University of Novi Sad
Faculty of Sciences
Department of mathematics and informatics

Detecting communities in networks using Belief Propagation algorithm

Jelena Mihajlović
Supervisor: Dr Dragana Bajović

September 3, 2020

Contents

Abstract **4**

Acknowledgement **5**

1 Introduction **6**

2 Stochastic block model **9**

2.1 Generative models 9

2.2 Hierarchical module structures 11

2.3 Stochastic block model 12

3 Graphical models **17**

3.1 Markov Networks 19

3.2 Factor graphs 21

3.3 Belief propagation algorithm on factor graphs 23

3.3.1 Marginal inference and message passing 23

3.3.2 BP algorithm 24

4 Algorithm **27**

4.1 Deriving BP 28

4.2 Belief Propagation 33

5 Algorithm implementation **35**

5.1 Implementation of BP in Python 35

5.2 Zachary's karate club 41

5.3 Results and discussion 43

Conclusion **46**

Bibliography **48**

Biography **51**

List of Figures

1.1 A network with 4 communities 6
1.2 Example of an assortative network 8

2.1 Graphical representation of the hierarchical structure [4] . . . 11
2.2 Resulting network with communities [4] 12
2.3 Examples of affinity matrices 13
2.4 An example of NG benchmark 14

3.1 Markov networks 19
3.2 Factor graphs 21
3.3 Markov network 22
3.4 Representation of the Markov network shown in Figure 3.3 with factor graphs 22
3.5 Four variable Markov chain 23
3.6 A tree without branches 24
3.7 Variable to Factor 25
3.8 Factor to Variable messages 26
3.9 Marginal 26

4.1 Graphical model associated with the pdf $P(x_1, x_2, x_3, x_4, x_5) \propto f_{1,2}(x_1, x_2)f_{1,3}(x_1, x_3)f_{1,4}(x_1, x_4)f_{1,5}(x_1, x_5)f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_4)f_5(x_5)$ 28
4.2 Message passing graph [12] 29

5.1 Synthetic network of two cliques 35
5.2 Table of random edge messages 36
5.3 Network after inferring group assignment 40
5.4 Zachary's karate club social network 41
5.5 Zachary's karate club neighbor random messages 42
5.6 Zachary's karate club community assignment 44
5.7 Zachary's karate club community assignment disassortative initialization parameter 45
5.8 The likelihood of the partitions when $q = 2$ [1] 46

Listings

5.1 Generation od random messages between the neighbor nodes . 36
5.2 Generation of random messages between the non-neighbor nodes 37
5.3 BP algorithm - calculating the messages between the nodes . . 37
5.4 Caltulating the stochastic block model parameters in Python . 39
5.5 Calculating the messages for Zachary's carate club network . . 42

Abstract

In this master thesis we implemented a common message passing algorithm called Belief Propagation [12]. We applied this algorithm on stochastic block model, an often used generative model for social and biological networks, and we tackled the problem of inferring communities or groups from the topology of the network.

We started by introducing theoretical aspects of the problem, considering stochastic block model, graphical models and finally belief propagation algorithm.

We implemented the algorithm in Python programming language and represented the code and its results in the final chapter. We used two different networks for the analysis, one of which was synthetically generated and other that is widely used for the community detection, Zachary's karate club. While BP is expected to work on tree graphs, our numerical results indicate the benefits of this algorithm even when that is not a case.

Acknowledgment

This work would not have been possible without support of many people. First of all, I would like to thank my mentor Dr. Dragana Bajovic who offered guidance and support with constructive advice, suggestions and assistance during the work on this master thesis. I owe a great deal of gratitude to my friends and colleagues who have, in many ways, helped and supported me during my studies, and made them more beautiful. And finally, I would like to thank Aleksa, my parents, and numerous close friends who were there every step of this long process with me, always offering support and love.

Chapter 1

Introduction

Many systems in nature and around us can be expressed by a large number of nodes or vertices where typically we can observe different kinds of connections between them. A structure that is described in this way is called a network.

Networks, as a systems' structure, could represent computer networks, telecommunication networks, biological networks, social networks, etc. For other examples see [3]. Lately, internet and social media have been supplying enormous amount of data for scientists to process and analyze. The necessity to deal with such a large amount of data has generated the need for more intensive analysis of networks.

