z_i)$ depends on the parameter ρ_i and on all the λ_{ki} with $k \in \partial i$.
It is important to notice that here the considered graph is always $\mathcal{G}$, not the true one $\mathcal{G}_{\text{sis}}$.
Reminding Equation 3.26, the derivatives can be written as:

$$\frac{\partial}{\partial\theta}\log(z_i) = \frac{1}{z_i}\frac{\partial z_i}{\partial\theta} = \frac{1}{z_i}\sum_{\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}} \frac{\partial}{\partial\theta}\Phi_i\left(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}\right)\prod_{j\in\partial i}\mu_{j\to i}\left(\overline{x}_j,\overline{x}_i\right) \tag{4.4}$$

and then what is left to compute is $\frac{\partial}{\partial\theta}\Phi_i\left(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}\right)$. As shown in Appendix E one has:

$$\frac{\partial}{\partial\lambda_{ji}}\Phi_i\left(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}\right)=w_i^0\left(x_i^0\right)\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left[\left(\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{k\in\partial i\setminus\{j\}}\prod_{t=0}^{T-1}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\times\right.$$
$$\left.\times\sum_{s=0}^{T-1}\left(\mathbb{I}\left[x_j^s=I\right]\left(\mathbb{I}\left[y_j^s=I\right]-\mathbb{I}\left[y_j^s=S\right]\right)\prod_{t\neq s}\mathbb{P}\left[y_j^t|x_j^t,x_i^t\right]\right)\right] \tag{4.5}$$

$$\frac{\partial}{\partial\rho_i}\Phi_i\left(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}\right)=w_i^0\left(x_i^0\right)\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left[\sum_{s=0}^{T-1}\left(\mathbb{I}\left[x_i^s=I\right]\left(\mathbb{I}\left[x_i^{s+1}=S\right]-\mathbb{I}\left[x_i^{s+1}=I\right]\right)\times\right.\right.$$
$$\left.\left.\times\prod_{t\neq s}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\right)\left(\prod_{t=0}^{T-1}\prod_{k\in\partial i}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\right] \tag{4.6}$$

And then, inserting into Equation 4.4:

$$\frac{\partial}{\partial\lambda_{ji}}\log(z_i)=\frac{1}{z_i}\sum_{s=0}^{T-1}\sum_{\{\overline{y}_k\}_{k\in\partial i}}\sum_{\overline{x}_i,\overline{x}_j}w_i^0\left(x_i^0\right)\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{t\neq s}\mathbb{P}\left[y_j^t|x_j^t,x_i^t\right]\mu_{j\to i}\left(\overline{x}_j,\overline{x}_i\right)\times$$
$$\times\mathbb{I}\left[x_j^s=I\right]\left(\mathbb{I}\left[y_j^s=I\right]-\mathbb{I}\left[y_j^s=S\right]\right)\tilde{\mu}_{\partial i\setminus\{j\}\to i}\left(\overline{y}_{\partial i\setminus\{j\}},\overline{x}_i\right) \tag{4.7}$$

$$\frac{\partial}{\partial\rho_i}\log(z_i)=\frac{1}{z_i}\sum_{s=0}^{T-1}\sum_{\{\overline{y}_k\}_{k\in\partial i}}\sum_{\overline{x}_i}\mathbb{I}\left[x_i^s=I\right]\left(\mathbb{I}\left[x_i^{s+1}=S\right]-\mathbb{I}\left[x_i^{s+1}=I\right]\right)w_i^0\left(x_i^0\right)\times$$
$$\times\prod_{t\neq s}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\tilde{\mu}_{\partial i\to i}\left(\overline{y}_{\partial i},\overline{x}_i\right) \tag{4.8}$$

It can be noticed that these are sums over s of T terms, each one being very similar to the normalization of the message in Equation 3.19. The only differences are:

- For $\frac{\partial}{\partial\lambda_{ji}}\log(z_i)$, instead of the regular $\mathbb{P}\left[y_j^t|x_j^t,x_i^t\right]$, for $t=s$ one has the term $\mathbb{I}\left[x_j^s=I\right]\left(\mathbb{I}\left[y_j^s=I\right]-\mathbb{I}\left[y_j^s=S\right]\right)$.
- For $\frac{\partial}{\partial\rho_i}\log(z_i)$, instead of the regular $\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]$, for $t=s$ one has the term $\mathbb{I}\left[x_i^s=I\right]\left(\mathbb{I}\left[x_i^{s+1}=S\right]-\mathbb{I}\left[x_i^{s+1}=I\right]\right)$.

Given this, the same algorithm used to calculate the messages and described in section 3.3 can be used to compute the derivatives in a fast way, provided that one makes the right substitutions.
The algorithm obtained in this way would have $\mathcal{O}\left(T^2\right)$ complexity, because it involves a sum over s, which acquires T values and a product over t, which acquires $T-1$ values. However, this computation is inefficient, as it computes the same quantities several times

(the terms of the s sum are almost equal, except for one single factor). Then, avoiding these duplicate computations makes the algorithm $\mathcal{O}(T)$.
To do this, some additional quantities have to be defined (here the case of $\frac{\partial}{\partial\lambda_{ji}}$ is shown):

$$L^t\left(x_i^{0:t-1},x_j^{0:t-1},x_i^t\right) := \prod_{p=0}^{t-1}\sum_{y_j^p}\mathbb{P}\left[x_i^{p+1}|y_j^p,x_i^p\right]\tilde{A}^p_{j\to i}\left(y_j^p,x_i^p\right) \tag{4.9}$$

$$R^t\left(x_i^{t:T-1},x_j^{t:T-1},x_i^T\right) := \prod_{p=t}^{T-1}\sum_{y_j^p}\mathbb{P}\left[x_i^{p+1}|y_j^p,x_i^p\right]\tilde{A}^p_{j\to i}\left(y_j^p,x_i^p\right) \tag{4.10}$$

where the factorization of the message $\mu_{j\to i}\left(\overline{x}_j,\overline{x}_i\right)$ is used. The quantities L^t and R^t are essentially analogous to the messages computed in section 3.3, but they just contain some of the time factors.
Recursively, all the L^t and R^t quantities can be computed as:

$$L^{t+1}\left(x_i^{0:t},x_j^{0:t},x_i^{t+1}\right) = L^t\left(x_i^{0:t-1},x_j^{0:t-1},x_i^t\right)\times\sum_{y_j^t}\mathbb{P}\left[x_i^{t+1}|y_j^t,x_i^t\right]\tilde{A}^t_{j\to i}\left(y_j^t,x_i^t\right) \tag{4.11}$$

$$R^{t-1}\left(x_i^{t-1:T-1},x_j^{t-1:T-1},x_i^T\right) = \sum_{y_j^{t-1}}\mathbb{P}\left[x_i^t|y_j^{t-1},x_i^{t-1}\right]\tilde{A}^{t-1}_{j\to i}\left(y_j^{t-1},x_i^{t-1}\right)\times R^t\left(x_i^{t:T-1},x_j^{t:T-1},x_i^T\right) \tag{4.12}$$

with initial conditions:

$$L^0\left(x_i^0\right) = 1 \tag{4.13}$$

$$R^T\left(x_i^T\right) = 1 \tag{4.14}$$

In the end, the s^{th} term of the sum is calculated as:

$$\sum_{\overline{x}_i,\overline{x}_j} w_i^0\left(x_i^0\right)\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\left\{y_k^t\right\}_{k\in\partial i},x_i^t\right]\tilde{\mu}_{\partial i\setminus\{j\}\to i}\left(\overline{y}_{\partial i\setminus\{j\}},\overline{x}_i\right)\times$$
$$\times L^s\left(x_i^{0:s-1},x_j^{0:s-1},s_i^s\right)M^s\left(x_i^s,x_j^s\right)R^s\left(x_i^{s:T-1},x_j^{s:T-1},x_i^T\right) \tag{4.15}$$

with $M^s\left(x_i^s,x_j^s\right) := \sum_{y_j^s}\mathbb{I}\left[x_j^s = I\right]\left(\mathbb{I}\left[y_j^s = I\right] - \mathbb{I}\left[y_j^s = S\right]\right)A^s_{j\to i}\left(x_j^s,x_i^s\right)$. At this point the procedure described in section 3.3 can be used to perform the calculation.
An analogous (actually easier) procedure can be carried out for the derivative with respect to ρ_i.

The algorithm is manifestly linear in T, as the precomputation of all the L^s and R^s are done recursively and are thus linear in T and, thanks to this, the terms of the sum over s in the derivatives are already computed. The Julia code that implements all these computations is available at [26].

5 Results

The algorithm developed and explained in chapter 4 has been tested in different situations and to obtain different results. Overall, it showed good performance, both in terms of accuracy and running time. The following sections will explore the tests and the gathered results.

5.1 Inference of Uniform Parameters

The first task the algorithm was tested on concerns the inference of the infection parameters when the graph is known. This task, although relatively easy compared to inferring the network topology, is of key importance in real-world applications.
The model is slightly different to that with heterogeneous infection and recovery rates. Referring to the formalism developed in section 4.2 one has:

$$\begin{cases} \lambda_{ij} = \lambda & \forall (i,j) \in \mathscr{E}_{\text{sis}} \\ \rho_i = \rho & \forall i \in \mathscr{V}_{\text{sis}} \end{cases} \tag{5.1}$$

and the graph used for inference is actually the same used for the definition of the SIS model:

$$\mathscr{G} = \mathscr{G}_{\text{sis}} \tag{5.2}$$

In order to be able to infer the parameters λ and ρ, one has to compute the derivative of the likelihood with respect to them. This is actually easy, because enforcing the equality between all the parameters and exploiting the chain rule of derivation one has:

$$\frac{\partial}{\partial \lambda}\log \mathbb{P}\left[\overline{\boldsymbol{x}}|\Theta\right] = \sum_{(i,j)\in\mathscr{E}} \frac{\partial}{\partial \lambda_{ij}}\log \mathbb{P}\left[\overline{\boldsymbol{x}}|\Theta\right]\frac{\mathrm{d}\lambda_{ij}}{\mathrm{d}\lambda} = \sum_{(i,j)\in\mathscr{E}} \frac{\partial}{\partial \lambda_{ij}}\log \mathbb{P}\left[\overline{\boldsymbol{x}}|\Theta\right] \tag{5.3}$$

$$\frac{\partial}{\partial \rho}\log \mathbb{P}\left[\overline{\boldsymbol{x}}|\Theta\right] = \sum_{i\in\mathscr{V}} \frac{\partial}{\partial \rho_{i}}\log \mathbb{P}\left[\overline{\boldsymbol{x}}|\Theta\right]\frac{\mathrm{d}\rho_{i}}{\mathrm{d}\rho} = \sum_{i\in\mathscr{V}} \frac{\partial}{\partial \rho_{i}}\log \mathbb{P}\left[\overline{\boldsymbol{x}}|\Theta\right] \tag{5.4}$$

Thus, the derivatives with respect to the uniform parameters λ and ρ are just the sums of the derivatives with respect to the edge- or node-dependent parameters.
In the tests, the graph was chosen to be an Erdős-Rényi random graph[1] with $N = 30$ nodes and average connectivity $c = 2.5$. The observations are taken as snapshots of the whole system every $k = 6$ times. The true infection parameters are $\lambda_{\text{true}} = 0.05$ and $\rho_{\text{true}} = 0.05$.
First, the free energy landscape was reconstructed as a function of the parameters λ and ρ to be inferred. This is shown in Figure 5.1. Then, the inference algorithm was run from

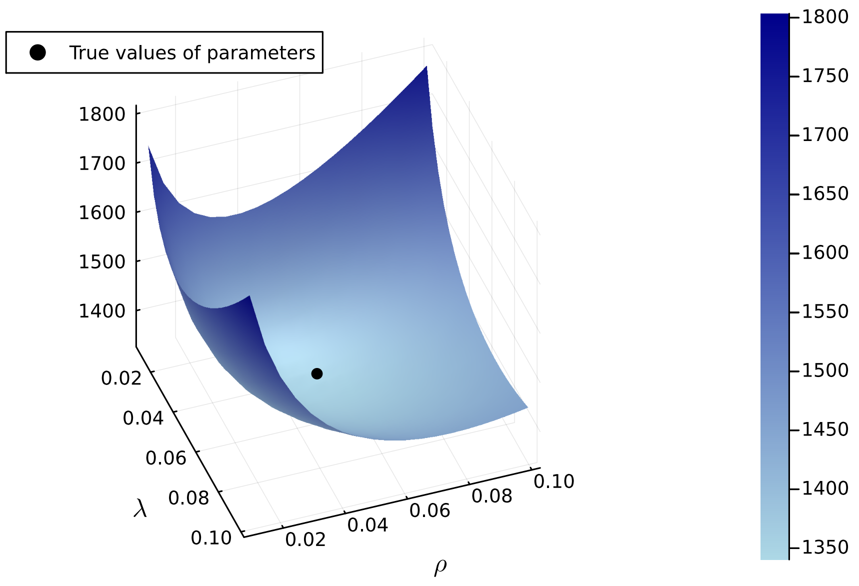

FIGURE 5.1. Free energy as a function of the parameters λ and ρ, calculated on a grid of dimension 19×19. The minimum of the free energy, with this discretization, corresponds to the most likely parameters.

different starting points and it converged to rather accurate parameters, as shown in Table 5.1. If the free energy is calculated in the true parameters and in the inferred ones,

[1]In particular, the $\mathscr{G}(N,p)$ model is employed, in which each potential edge between a pair of nodes is present with a probability $p = c/N$, independently from the others.

Starting parameters	$\lambda = 0.01$ $\rho = 0.01$	$\lambda = 0.03$ $\rho = 0.09$	$\lambda = 0.07$ $\rho = 0.01$	$\lambda = 0.1$ $\rho = 0.1$
Parameters at convergence	$\lambda = 0.04821$ $\rho = 0.04720$	$\lambda = 0.04828$ $\rho = 0.04712$	$\lambda = 0.04827$ $\rho = 0.04714$	$\lambda = 0.04827$ $\rho = 0.04714$

TABLE 5.1. Inferred parameters from different starting conditions.

the results are $F(\lambda = 0.048, \rho = 0.047) = 1339.82$ and $F(\lambda_{\text{true}}, \rho_{\text{true}}) = 1340.33$, showing that the free energy is actually lower for the inferred parameters than for the true ones (and thus the likelihood is higher for the inferred parameters than for the true ones).

5.2 INFERENCE OF THE GRAPH

The primary and most ambitious objective of the inference method is to reconstruct the topology of the network of interactions. In order to do this, the algorithm is left free to learn all the parameters, and the learned values for the λ_{ij}s are used as a score for the edges.

The algorithm tried to reconstruct the well-known Zachary's karate club network [27] starting from an SIS dynamics with $\lambda = 0.05$ and $\rho = 0.07$. The observations were taken as $n_{\text{snaps}} = 300$ snapshots of the whole system every $k = 4$ times. Thus, the output of

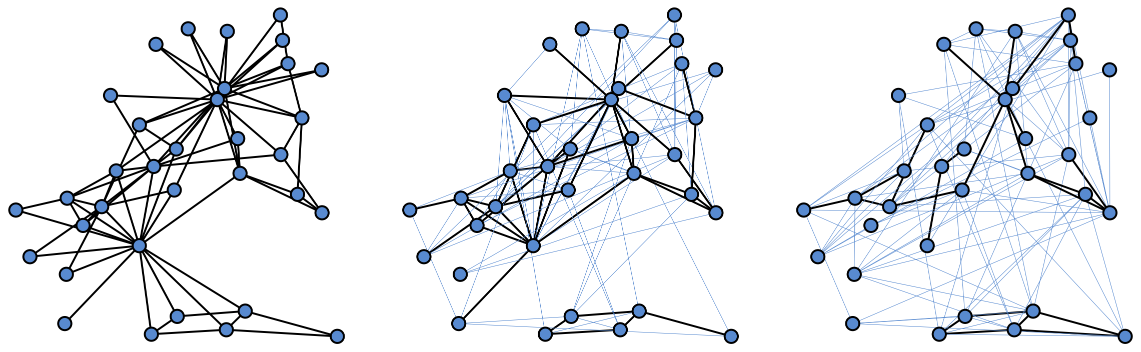

FIGURE 5.2. True karate club network (left) and top-10% scoring edges with the inference method (middle) and with just correlations (right). The true edges are drawn with a thick line, while the ones that are not present in the original graph are drawn with a thin line.

the algorithm is one λ_{ij} parameter for each potential edge of the graph (i.e. for each pair of nodes i and j). Figure 5.2 shows the 10% of edges with higher inferred λ_{ij}. The algorithm is compared to a naïve method that assigns edge scores by simply computing the correlations between pairs of observations of single-node trajectories. As shown by the figure, the algorithm is able to find much more of the true edges than the naïve method. In particular, the inference technique finds 44 of the 78 true edges, while the correlations are able to find just 24 of them. This is a remarkable accuracy, and it is believed that the algorithm can perform even better if provided with more information.