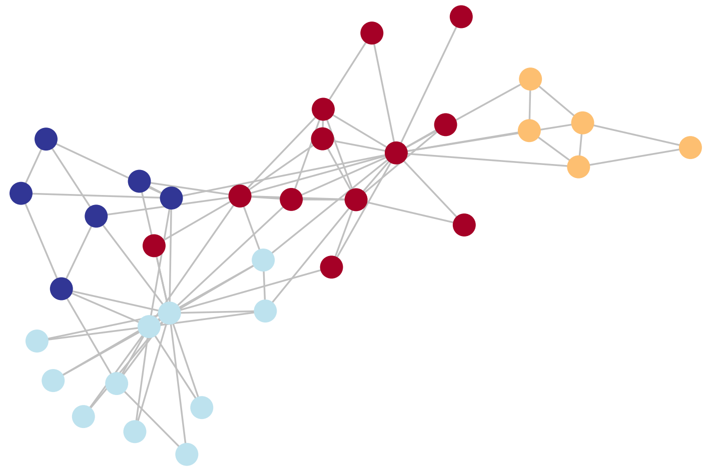


Figure 1.1: A network with 4 communities

Existence of community structure is one of the most important aspects of these networks. The society offers a wide variety of examples of possible group organizations: families, work circles and friendships, villages, cities, nations. Examples of communities are also proteins that have the same specific function in a cell, inside World Wide Web this corresponds to groups of pages that deal with the same or related topics, etc.

Identifying communities is important for other reasons as well. For example, a community of customers with similar interests within the Amazon customer-product relationship network represents a prime example that allows for an effective recommender systems. Recognizing modules and their boundaries, for instance, enables classification of nodes according to their structural position in modules. Nodes with a central position in a network can have an important control role. Such classification is significant in social and metabolic networks.

The arising question is can something be concluded about the underlying system based purely on the topology of the network.

Let's assume each of the nodes may belong to one of q communities, and the structure of the network may depend in an a priori unknown way on the group memberships. Hence, essentially important question is if it is possible to learn in what way the group membership influences the structure of the network, and which node belongs to which group, considering we only know the topology of the network.

There are two different types of network structures. First type are *assortative* networks, meaning that there is a higher density of connections within communities than across them. Typical example of this kind of network are social networks, as people with similar interests or characteristics tend to be more connected. For example, people with same level of education, same age or gender. The second type of network structure are *disassortative* networks. For these kinds of networks there are less connections inside a group and more between different communities. For instance if we observe food network a group of predators can be considered as one community. Typically, there are fewer connections between them because they do not eat each other but rather they eat similar prey. On Figures 1.1 and 1.2 we can see examples of assortative networks with 4 and 6 different communities, respectively. As we can see from the figures, there are more connections (edges) inside of an community than between the communities.

In this work we are going to analyze a generative model for random graphs, known as the *stochastic block model* [1]. This is the most commonly used generative model for random modular networks. Stochastic block model is defined in Chapter 2. After that we focus on community detection of two different networks. There are many different algorithms developed for community detection. The one studied in this work is based on the well known message-passing algorithm called *belief propagation* [9].

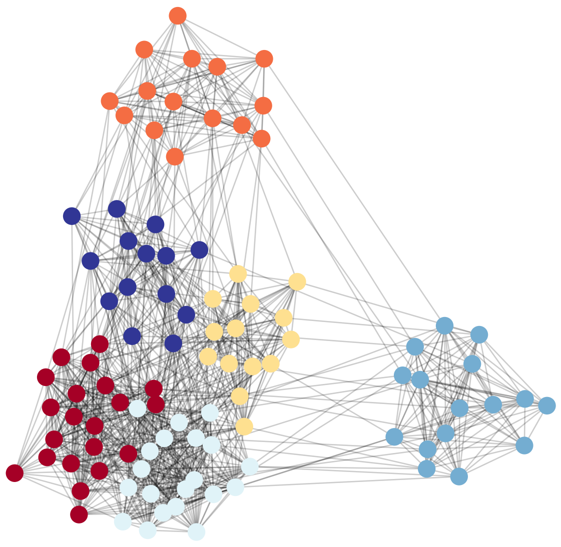


Figure 1.2: Example of an assortative network

Chapter 2

Stochastic block model

2.1 Generative models

Generative models are a tool for generating random networks based on a set of input parameters θ. They typically define the probability $P(G|\theta)$ of each possible network instance G, expressed as a function of model parameters. Given a network instance G it is often of interest to find θ that maximizes the probability of G $P(G|\theta)$

Set of parameters θ can be discrete or continuous. For communities model, it is specified as the triplet $(\{t\}, \{\pi\}, q)$, where $\{t\}$ indicates the community assignment, $\{\pi\}$ the model parameters, and q the number of modules. The notation and model structure may vary from model to model, but this is the general concept.

The communities in a network are then discovered by fitting the generative model to the underlying network. Generative models are the main focus of many methods for detecting communities.

General form of generative models for complex networks is given by:

$$P(G|\theta) = \prod_{i<j} P(A_{ij}|\theta), \tag{2.1}$$

where $\mathbf{A}$ is the adjacency matrix of the network G,i.e., $A_{ij} = 1$ if there is an edge between nodes i and j and $A_{ij} = 0$ otherwise. Hence, the probability of generating the network G is the probability of generating the adjacency matrix $\mathbf{A}$.

The simplest generative model is the Erdős - Rényi random graph. In this model every pair of nodes is connected independently with the same probability.

There are two classes of generative models for networks with communities: Stochastic block models and Latent space models [11]. We will focus on the first class, stochastic block model.

However, we remark that the algorithm that is studied here can easily be generalized to several generative models including hierarchical module structures [4], overlapping modules [5], or degree-corrected versions of the stochastic block model [6].

2.2 Hierarchical module structures

Networks frequently illustrate hierarchical organization where vertices are split into groups that further split into groups of groups, and so forth. Many times these groups correspond to known functional communities.

The hierarchical organization of a network model is defined as follows. Let G be a graph with n vertices. And let D be the dendrogram, a binary tree with n leaves that correspond to the vertices of graph G. There are $n-1$ internal nodes of the dendogram D and each corresponds to a group of vertices descending from it. With each internal node v there is a probability p_v associated with it. It holds that $p_v = p_{ij}$, where p_{ij} corresponds to the probability of an edge between vertices i and j from graph G and v is their lowest common ancestor in the dendogram D. The hierarchical random graph is then defined by a pair (D, p_v)

Hierarchical structure is depicted by a tree or dendrogram shown in Figure 2.1. Gray nodes represent the groups of vertices bellow them and the shades correspond to probabilities of an edge between left and right subtrees, while colored squares at the bottom correspond to vertices of the graph.

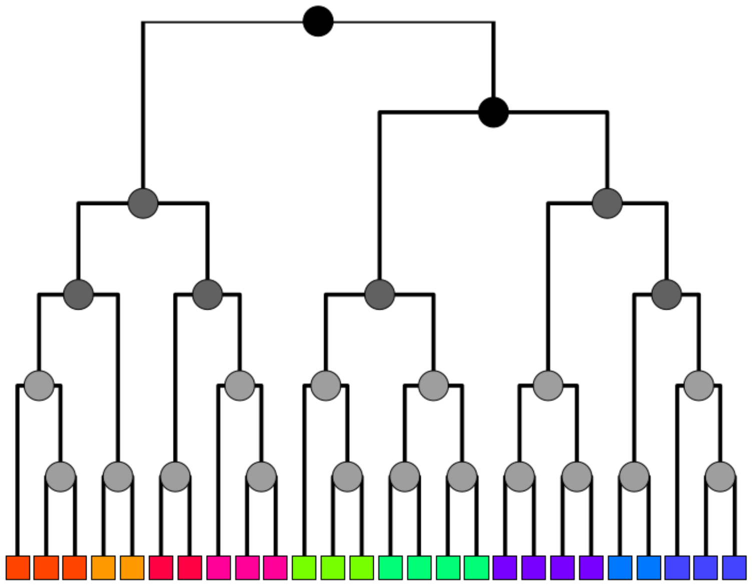


Figure 2.1: Graphical representation of the hierarchical structure [4]

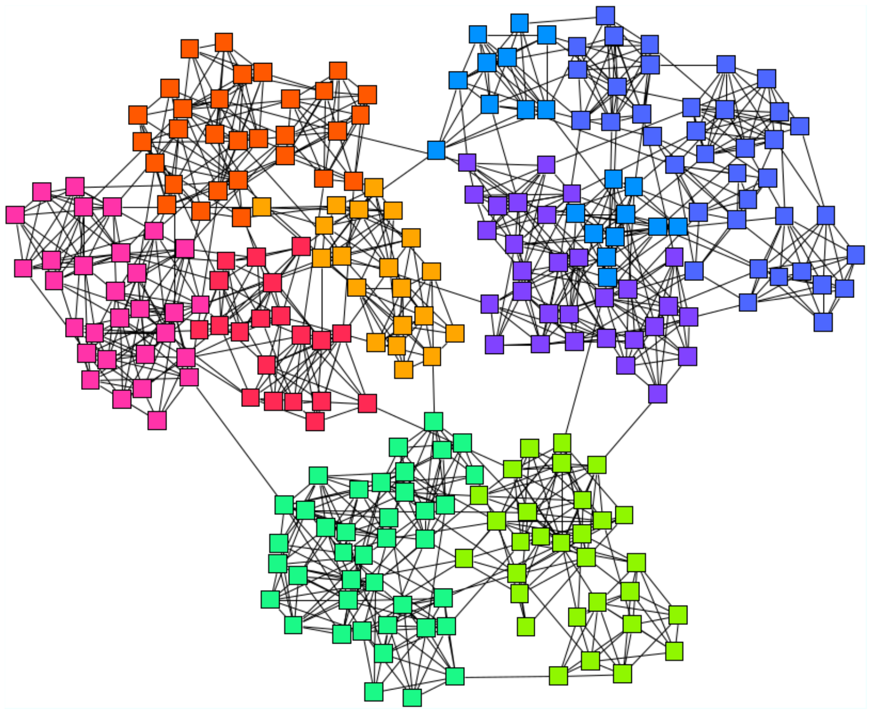


Figure 2.2: Resulting network with communities [4]

2.3 Stochastic block model

Consider network G of N nodes. Parameter set θ contains the following parameters: q - the number of modules (communities), n_a - the expected probability of node belonging to group a, $1 \leq a \leq q$ and $q \times q$ affinity matrix p_{ab} (Figure 2.3).