5.3 STRATEGIES FOR PERFORMANCE

As previously seen, the algorithm ensures good reconstruction accuracy and rather good performance, being linear in the time horizon T of the process (chapter 4). Despite this, the update of the messages and the calculation of the derivatives must be performed for each and every possible edge in the graph $\mathcal{G}$. If no a priori information is available on the candidate edges, their number is $N(N-1)$, where N is the number of nodes. This means that the algorithm is of quadratic order in the number of nodes, making it rather slow for moderately large networks.
If the graph is known to be sparse, as is the case in most real-world applications, one possible solution to this problem could be to try and rank the nodes with some naïve procedure and then retain only the highest-ranked ones for running the MPBP inference. This has been tried by computing some measure of the "similarity" between pairs of single-node observations and using them to rank the possible edges. Two candidate measures, sufficiently easy to calculate, were the correlation and the mutual information. Correlation is a standard metric in statistics, being the normalized version of the covariance of two random variables. The mutual information, instead, is a measure of mutual dependence between two random variables, mostly used in information theory. For random variables x and y, distributed according to p_x and p_y, and with joint distribution p_{xy}, they can be calculated as:

$$\mathrm{Corr}(x,y) = \frac{\mathrm{Cov}(x,y)}{\sqrt{\mathrm{Var}[x]\mathrm{Var}[y]}} = \frac{\mathbb{E}[xy]-\mathbb{E}[x]\mathbb{E}[y]}{\sqrt{\mathrm{Var}[x]\mathrm{Var}[y]}} \tag{5.5}$$

$$\mathrm{MI}(x,y) = \sum_{x,y} p_{xy}(x,y)\log\left(\frac{p_{xy}(x,y)}{p_x(x)p_y(y)}\right) \tag{5.6}$$

In order to assess which of the two measures was best, the area under the ROC curve (see Appendix F) was first taken into account. For each pair of nodes, the correlation and the mutual information between the observations of the single-node trajectories were computed, and all the possible edges were ranked according to those two measures. Figure 5.3 shows the AUC in different scenarios. The correlations seem to rank the edges

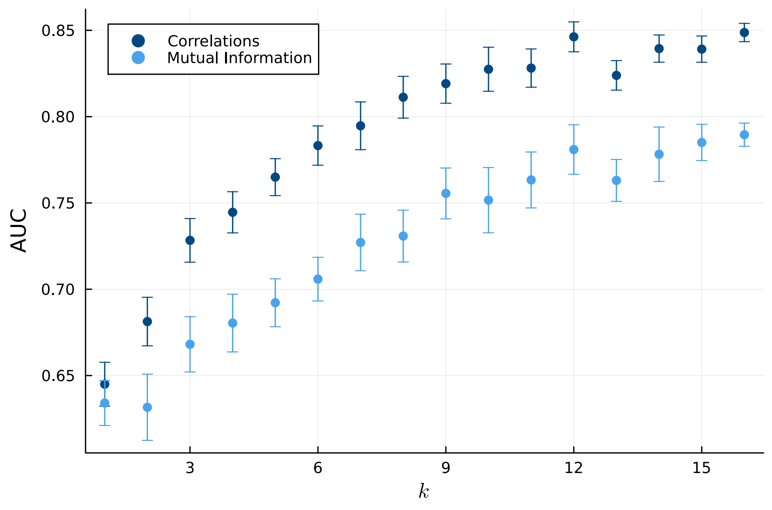

FIGURE 5.3. Area under the ROC curve, with edges ranked according to the correlations and mutual information. Results refer to an Erdős-Rényi random graph with $N = 30$ nodes, average connectivity $c = 2.5$, infection probability $\lambda = 0.05$ and recovery probability $\rho = 0.05$. The observations were taken as snapshots of the whole system every k times.

better than the mutual information.

However, as explained previously, the objective of this analysis is to perform a pre-screening of all the possible edges of a graph, in order to run the inference algorithm on a more diluted graph. The inference algorithm can not give good results if some “real” edges are discarded by the pre-screening, so one should try to include as many of the real edges as possible, without taking too many edges in total. For this reason, the following procedure was defined for the pre-screening:

- For each node, n_{neigs} edges are taken connecting it to the n_{neigs} most correlated nodes.
- Then, the most correlated pairs of nodes are connected, until the total number of $N \cdot n_{max}$ edges is reached.

Figure 5.4 shows the fraction of the original edges found by the pre-screening as a function of n_{neigs} and n_{max}, both in the case of correlations and mutual information. The analysis seems to show that, for correlations, a good fraction of the original edges (i.e.

more than 90%) is found when retaining $(8 \div 10)N$ edges of the original graph. Moreover, the results seem to be independent of n_{neigs}. Mutual information is confirmed to be worse than correlations for this scope.

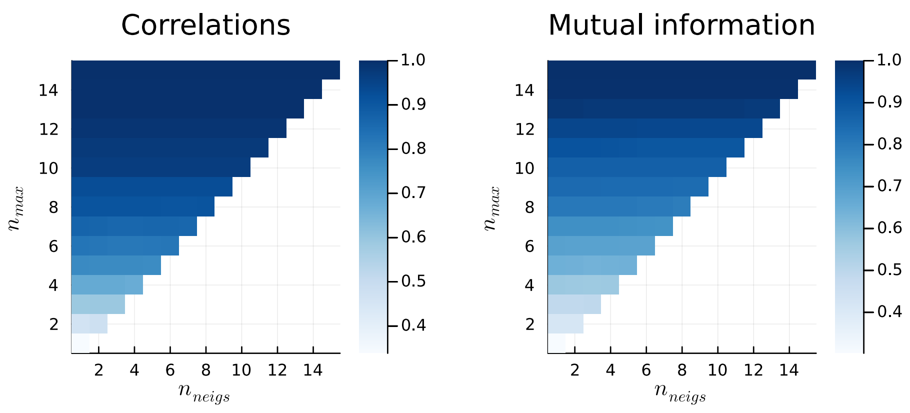

FIGURE 5.4. Fraction of the original edges that were retained by the pre-screening, as a function of n_{neigs} and n_{max}. The settings are the same as for Figure 5.3, but with fixed $k = 6$.

5.4 INFERENCE WITH PRE-SCREENING

As seen in section 5.3, a pre-screening procedure can be used to make computations faster. This, in turn, allows to calculate some statistics, in order to assess the accuracy of the inference algorithm.

In particular, it can be interesting to investigate the behaviour of the method as a function of the observation timescale. A system undergoing a dynamical process, indeed, changes with a speed that is related to its governing parameters. For example, for an SIS model, the dynamic is faster the more connected the graph and the higher the infection and recovery probabilities.

This also affects the inference procedure: for the same number of observations, one can expect that, if these are taken in a small time window, no major changes in the system are observed and the observations, to some extent, are "wasted". Similarly, one could argue that, if the observations are too sparse in time, they are not correlated to each other and the amount of extracted information is equally low.

To inquire about the behaviour of the inference method in this respect, it was run after having simulated some trajectories and having taken snapshots of the whole system every k times, with k ranging from 1 to 16. Moreover, the initial graphs were

decimated by means of the procedure explained in section 5.3, retaining the $6N$ most correlated neighbours. The resulting ROC areas are shown in Figure 5.5. The inset shows the correlations between two consecutive observations, a measure of the amount of information that the observations bring. It can be clearly seen that the reconstruction is better, the bigger the time between observations. Actually, for small k, as shown in the inset, the observations are highly correlated, so almost no dynamic is observed.
However, it can be noticed that, even for $k = 16$, consecutive snapshots are still correlated. Then, another simulation was run with higher infection and recovery probability, in order to obtain trajectories with shorter characteristic timescales. The results are shown in Figure 5.6.
As clearly visible, when consecutive snapshots are not correlated anymore, running the inference algorithm has the same accuracy as simply evaluating correlations. This is intuitive, as observing the system in times that are too separated obviously tells nothing about the causal relations between different nodes. The only useful information that can be gathered in this regime is the simultaneous state of two nodes. If, for example, nodes i and j are observed several times to be both in the infectious state, it is likely that one infects the other, hence the edge (ij) is probably present in the original graph.

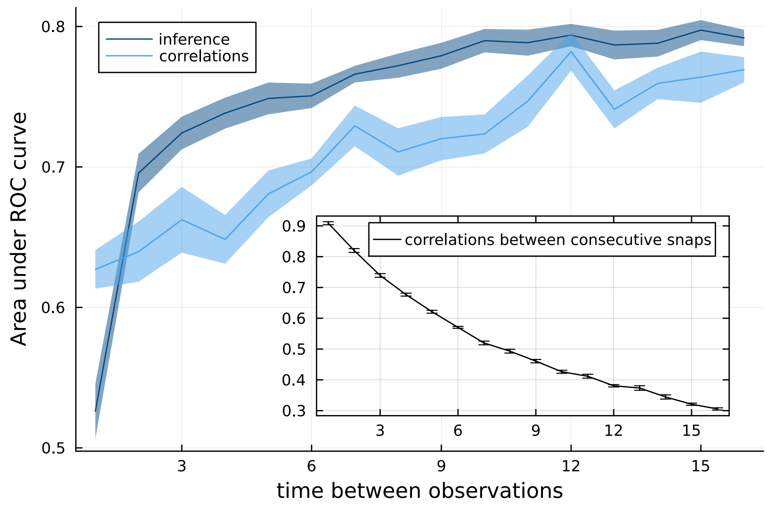

FIGURE 5.5. Area under the ROC curve for the inference method, compared with that obtained just from correlations. The results refer to an Erdős-Rényi random graph with $N = 30$ nodes, average connectivity $c = 2.5$, infection probability $\lambda = 0.05$ and recovery probability $\rho = 0.05$.

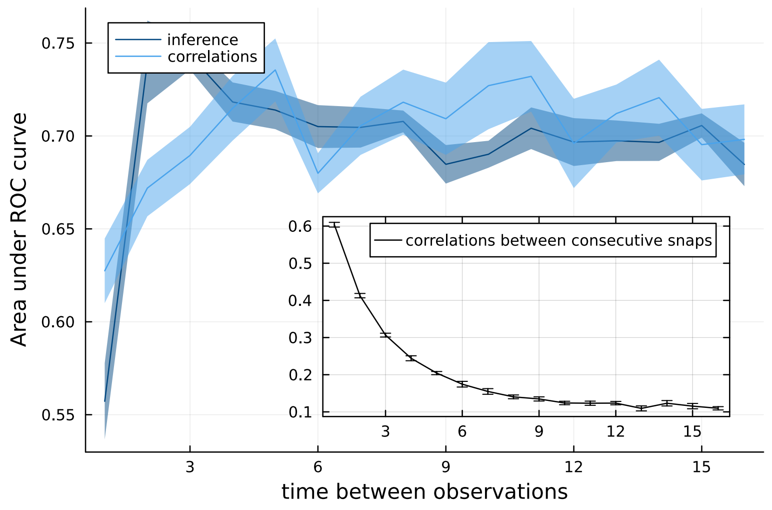

FIGURE 5.6. Area under the ROC curve for the inference method, compared with that obtained just from correlations. The results refer to an Erdős-Rényi random graph with $N = 30$ nodes, average connectivity $c = 2.5$, infection probability $\lambda = 0.2$ and recovery probability $\rho = 0.2$.

6 Conclusion

This work explored a new method that allows to perform inference of parameters and hyperparameters in a class of Markov processes: those occurring on a graph and in which the update probability for the state of a certain node depends only on the state of neighbouring nodes.
This class is wide and includes models that are key to studying several real-world phenomena, such as the SIS epidemic model, the voter model or the Glauber dynamics for the Ising model.
The inference procedure is based on two previously developed techniques coming from different fields of Physics, namely the BP algorithm and the Tensor-Train decomposition, which together give rise to the MPBP algorithm, defined in [21].
In particular, the SIS epidemic model has been the working example throughout this entire thesis, offering a case study whose complexity was enough to render it intractable before, but at the same time simple enough to be used to represent several and diverse real-world systems.
As shown in chapter 4, the method has linear computational complexity both in the time horizon of the dynamics and in the number of possible edges of the graph one is dealing with. This makes it already suitable to be used, even if some optimization can be made.
A Julia code [26] has been developed and tested, with the obtained results described in chapter 5.
These show that the developed method performs fairly well. In some regimes, it was tested against some naïve methods, that were before the best possibility one could hope for, and it always performed better. In addition, this novel procedure can be used also in

other cases, for which no attempt could be made in the past.
One of the main strengths of the method is its remarkable ability to infer the parameters of the models and the topology of the graph even when the observations are sparse in time, meaning that it does not have access to the infection and recovery events (i.e. it does not really observe the change in the state of a node).

6.1 POSSIBLE APPLICATIONS

The great power of this novel method lies in its ability to reconstruct networks, based only on observations of the time series of the stochastic process. Although the first possible application that comes to mind, in these cases, is the reconstruction of contacts in large epidemic events such as COVID-19, the power of the method should not be overestimated. Throughout this work, it has been tested when a decent amount of information (of the order of a few hundred observations per node) was available, making it unsuitable for cases where one can rely on very few and sparse observations, like nation or world scale epidemics.
However, there are some real-world scenarios where a frequent and accurate observation of the individuals is available. For example, staff and patients in hospital intensive care units are routinely tested for the most common infectious diseases, to limit their spread. This provides a particularly suitable playground for the algorithm presented here, similar to the simulations conducted so far. In the future, it could be very helpful in providing a reliable tool for addressing the spread of diseases in such delicate situations.
Moreover, one should not forget the great ability the algorithm has proven in descending the free-energy profile and inferring the parameters of the model when the graph is known. As shown in section 5.1, the outputs of the inference were really close to the real values of the parameters. This is a really powerful feature that can be very useful in real-world cases, even big epidemic events.

6.2 FURTHER WORK

As already mentioned, the inference method presented here is general, and its applications go far beyond the SIS and related epidemic spreading models. Indeed, one of the first generalizations that should be made is the application of the same techniques to the treatment of other models.
For example, the Glauber dynamics for Ising model could be the next one to be considered,

as it is very general and can describe several real-world phenomena, such as neural cascades or magnetism. However, some technical difficulties should be overcome before an algorithm like the one presented in this work can be developed.
Besides this, the algorithm for the SIS model presented here is far from being really understood. The present analysis has only scratched the surface of examining its behaviour in different regimes and with different graphs, as well as the scalability of its results. A future in-depth analysis could try and inspect the performance of the algorithm with respect to the gradient ascent policy, for example, or the sparsity and type of the considered graph. All this could shed more light both on the inference method itself and the underlying models.

Appendix

A Probabilistic Graphical Models

Take a probability distribution of the factorized form[1] $\mathbb{P}[\boldsymbol{x}] = \frac{1}{Z}\prod_{a\in A} f_a(\boldsymbol{x}_a)$. Here, a are subsets of indices such that f_a depends only on the variables with index $i \in a \subseteq I$.

This can be easily represented on a bipartite graph, called a factor graph, which has a *variable node* i for each variable x_i and a *factor node* a for each factor f_a. Each variable node is connected to a factor node if and only if $i \in a$, that is if f_a depends on x_i. In mathematical terms, a factor graph is defined as:

$$\mathcal{G} = (\mathcal{V} = A \cup I, \mathcal{E} = \{(ia) : f_a \text{ depends on } x_i\}) \tag{A.1}$$

An example can be found in Figure A.1.

Factor graphs are useful because their properties reflect the properties of probability and because they often allow to perform some otherwise difficult calculations by means of suitable algorithms. As an example, they permit the calculation of single-node marginals by means of the BP algorithm. Particularly relevant is the following theorem, which specifies how to reconstruct a probability distribution given its marginals (for a proof see [28], Theorem 14.2):

[1]Actually, every probability distribution function admits a factorized representation. For example, all probability distributions can be thought of as a naïve product of a single factor.