Each node has a hidden label t_i, $t_i \in \{1, 2, ..., q\}$, which tells us to which community node i belongs to. For each pair of nodes i and j, $i \neq j$, an edge is placed with probability p_{t_i,t_j}, or equivalently with probability $1 - p_{t_i,t_j}$ there is no edge between i and j. In this work we will use the re-scaled affinity matrix $c_{ab} = Np_{ab}$ because we are interested in the sparse network regime where $p_{ab} = O(1/N)$. We assume that $c_{ab} = O(1)$ when $N \to \infty$.

The only information about the network that is provided is the adjacency matrix $\mathbf{A}$. Self loops are not allowed, which means $A_{ii} = 0$.

Let N_a denote the number of nodes in group a, $1 \leq a \leq q$. From here follows $n_a = \frac{N_a}{N}$. We calculate the expected number of edges from group a to group b as $M_{ab} = p_{ab}N_aN_b$ and $M_{aa} = p_{aa}\frac{N_a(N_a-1)}{2}$ when $a = b$. As previously mentioned, we use the re-scaled affinity matrix $c_{ab} = Np_{ab}$. In the limit of large N, the average degree of the network G is:

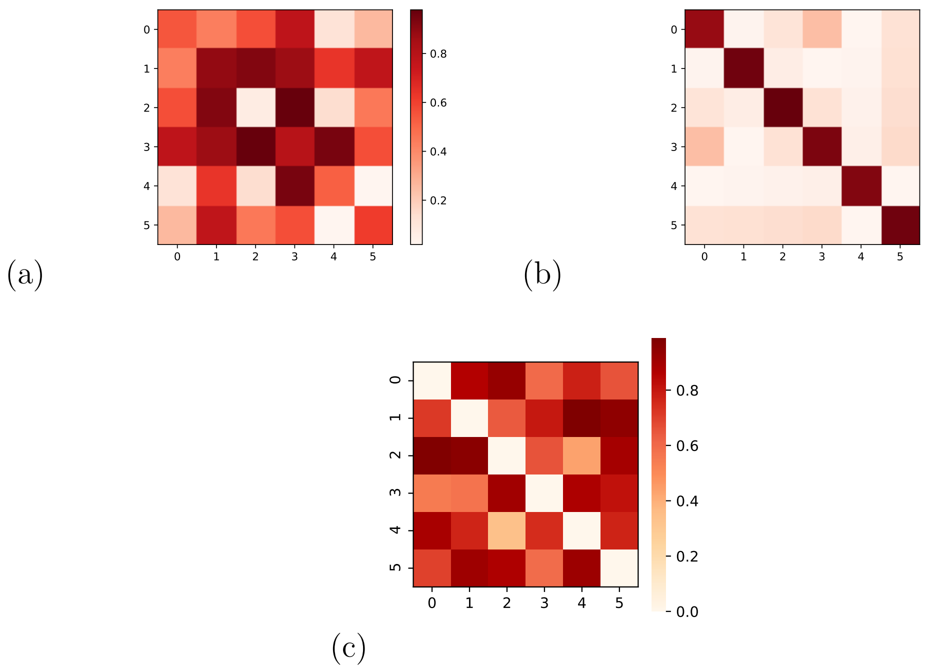


Figure 2.3: Examples of affinity matrices

(a) Affinity matrix of a random network (b) Affinity matrix of an assortative network (c) Affinity matrix of an disassortative network

$$c = \sum_{a,b} \frac{M_{ab}}{N} = \sum_{a,b} \frac{p_{ab} N_a N_b}{N} = \sum_{a,b} c_{ab} n_a n_b. \tag{2.2}$$

Equation 2.2 refers to a directed network. In undirected case we have the following:

$$c = \sum_{a<b} \frac{M_{ab}}{N} + \sum_{a} \frac{M_{aa}}{N} = \sum_{a<b} c_{ab} n_a n_b + \sum_{a} c_{aa} \frac{n_a^2}{2}. \tag{2.3}$$

There are several special cases of the stochastic block model; we will mention two of them:

(1) “Four groups” test of Newman and Girvan. This is also a special case of the *planted-l-partition model* [7] which partitions the graph with $N = q * N_a$ vertices into q groups with each group having N_a vertices. For the ”four groups” test, a network is divided into four groups, $q = 4$ and $n_a = 1/4$ for each a, $1 \leq a \leq 4$, the groups are uniformly represented. The structure of the network is assortative and it holds:

$$p_{ab} = \begin{cases} p_{in}, & a = b \\ p_{out}, & a \neq b, \end{cases} \tag{2.4}$$

where $p_{in} > p_{out}$. For this model average degree can be calculated in the following way:

$$c = p_{in} * (N_a - 1) + p_{out} * N_a * (q - 1). \tag{2.5}$$

By altering the difference between p_{in} and p_{out}, the more or less challenging structures are created for community detection algorithms. The network is generated from 128 nodes,with each group having 32 nodes. Average degree c of the network is equal to 16. From this and the equation 2.5 it can be concluded that $p_{in} + 3p_{out} \approx 1/2$, which means that, in this case, p_{in} and p_{out} are can be chosen independently. It is common to use also the following parameters $z_{in} = p_{in}(N_a - 1) = 31p_{in}$ and $z_{out} = p_{out}N_a(q - 1) = 96p_{out}$, indicating the expected internal and external degree of a vertex, respectively.

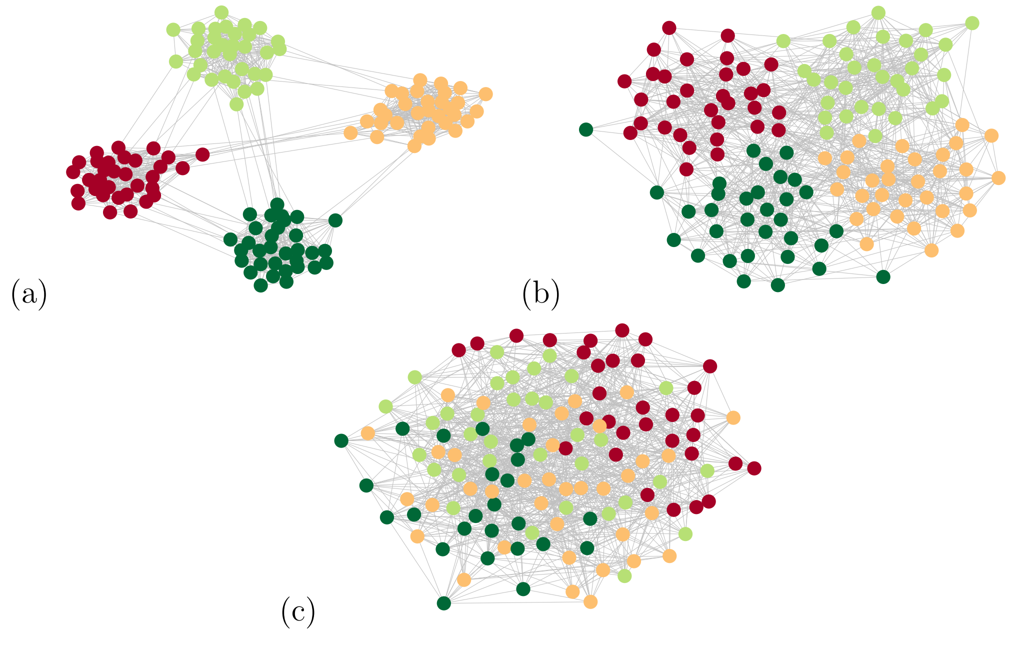


Figure 2.4: An example of NG benchmark

(a): $z_{in} = 15$; (b)$z_{in} = 11$; (c): $z_{in} = 8$

(2) The *planted partition model* is a generalization of the example given in (1). We have $n_a = 1/q$. Now, we are not only interested in the sparse case, so we do not need to use the scaled affinity matrix. We have that $p_{ab} = p_{in}$ if $a = b$ and $p_{ab} = p_{out}$ if $a \neq b$ and, again, $p_{in} > p_{out}$. For

$p_{in} - p_{out} > O(\log N/N)$ it can be shown [8] that the planted partition is with high probability equal to the best possible partition, in terms of minimizing the number of edges between groups.

Now, assume that a network is generated using the stochastic block model. All we know about this network is the resulting graph G and, with that, its adjacency matrix $\mathbf{A}$. Parameters q, n_a, p_{ab} and the labels t_i are yet to be determined. So, we need to resolve the following two questions:

(1) *Parameter learning* - Given the graph G, what are the most likely values of the parameters q, n_a, p_{ab} that were used to generate the graph [1]?

(2) *Inferring the group assignment* - Given the graph G and the parameters q, n_a, p_{ab}, what is the most likely assignment of a label (group) to a given node [1]?

To answer these questions we need to introduce the following measures:

$$C(\{t_i\}, \{q_i\}) = \max_{\pi} \frac{1}{N} \sum_i \delta_{t_i, \pi(q_i)} [1], \tag{2.6}$$

$C(\{t_i\}, \{q_i\})$ we call *agreement* between the original assignment $\{t_i\}$ and its estimate $\{q_i\}$, where π ranges over the permutations on q elements. This measure helps us answer the second question - determining the most likely group assignment of a label, this is why we are considering all possible permutations. If some of the permutations correspond to original labeling, the agreement $C(\{t_i\}, \{q_i\})$ would be exactly 1, if this is not the case we would have the fraction of overlapping group assignments.