Theorem A.1. *Given an acyclic factor graph $\mathcal{G} = (I \cup A, \mathcal{E})$ and a probability distribution $p(\boldsymbol{x}) = \frac{1}{Z}\prod_a f_a(\boldsymbol{x}_a)$, with marginals defined by $p(x_i) = \sum_{\boldsymbol{x}_{-i}} p(\boldsymbol{x})$ and $p(\boldsymbol{x}_a) = \sum_{x_j : j \notin \partial a} p(\boldsymbol{x})$, it holds that:*

$$p(\boldsymbol{x}) = \prod_{a \in A} p(\boldsymbol{x}_a) \prod_{i \in I} p(x_i)^{1-|\partial i|} = \prod_{a \in A} \frac{p(\boldsymbol{x}_a)}{\prod_{i \in \partial a} p(x_i)} \prod_{i \in I} p(x_i)$$

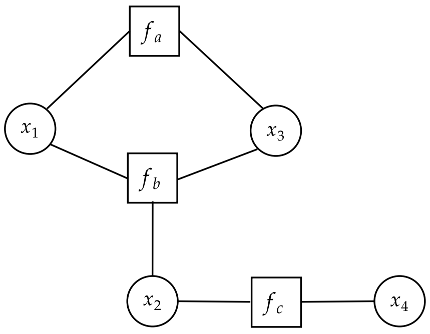

FIGURE A.1. Example of a factor graph, where the variable nodes are represented by circles and the factor nodes are represented by squares. The graph above represents the probability distribution $p(x_1, x_2, x_3, x_4) = f_a(x_1, x_3) f_b(x_1, x_2, x_3) f_c(x_2, x_4)$

B

DERIVATION OF THE BP EQUATIONS

Given a factorized probability distribution $\mathbb{P}[\boldsymbol{x}] = \frac{1}{z}\prod_a f_a(\boldsymbol{x}_a)$ and its factor graph $\mathscr{G} = (\mathscr{V} = A \cup I, \mathscr{E})$, if such graph is acyclic, one can compute single-variable marginals by means of Belief Propagation.

As an example, suppose one is interested in calculating the marginal $\mathbb{P}[x_i]$. Then, the factor graph can be divided as in Figure B.1, taking the node i as the root.

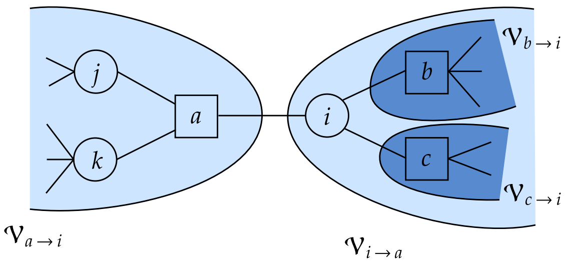

FIGURE B.1. Taking node i as root, the branches going to factors a, b and c are called respectively $\mathscr{V}_{a\to i} = I_{a\to i} \cup A_{a\to i}$, $\mathscr{V}_{b\to i} = I_{b\to i} \cup A_{b\to i}$ and $\mathscr{V}_{c\to i} = I_{c\to i} \cup A_{c\to i}$. An analogous division can be performed by taking a factor node as root.

Then one has:

$$\begin{aligned}\mathbb{P}[x_i] = \sum_{\boldsymbol{x}_{-i}} \mathbb{P}[\boldsymbol{x}] \propto \sum_{\boldsymbol{x}_{-i}} \prod_{a} f_a(\boldsymbol{x}_a) \propto \\ \propto \sum_{\boldsymbol{x}_{-i}} \prod_{a \in \partial i} \prod_{b \in A_{a \to i}} f_b(\boldsymbol{x}_b) \propto \\ \propto \prod_{a \in \partial i} \sum_{\boldsymbol{x}_{I_{a \to i}}} \prod_{b \in A_{a \to i}} f_b(\boldsymbol{x}_b) \propto \\ \propto \prod_{a \in \partial i} \mu_{a \to i}(x_i)\end{aligned} \tag{B.1}$$

where $\mu_{a \to i}(x_i) \propto \sum_{\boldsymbol{x}_{I_{a \to i}}} \prod_{b \in A_{a \to i}} f_b(\boldsymbol{x}_b)$. Then the update rules Equation 2.4 and 2.5 and the equation for beliefs 2.6 can be straightforwardly derived:

$$\begin{aligned}\mu_{a \to i}(x_i) \propto \sum_{\boldsymbol{x}_{I_{a \to i}}} \prod_{b \in A_{a \to i}} f_b(\boldsymbol{x}_b) \propto \sum_{\boldsymbol{x}_{I_{a \to i}}} f_a(\boldsymbol{x}_a) \prod_{j \in \partial a \setminus i} \prod_{c \in A_{j \to a}} f_c(\boldsymbol{x}_c) \propto \\ \propto \sum_{\boldsymbol{x}_{a \setminus i}} f_a(\boldsymbol{x}_a) \prod_{j \in \partial a \setminus i} \sum_{\boldsymbol{x}_{I_{j \to a} \setminus j}} \prod_{c \in A_{j \to a}} f_c(\boldsymbol{x}_c) \propto \sum_{\boldsymbol{x}_{a \setminus i}} f_a(\boldsymbol{x}_a) \prod_{j \in \partial a \setminus i} \mu_{j \to a}(x_j)\end{aligned} \tag{B.2}$$

$$\mu_{j \to a}(x_j) \propto \sum_{\boldsymbol{x}_{I_{j \to a} \setminus j}} \prod_{c \in A_{j \to a}} f_c(\boldsymbol{x}_c) \propto \prod_{b \in \partial j \setminus a} \sum_{\boldsymbol{x}_{I_{b \to j}}} \prod_{c \in A_{b \to j}} f_b(\boldsymbol{x}_b) \propto \prod_{b \in \partial j \setminus a} \mu_{b \to j}(x_j) \tag{B.3}$$

$$b_i(x_i) \propto \prod_{a \in \partial i} \mu_{a \to i}(x_i) \tag{B.4}$$

C

Singular-Value Decomposition

In linear algebra, Singular-Value Decomposition is a factorization of a matrix that generalizes the eigendecomposition of square normal matrices to any matrix. It consists in writing a matrix $A \in \mathbb{R}^{m\times n}$ as a product of three matrices $A = U\Lambda V^\dagger$, with $U \in \mathbb{R}^{m\times m}$, $\Lambda \in \mathbb{R}^{m\times n}$ and $V \in \mathbb{R}^{n\times n}$.
In particular, $U^\dagger U = VV^\dagger = \mathbb{1}$ and Λ is diagonal. The diagonal entries $\lambda_k := \Lambda_{kk}$ are known as the singular values of A and they are uniquely determined by it. By calling $\boldsymbol{u}_1,\dots,\boldsymbol{u}_m$ the columns of U and $\boldsymbol{v}_1,\dots,\boldsymbol{v}_n$ the columns of V, one can write:

$$A = \sum_{k=1}^{\min\{m,n\}} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\dagger \tag{C.1}$$

which clarifies why SVD is a generalization of eigendecomposition.
This decomposition is often used to compute approximations for matrices: if the singular values are sorted in decreasing order $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_K$, one can approximate the matrix A by retaining only the largest r singular values, thus obtaining a truncated version of A of rank r:

$$\tilde{A} = \sum_{k=1}^{r} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\dagger \tag{C.2}$$

This truncation, called SVD truncation, is the one that minimizes the Frobenius norm of the difference between A and $\tilde{A}$ (this is the content of the Eckart-Young theorem [29]).

SVD on tensors

If one wants to perform SVD truncation on a tensor (i.e. an array with more than two indices), it must be *reshaped* into a two-dimensional array, namely its indices must be grouped in two subsets and treated as macro-indices. For example, if one has a tensor written as those of the SIS model, i.e. a matrix whose entries depend one some (here one) variables $A_{i,j} = A(x)_{i,j}$, the reshape operation puts together (j,x) to form a macro index. Then, SVD reads:

$$A(x)_{i,j} = \sum_{k=1}^{K} \boldsymbol{u}_{i,k} \lambda_k \boldsymbol{v}(x)^{\dagger}_{k,j} \tag{C.3}$$

from which it is clear that i serves as row index for A and (j,x) as column index. Figure C.1 shows schematically how to reshape a tensor.

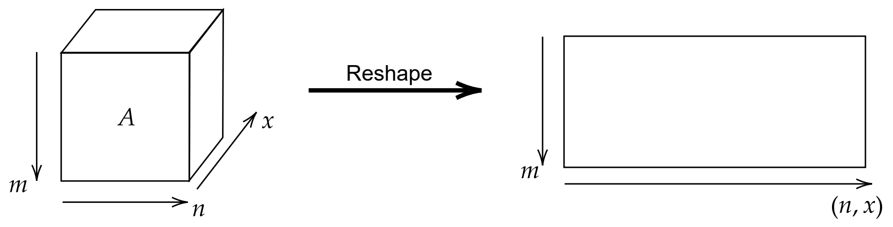

FIGURE C.1. A tensor $A(x)$ is reshaped to become a matrix, incorporating x in the column index. In the reshaped tensor, m is the row index and (n,x) is the column index

D Bethe Free Energy

When running the Belief Propagation algorithm, one tries to reconstruct the marginals of a probability distribution through successive iterations. In the following, a metric for the accuracy of the reconstruction is shown. This, in turn, allows to frame the problem in an equivalent way as an optimization one and makes possible useful connections to statistical physics.
Given a set of messages $\{\mu_{a\to i}, \mu_{i\to a}\}_{a\in A, i\in I}$ and the original probability distribution $p(\boldsymbol{x}) = \frac{1}{Z} e^{-H(\boldsymbol{x})} = \frac{1}{Z}\prod_a f_a(\boldsymbol{x}_a)$, the beliefs b_i and b_a can be derived from Equation 2.6 and Equation 2.7. Then reminding Theorem A.1, an approximation of p can be built as:

$$\begin{aligned}
q(\boldsymbol{x}) &:= \prod_{a\in A} \frac{b_a(\boldsymbol{x}_a)}{\prod_{i\in\partial a} b_i(x_i)} \prod_{i\in I} b_i(x_i) = \\
&= \prod_{a\in A} \frac{\frac{1}{z_a} f_a(\boldsymbol{x}_a) \prod_{i\in\partial a} \mu_{i\to a}(x_i)}{\prod_{i\in\partial a} \frac{1}{z_i} \prod_{b\in\partial i} \mu_{b\to i}(x_i)} \prod_{i\in I} \frac{1}{z_i} \prod_{a\in\partial i} \mu_{a\to i}(x_i) = \\
&= Zp(\boldsymbol{x}) \prod_{a\in A} \frac{\frac{1}{z_a} \prod_{i\in\partial a} \mu_{i\to a}(x_i)}{\prod_{i\in\partial a} \frac{1}{z_i} \prod_{b\in\partial i} \mu_{b\to i}(x_i)} \prod_{i\in I} \frac{1}{z_i} \prod_{a\in\partial i} \mu_{a\to i}(x_i)
\end{aligned}$$

Now, one can notice that the last product term contains $\prod_{i\in I}\prod_{a\in\partial i} \mu_{a\to i}(x_i)$ and the denominator of the first product term can be split as $\prod_{i\in\partial A}\left(\mu_{a\to i}(x_i) \prod_{b\in\partial i\setminus a} \mu_{b\to i}(x_i)\right)$, so that a simplification can be made:

$$q(\boldsymbol{x}) = Zp(\boldsymbol{x}) \prod_{a\in A} \frac{\frac{1}{z_a} \prod_{i\in\partial a} \mu_{i\to a}(x_i)}{\prod_{i\in\partial a} \frac{1}{z_i} \prod_{b\in\partial i\setminus a} \mu_{b\to i}(x_i)} \prod_{i\in I} \frac{1}{z_i}$$

Then, realizing that from Equation 2.4 one has $\mu_{i\to a}(x_i)\mu_{a\to i}(x_i)z_{i\to a} = \prod_{b\in\partial i}\mu_{b\to i}(x_i)$, the final form can be recovered:

$$\begin{aligned} q(\boldsymbol{x}) &= Zp(\boldsymbol{x})\prod_{a\in A}\frac{\frac{1}{z_a}\prod_{i\in\partial a}\mu_{i\to a}(x_i)}{\prod_{i\in\partial a}\frac{1}{z_i}z_{i\to a}\mu_{i\to a}(x_i)}\prod_{i\in I}\frac{1}{z_i} = \\ &= Zp(\boldsymbol{x})\prod_{a\in A}\frac{1}{z_a}\prod_{a\in A}\frac{1}{\prod_{i\in\partial a}\frac{1}{z_i}}\prod_{i\in I}\frac{1}{z_i}\prod_{a\in A}\frac{1}{\prod_{i\in\partial a}z_{i\to a}} \end{aligned}$$

or

$$q(\boldsymbol{x}) = \frac{Z}{Z_{\text{Bethe}}}p(\boldsymbol{x}) \tag{D.1}$$

with the quantity Z_{Bethe} defined as the normalization of $q(\boldsymbol{x})$, the approximate joint probability distribution:

$$\begin{aligned} Z_{\text{Bethe}} &:= \left(\prod_{a\in A}\frac{1}{z_a}\prod_{a\in A}\frac{1}{\prod_{i\in\partial a}\frac{1}{z_i}}\prod_{i\in I}\frac{1}{z_i}\prod_{a\in A}\frac{1}{\prod_{i\in\partial a}z_{i\to a}}\right)^{-1} = \\ &= \prod_{a\in A}z_a\prod_{a\in A}\prod_{i\in\partial a}\frac{1}{z_i}\prod_{i\in I}z_i\prod_{a\in A}\prod_{i\in\partial a}z_{i\to a} = \\ &= \prod_{a\in A}z_a\prod_{i\in I}z_i^{1-|\partial i|}\prod_{a\in A}\prod_{i\in\partial a}z_{i\to a} \end{aligned} \tag{D.2}$$

In analogy with Statistical Physics, this is called the Bethe partition function and a Bethe free energy can also be defined:

$$F_{\text{Bethe}} := -\log(Z_{\text{Bethe}}) = -\sum_{a\in A}\log(z_a) - \sum_{i\in I}(1-|\partial i|)\log(z_i) - \sum_{a\in A}\sum_{i\in\partial a}\log(z_{i\to a}) \tag{D.3}$$

This treatment has the consequence that for any BP fixed point, if $F = F_{\text{Bethe}}$, then $q \equiv p$ and vice versa.
Moreover, it can be shown that the BP update equations 2.4 and 2.5 are satisfied if and only if the Bethe free energy is minimized.

$$\{\mu_{a\to i}\mu_{i\to a}\}_{a\in A, i\in I} \text{ Satisfy 2.4 and 2.5} \quad \Longleftrightarrow \quad \nabla F_{\text{Bethe}} = 0 \tag{D.4}$$

Thus, the BP algorithm, if convergent, converges to a stationary point of the Bethe free energy.

E

Detailed Calculation of Derivatives

As pointed out in section 4.4, the important parts for the calculation of the derivatives of the log-likelihoods are $\frac{\partial}{\partial \lambda_{ji}} \Phi_i(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i})$ and $\frac{\partial}{\partial \rho_i} \Phi_i(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i})$. Here it is shown how the expressions 4.5 and 4.6 can be derived.

Concerning the derivatives with respect to the λ_{ji} parameters:

$$\begin{aligned}
\frac{\partial}{\partial \lambda_{ji}} \Phi_i(\overline{x}_i, \overline{\boldsymbol{x}}_{\partial i}) &= \frac{\partial}{\partial \lambda_{ji}} \left[w_i^0(x_i^0) \prod_{t=0}^{T-1} w_i^{t+1}(x_i^{t+1}, x_i^t, \boldsymbol{x}_{\partial i}^t) \right] = \\
&= \frac{\partial}{\partial \lambda_{ji}} \left[w_i^0(x_i^0) \prod_{t=0}^{T-1} \sum_{\{y_k^t\}_{k\in\partial i}} \left(\mathbb{P}[x_i^{t+1} | \{y_k^t\}_{k\in\partial i}, x_i^t] \prod_{k\in\partial i} \mathbb{P}[y_k^t | x_k^t, x_i^t] \right) \right] = \\
&= w_i^0(x_i^0) \frac{\partial}{\partial \lambda_{ji}} \left[\sum_{\{\overline{y}_k\}_{k\in\partial i}} \left(\prod_{t=0}^{T-1} \mathbb{P}[x_i^{t+1} | \{y_k^t\}_{k\in\partial i}, x_i^t] \prod_{k\in\partial i} \mathbb{P}[y_k^t | x_k^t, x_i^t] \right) \right] = \\
&= w_i^0(x_i^0) \sum_{\{\overline{y}_k\}_{k\in\partial i}} \left[\left(\prod_{t=0}^{T-1} \mathbb{P}[x_i^{t+1} | \{y_k^t\}_{k\in\partial i}, x_i^t] \right) \frac{\partial}{\partial \lambda_{ji}} \left(\prod_{t=0}^{T-1} \prod_{k\in\partial i} \mathbb{P}[y_k^t | x_k^t, x_i^t] \right) \right] = \\
&= w_i^0(x_i^0) \sum_{\{\overline{y}_k\}_{k\in\partial i}} \left[\left(\prod_{t=0}^{T-1} \mathbb{P}[x_i^{t+1} | \{y_k^t\}_{k\in\partial i}, x_i^t] \prod_{k\in\partial i\setminus\{j\}} \prod_{t=0}^{T-1} \mathbb{P}[y_k^t | x_k^t, x_i^t] \right) \times \right. \\
&\qquad \left. \times \frac{\partial}{\partial \lambda_{ji}} \left(\prod_{t=0}^{T-1} \mathbb{P}\left[y_j^t | x_j^t, x_i^t \right] \right) \right]
\end{aligned}$$