The second measure that we consider is called the *overlap* and it is actually the normalized agreement:

$$Q(\{t_i\}, \{q_i\}) = \max_{\pi} \frac{\frac{1}{N} \sum_i \delta_{t_i, \pi(q_i)} - \max_a n_a}{1 - \max_a n_a} [1]. \tag{2.7}$$

It also holds that if $t_i = q_i$ for all nodes i then $Q = 1$, which means that we've found the exact labeling.

Original labeling $\{q_i\}$ is correlated with the original one $\{t_i\}$ if when $N \to \infty$ the overlap Q is strictly positive.

On the other hand, when we naively guess that every vertex belongs to the largest group, then the numerator of Q is 0 and $Q = 0$. Given $\{c_{a,b}\}$, $\{n_a\}$ and a set of observed edges E, we can write down the probability of a labeling $\{q_i\}$ as

$$P(\{q_i\}_i) = \prod_{\substack{(i,j) \notin E \\ i \neq j}} (1 - p_{q_i, q_j}) \prod_{(i,j) \in E} p_{q_i, q_j} \prod_{i \in [q]} n_{q_i}$$

How do we try to infer $\{q_i\}$ such that we have maximum correlation (up to permutation) with the true labeling? The answer is to use the maximum likelihood estimator of the marginal distribution of each q_i, but with caution. We should label q_i with the $r \in [q]$ such that $P(q_i = r)$ is maximized. We must be careful when $P(\{q_i\}_i)$ is invariant under permutations of the labeling $\{q_i\}_i$, so that each marginal $P(q_i)$ is the uniform distribution. For example, this happens in community detection, when all the group sizes $n_1, \ldots, n_q$ are equal.

Chapter 3

Graphical models

Since we are already familiar with the network structures let us now consider graphical models. Since we can say that graphical model is also a network the main difference is that in probabilistic graphical model, each node represents a random variable, and each edge corresponds to probabilistic relationships between these variables. In a random network this is not necessarily true.

A graph-based representation of probability distributions is called *Probabilistic graphical model.* Nodes in this graph represent random variables (in some cases group of random variables), and the edges express probabilistic relationships between them.

The graphical models are there to help us in describing a certain problem. After representing a problem as a graphical model, we perform the probabilistic inference. Hence, this is a two part process:

1. **Modeling** First all potentially relevant variables are identified. Next step then is to describe how these variables can interact. This is accomplished by using structural assumptions considering the form of the joint probability distribution of all the variables, typically, assumptions of independence of variables. Each class of graphical model corresponds to a factorization property of the joint distribution [9].

2. **Inference** Combining graphical models with accurate inference algorithms is central to successful graphical modelling.

There are two broad classes of graphical models, although this is not a strict separation. First class are graphical models that are more useful in modeling. Both Markov networks and Bayesian networks are in this class. Second class are those graphical models that are useful in representing inference algorithms. Here the most popular graphical models are *factor graphs.*

There are, also, two types of graphical models: directed and undirected, and they are defined as follows:

Definition 3.0.1. Directed graphical model is a model where the links of the graph are directed at one side and this is indicated by arrows. This model is also known as *Bayesian network*.

Definition 3.0.2. Undirected graphical model, also known as *Markov random field*, is a model in which the links have no directional significance.

3.1 Markov Networks

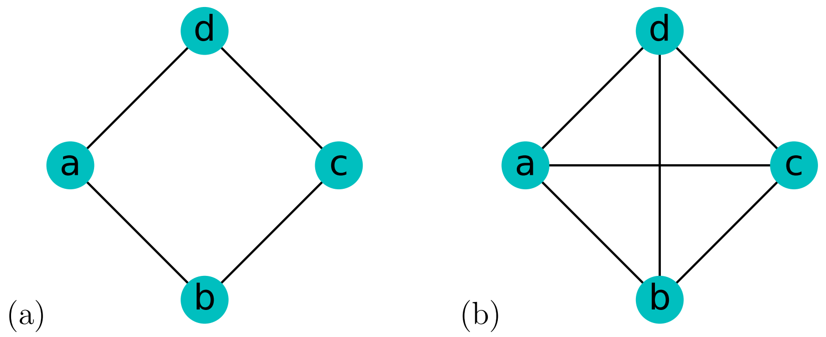


Figure 3.1: Markov networks

(a): $\phi(a,b)\phi(b,c)\phi(c,d)\phi(d,a)/Z_1$
(b): $\phi(a,b,c,d)/Z_2$

There are different kinds of factorization of the joint probability distribution:

$$p(a,b,c) = p(a|b,c)p(b|c)p(c) \tag{3.1}$$

$$p(a,b,c) = \frac{1}{Z}\phi(a,b)\phi(b,c) \tag{3.2}$$

Definition 3.1.1. A potential $\phi(x)$ is a non-negative function of the variable x, $\phi(x) \geq 0$. A joint potential $\phi(x_1, ..., x_n)$ is a non-negative function of the set of variables. A distribution is a special case of a potential satisfying normalization, $\sum_x \phi(x) = 1$. This holds similarly for continuous variables, with summation replaced by integration.[9]

As for a distribution, the ordering of the variables in the potential function is not relevant.