Using Equation 3.18 the derivative is easily calculated:

$$\frac{\partial}{\partial\lambda_{ji}}\Phi_i(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}) = w_i^0\left(x_i^0\right)\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left[\left(\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{k\in\partial i\setminus\{j\}}\prod_{t=0}^{T-1}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\times\right.$$
$$\left.\times\sum_{s=0}^{T-1}\left(\mathbb{I}\left[x_j^s=I\right]\left(\mathbb{I}\left[y_j^s=I\right]-\mathbb{I}\left[y_j^s=S\right]\right)\prod_{t\neq s}\mathbb{P}\left[y_j^t|x_j^t,x_i^t\right]\right)\right]$$

While, for what concerns the derivatives with respect to the ρ_i parameters:

$$\frac{\partial}{\partial\rho_i}\Phi_i(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}) = \frac{\partial}{\partial\rho_i}\left[w_i^0\left(x_i^0\right)\prod_{t=0}^{T-1}w_i^t\left(x_i^t,x_i^{t-1},\boldsymbol{x}_{\partial i}^{t-1}\right)\right]=$$
$$=\frac{\partial}{\partial\rho_i}\left[w_i^0\left(x_i^0\right)\prod_{t=0}^{T-1}\sum_{\{y_k^t\}_{k\in\partial i}}\left(\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{k\in\partial i}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\right]=$$
$$=w_i^0\left(x_i^0\right)\frac{\partial}{\partial\rho_i}\left[\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left(\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{k\in\partial i}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\right]=$$
$$=w_i^0\left(x_i^0\right)\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left[\frac{\partial}{\partial\rho_i}\left(\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\right)\left(\prod_{t=0}^{T-1}\prod_{k\in\partial i}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\right]$$

Using Equation 3.16 and Equation 3.17:

$$\frac{\partial}{\partial\rho_i}\Phi_i(\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}) = w_i^0\left(x_i^0\right)\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left[\sum_{s=0}^{T-1}\left(\mathbb{I}\left[x_i^s=I\right]\left(\mathbb{I}\left[x_i^{s+1}=S\right]-\mathbb{I}\left[x_i^{s+1}=I\right]\right)\times\right.\right.$$
$$\left.\left.\times\prod_{t\neq s}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\right)\left(\prod_{t=0}^{T-1}\prod_{k\in\partial i}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\right]$$

Then, the expressions 4.7 and 4.8 can be obtained by inserting these two last results into Equation 4.4:

$$\frac{\partial}{\partial\lambda_{ji}}\log(z_i)=\frac{1}{z_i}\sum_{\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}}w_i^0\left(x_i^0\right)\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left[\left(\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{k\in\partial i\setminus\{j\}}\prod_{t=0}^{T-1}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\times\right.$$
$$\left.\times\sum_{s=0}^{T-1}\left(\mathbb{I}\left[x_j^s=I\right]\left(\mathbb{I}\left[y_j^s=I\right]-\mathbb{I}\left[y_j^s=S\right]\right)\prod_{t\neq s}\mathbb{P}\left[y_j^t|x_j^t,x_i^t\right]\right)\right]\prod_{k\in\partial i}\mu_{k\to i}(\overline{x}_k,\overline{x}_i)=$$
$$=\frac{1}{z_i}\sum_{s=0}^{T-1}\sum_{\{\overline{y}_k\}_{k\in\partial i}}\sum_{\overline{x}_i,\overline{x}_j}w_i^0\left(x_i^0\right)\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{t\neq s}\mathbb{P}\left[y_j^t|x_j^t,x_i^t\right]\mu_{j\to i}\left(\overline{x}_j,\overline{x}_i\right)\times$$
$$\times\mathbb{I}\left[x_j^s=I\right]\left(\mathbb{I}\left[y_j^s=I\right]-\mathbb{I}\left[y_j^s=S\right]\right)\sum_{\overline{\boldsymbol{x}}_{\partial i\setminus\{j\}}}\prod_{k\in\partial i\setminus\{j\}}\prod_{t=0}^{T-1}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\mu_{k\to i}(\overline{x}_k,\overline{x}_i)=$$
$$=\frac{1}{z_i}\sum_{s=0}^{T-1}\sum_{\{\overline{y}_k\}_{k\in\partial i}}\sum_{\overline{x}_i,\overline{x}_j}w_i^0\left(x_i^0\right)\prod_{t=0}^{T-1}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\prod_{t\neq s}\mathbb{P}\left[y_j^t|x_j^t,x_i^t\right]\mu_{j\to i}\left(\overline{x}_j,\overline{x}_i\right)\times$$
$$\times\mathbb{I}\left[x_j^s=I\right]\left(\mathbb{I}\left[y_j^s=I\right]-\mathbb{I}\left[y_j^s=S\right]\right)\tilde{\mu}_{\partial i\setminus\{j\}\to i}\left(\overline{y}_{\partial i\setminus\{j\}},\overline{x}_i\right)$$

and

$$\begin{aligned}\frac{\partial}{\partial\rho_i}\log(z_i) &= \frac{1}{z_i}\sum_{\overline{x}_i,\overline{\boldsymbol{x}}_{\partial i}} w_i^0\left(x_i^0\right)\sum_{\{\overline{y}_k\}_{k\in\partial i}}\left[\sum_{s=0}^{T-1}\left(\mathbb{I}\left[x_i^s=I\right]\left(\mathbb{I}\left[x_i^{s+1}=S\right]-\mathbb{I}\left[x_i^{s+1}=I\right]\right)\times\right.\right.\\
&\qquad\left.\left.\times\prod_{t\neq s}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\right)\left(\prod_{t=0}^{T-1}\prod_{k\in\partial i}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\right)\right]\prod_{k\in\partial i}\mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right)=\\
&=\frac{1}{z_i}\sum_{s=0}^{T-1}\sum_{\{\overline{y}_k\}_{k\in\partial i}}\sum_{\overline{x}_i}\mathbb{I}\left[x_i^s=I\right]\left(\mathbb{I}\left[x_i^{s+1}=S\right]-\mathbb{I}\left[x_i^{s+1}=I\right]\right)w_i^0\left(x_i^0\right)\times\\
&\qquad\times\prod_{t\neq s}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\sum_{\overline{x}_{\partial i}}\prod_{k\in\partial i}\prod_{t=0}^{T-1}\mathbb{P}\left[y_k^t|x_k^t,x_i^t\right]\prod_{k\in\partial i}\mu_{k\to i}\left(\overline{x}_k,\overline{x}_i\right)=\\
&=\frac{1}{z_i}\sum_{s=0}^{T-1}\sum_{\{\overline{y}_k\}_{k\in\partial i}}\sum_{\overline{x}_i}\mathbb{I}\left[x_i^s=I\right]\left(\mathbb{I}\left[x_i^{s+1}=S\right]-\mathbb{I}\left[x_i^{s+1}=I\right]\right)w_i^0\left(x_i^0\right)\times\\
&\qquad\times\prod_{t\neq s}\mathbb{P}\left[x_i^{t+1}|\{y_k^t\}_{k\in\partial i},x_i^t\right]\tilde{\mu}_{\partial i\to i}\left(\overline{y}_{\partial i},\overline{x}_i\right)\end{aligned}$$

F

ROC CURVE

Classification problems are those in which some input elements have to be assigned to two or more classes, according to data. For example, discriminating whether a potential edge is present or not in a graph is a classification problem, more exactly a binary classification problem, the two classes being that of true edges and that of false edges.
The Receiver Operating Characteristic (ROC) curve is a tool to evaluate the accurateness of a classifier (the solver of a classification problem).
Usually, binary classifiers give a score to each element (if normalized to 1 it can be interpreted as the probability of belonging to one class) and then decide the belonging of each element to one class based on a threshold.
Assuming that the classifier is looking for some sort of condition, and the two classes are thus called positive and negative, several parameters can be defined to assess the performance of the classifier:

- True Positive Rate (TPR): it is the fraction of positive results conditioned on the tested element being positive

- False Positive Rate (False Positive Rate): it is the fraction of positive results conditioned on the tested element being negative

- True Negative Rate (True Negative Rate): it is the fraction of negative results conditioned on the tested element being negative

- False Negative Rate (False Negative Rate): it is the fraction of negative results conditioned on the tested element being positive