For defining the Markov network, we first need to define a *clique*:

Definition 3.1.2. A **clique**(C) is a set of nodes in a graph that form a complete graph.

Definition 3.1.3. A **maximal** clique is a clique that couldn't be extended by more nodes.

Definition 3.1.4. Consider a set of variables $\chi = \{x_1, ..., x_n\}$, a Markov network is defined as a product of potentials on subsets of the variables $\chi_i \subseteq \chi$:

$$p(x_1, x_2, ..., x_N) = \frac{1}{Z} \prod_C \phi_i(\chi_i). \tag{3.3}$$

The constant Z ensures the distribution is normalized. This is represented by an undirected graph G where $\chi_i; i = 1, ..., C$ being the maximal cliques of the graph G.

3.2 Factor graphs

In order to solve inference problems, it can be helpful to convert directed and undirected graphs into so-called factor graphs. Factor graphs make the decomposition of the joint distribution more clear on the graphical model by introducing additional nodes called the factor nodes. These nodes are usually represented by squares.

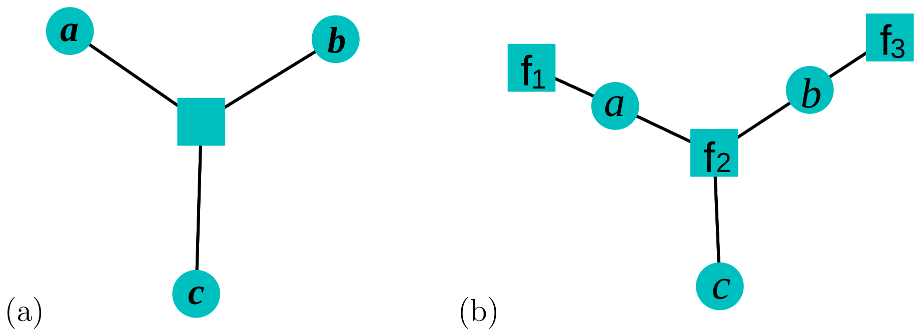


Figure 3.2: Factor graphs

Definition 3.2.1. Factor graphs describe the factorization of functions and are not necessarily related to probability distributions. Consider the function:

$$f(x_1, ..., x_n) = \prod_i \psi_i(\chi_i). \tag{3.4}$$

The factor graph has a node for each factor ψ_i and each variable x_j. There is an undirected link between factor ψ_i and variable x_j for each $x_j \in \chi_i$.

Now, consider the representation of the following distribution:

$$p(x_1, ..., x_n) = \frac{1}{Z} \prod_i \psi_i(\chi_i) \tag{3.5}$$

A normalization constant $Z = \sum_\chi \prod_i \psi_i(\chi_i)$ is assumed and χ represents all variables in the distribution.

Some factors $\psi_i(\chi_i)$ may represent a conditional distribution. For these factors we may use directed links from the parents to the factor node, and a directed link from the factor node to the child. This representation has the same structure as an undirected factor graph, but preserves the information that the factors are distributions.

Factor graph is a **bipartite graph** because it has two sets of nodes and no links between the nodes which are in the same set.

Consider the graph on the figure bellow:

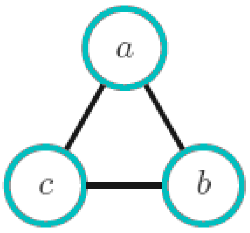


Figure 3.3: Markov network

This Markov network can easily be represented with the factor graph. It doesn't necessarily have to be a unique representation as we're going to see on the Figure 3.4

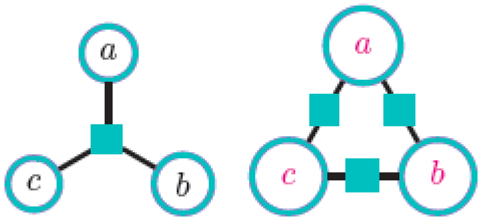


Figure 3.4: Representation of the Markov network shown in Figure 3.3 with factor graphs

Consider the distribution:

$$p(a, b, c) = \phi(a, b)\phi(a, c)\phi(b, c) \tag{3.6}$$

Markov Network representation from the Figure 3.4 could also correspond to an unfactored clique potential $\phi(a, b, c)$. In this sense, the factor graph in the Figure 3.3 right, which corresponds to the clique potential $\phi(a, b)\phi(a, c)\phi(b, c)$ more accurately conveys the form of the distribution in the Equation 3.6. An unfactored clique potential $\phi(a, b, c)$ is also represented by the Figure 3.3 left. Therefore, different factor graphs can have the same Markov network.