The ROC curve is a plot of the true positive rate against the false positive rate for each possible threshold. In this way, the performance of the algorithm can be captured, independently of the threshold. Some examples are represented in Figure F.1. The area

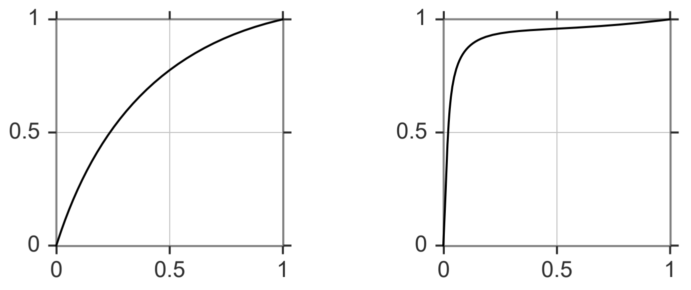

FIGURE F.1. Two examples of ROC curves. Better classifiers have an ROC curve that is shifted towards the upper-left corner, which means that, for a certain threshold, they have a high TPR and a low FPR.

under the ROC curve (AUC) represents the probability that the model, when given a positive and a negative element as input, gives the positive one a higher score than the negative one. For this reason, usually the AUC is used as a single parameter to measure the accurateness of a classifier, with an AUC of 0.5 indicating a random classifier and an AUC of 1.0 indicating a perfect classifier.

Bibliography

[1] M. Timme and J. Casadiego.
“Revealing Networks from Dynamics: An Introduction”.
In: *Journal of Physics A: Mathematical and Theoretical* 47.34 (Aug. 29, 2014), p. 343001.
DOI: `10.1088/1751-8113/47/34/343001`.

[2] H. S. Rodrigues.
“Application of SIR Epidemiological Model: New Trends”.
In: *International Journal of Applied Mathematics and Informatics* 10 (2016).

[3] T. M. Liggett.
Stochastic Interacting Systems: Contact, Voter and Exclusion Processes.
Red. by S. S. Chern et al.
Vol. 324.
Grundlehren Der Mathematischen Wissenschaften.
Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.
DOI: `10.1007/978-3-662-03990-8`.

[4] D. Kempe, J. Kleinberg, and É. Tardos.
“Maximizing the Spread of Influence through a Social Network”.
In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
KDD ’03.
Washington, D.C., 2003,
Pp. 137–146.

DOI: 10.1145/956750.956769.

[5] D. F. Bernardes, M. Latapy, and F. Tarissan.
"Relevance of SIR Model for Real-world Spreading Phenomena: Experiments on a Large-scale P2P System".
In: *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*.
2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012).
Istanbul: IEEE, Aug. 2012,
Pp. 327–334.
DOI: 10.1109/ASONAM.2012.62.

[6] D. Aadland, D. Finno, and K. X. D. Huang.
The Dynamics of Economic Epidemiology Equilibria.
MPRA Paper 29299.
Munich: University Library of Munich, Germany, 2011.

[7] A. Braunstein, A. Ingrosso, and A. P. Muntoni.
"Network Reconstruction from Infection Cascades".
In: *Journal of The Royal Society Interface* 16.151 (Feb. 2019), p. 20180844.
DOI: 10.1098/rsif.2018.0844.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin.
"Maximum Likelihood from Incomplete Data via the EM Algorithm".
In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38.
JSTOR: 2984875.

[9] F. Kschischang, B. Frey, and H.-A. Loeliger.
"Factor Graphs and the Sum-Product Algorithm".
In: *IEEE Transactions on Information Theory* 47.2 (Feb./2001), pp. 498–519.
DOI: 10.1109/18.910572.

[10] J. S. Yedidia, W. T. Freeman, and Y. Weiss.
"Generalized Belief Propagation".
In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*.
Advances in Neural Information Processing Systems.
Cambridge, MA, USA: MIT Press, 2001.

[11] M. Chertkov and V. Y. Chernyak.
"Loop Calculus in Statistical Physics and Information Science".
In: *Physical Review E* 73.6 (June 1, 2006), p. 065102.
DOI: 10.1103/PhysRevE.73.065102.

[12] G. T. Cantwell and M. E. J. Newman.
"Message Passing on Networks with Loops".
In: *Proceedings of the National Academy of Sciences* 116.47 (Nov. 19, 2019), pp. 23398–23403.
DOI: 10.1073/pnas.1914893116.

[13] A. Kirkley, G. T. Cantwell, and M. E. J. Newman.
"Belief Propagation for Networks with Loops".
In: *Science Advances* 7.17 (Apr. 23, 2021), eabf1211.
DOI: 10.1126/sciadv.abf1211.

[14] J. Biamonte and V. Bergholm.
Tensor Networks in a Nutshell.
July 31, 2017.
arXiv: 1708.00006 [cond-mat, physics:gr-qc, physics:hep-th, physics:math-ph, physics:quant-ph].
URL: http://arxiv.org/abs/1708.00006 (visited on 05/17/2024).
Pre-published.

[15] I. V. Oseledets.
"Tensor-Train Decomposition".
In: *SIAM Journal on Scientific Computing* 33.5 (Jan. 2011), pp. 2295–2317.
DOI: 10.1137/090752286.

[16] I. Oseledets and E. Tyrtyshnikov.
"TT-cross Approximation for Multidimensional Arrays".
In: *Linear Algebra and its Applications* 432.1 (Jan. 2010), pp. 70–88.
DOI: 10.1016/j.laa.2009.07.024.

[17] B. T. Polyak.
"Some Methods of Speeding up the Convergence of Iteration Methods".
In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (Jan. 1, 1964), pp. 1–17.
DOI: 10.1016/0041-5553(64)90137-5.

[18] D. P. Kingma and J. Ba.

Adam: A Method for Stochastic Optimization.
Jan. 29, 2017.
arXiv: `1412.6980 [cs]`.
URL: `http://arxiv.org/abs/1412.6980` (visited on 06/17/2024).
Pre-published.

[19] J. Duchi, E. Hazan, and Y. Singer.
"Adaptive Subgradient Methods for Online Learning and Stochastic Optimization".
In: *Journal of Machine Learning Research* 12 (July 11, 2011).

[20] S. Ruder.
An Overview of Gradient Descent Optimization Algorithms.
June 15, 2017.
arXiv: `1609.04747 [cs]`.
URL: `http://arxiv.org/abs/1609.04747` (visited on 06/17/2024).
Pre-published.

[21] S. Crotti and A. Braunstein.
"Matrix Product Belief Propagation for Reweighted Stochastic Dynamics over Graphs".
In: *Proceedings of the National Academy of Sciences* 120.47 (Nov. 21, 2023), e2307935120.
DOI: `10.1073/pnas.2307935120`.

[22] F. Altarelli et al.
"Large Deviations of Cascade Processes on Graphs".
In: *Physical Review E* 87.6 (June 11, 2013), p. 062115.
DOI: `10.1103/PhysRevE.87.062115`.

[23] T. Barthel, C. De Bacco, and S. Franz.
"Matrix Product Algorithm for Stochastic Dynamics on Networks Applied to Nonequilibrium Glauber Dynamics".
In: *Physical Review E* 97.1 (Jan. 29, 2018), p. 010104.
DOI: `10.1103/PhysRevE.97.010104`.

[24] U. Schollwöck.
"The Density-Matrix Renormalization Group in the Age of Matrix Product States".
In: *Annals of Physics* 326.1 (Jan. 2011), pp. 96–192.
DOI: `10.1016/j.aop.2010.09.012`.

[25] T. Barthel.

"The Matrix Product Approximation for the Dynamic Cavity Method".
In: *Journal of Statistical Mechanics: Theory and Experiment* 2020.1 (Jan. 23, 2020), p. 013217.
DOI: 10.1088/1742-5468/ab5701.

[26] F. Florio.
FedericoFlorio/MatrixProductBP.Jl.
May 17, 2024.
URL: https://github.com/FedericoFlorio/MatrixProductBP.jl (visited on 08/14/2024).

[27] R. A. Rossi and N. K. Ahmed.
"The Network Data Repository with Interactive Graph Analytics and Visualization".
In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.
AAAI Conference on Artificial Intelligence.
2015.

[28] M. Mézard and A. Montanari.
Information, Physics, and Computation.
Oxford Graduate Texts.
Oxford: Oxford university press, 2009.

[29] C. Eckart and G. Young.
"The Approximation of One Matrix by Another of Lower Rank".
In: *Psychometrika* 1.3 (Sept. 1936), pp. 211–218.
DOI: 10.1007/BF02288367.