3.3 Belief propagation algorithm on factor graphs

3.3.1 Marginal inference and message passing

Definition 3.3.1. (*Marginal inference*) Inference is the process of computing functions of the distribution $p(x_1, ..., x_n)$. Marginal inference is the inference on a subset of variables, possibly conditioned on another subset.

A key concept in efficient inference is message passing. Consider now the chain graphical model, the four variable Markov chain, figure 3.5

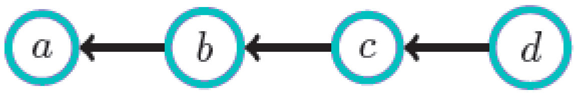


Figure 3.5: Four variable Markov chain

$$p(a, b, c, d) = p(a|b)p(b|c)p(c|d)p(d). \tag{3.7}$$

Our task is to calculate the marginal $p(a)$. Assume that each variable is binary, it only takes values from the set $\{0, 1\}$.

$$\begin{aligned} p(a = 0) = &\sum_{b\in\{0,1\},c\in\{0,1\},d\in\{0,1\}} p(a = 0, b, c, d) = \\ = &\sum_{b\in\{0,1\},c\in\{0,1\},d\in\{0,1\}} p(a = 0|b)p(b|c)p(c|d)p(d) \end{aligned} \tag{3.8}$$

We could calculate this by summing the 8 possible states of variables, however there is a more efficient way. We push the summation over variable d as far to the right as we can:

$$p(a = 0) = \sum_{b\in\{0,1\},c\in\{0,1\}} p(a = 0|b)p(b|c)\gamma_d(c), \tag{3.9}$$

$$\gamma_d(c) = \sum_{d\in\{0,1\}} p(c|d)p(d), \tag{3.10}$$

$\gamma_d(c)$ is a two state potential.

Next, we push the summation over c as far to the right as possible so that we obtain:

$$p(a=0) = \sum_{b\in\{0,1\}} p(a=0|b)\gamma_c(b), \tag{3.11}$$

$$\gamma_c(b) = \sum_{c\in\{0,1\}} p(b|c)\gamma_d(c). \tag{3.12}$$

This procedure is called *variable elimination*. This can also be viewed as passing a message to a neighbouring node on the graph. We can calculate any variable marginal of any tree by starting at the leaf of the tree.

3.3.2 BP algorithm

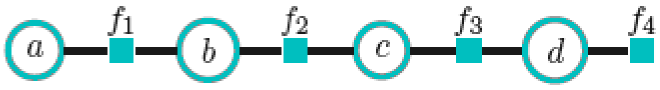


Figure 3.6: A tree without branches

Both directed and undirected networks can be represented using factor graphs. Because of this it is useful to derive a marginal inference algorithm for factor graphs.

Consider the distribution:

$$p(a,b,c,d) = f_1(a,b)f_2(b,c)f_3(c,d)f_4(d), \tag{3.13}$$

for which we want to compute the marginal $p(a,b,c)$. Variable d only occurs locally, so we have:

$$\begin{aligned} p(a,b,c) = \sum_d p(a,b,c,d) = \sum_d f_1(a,b)f_2(b,c)f_3(c,d)f_4(d) = \\ = f_1(a,b)f_2(b,c)\mu_{d\to c}(c), \end{aligned} \tag{3.14}$$

$$\mu_{d\to c}(c) = \sum_d f_3(c,d)f_4(d), \tag{3.15}$$

where $\mu_{d\to c}(c)$ defines a message from node d to node c and is a function of the variable c. Similarly,

$$p(a,b) = \sum_c p(a,b,c) = f_1(a,b)\mu_{c\to b}(b) \tag{3.16}$$

and

$$\mu_{c\to b}(b) = \sum_c f_2(b,c)\mu_{d\to c}(c). \tag{3.17}$$

For simple structures with no branching, messages from one variable to another are sufficient. But, in more complex structures, it is useful to consider also the messages from variables to factors and vice-versa.

Definition 3.3.2. A message schedule is a specified sequence of message updates. A valid schedule is a schedule in which a message is sent from a node only when that node has received all requisite messages from its neighbours. In general, there is more than one valid updating schedule [9].

In BP algorithm messages are updated as a function of incoming messages. The new message is computed based on the previously computed message, until all messages from all factors to variables and vice-versa have been computed.

Consider the distribution:

$$p(\chi) = \frac{1}{Z} \prod_f \phi_f(\chi_f). \tag{3.18}$$

Messages from leaf node factors are initialised to the factor. Messages from leaf variable nodes are set to unity [9].

Variable to Factor message:

$$\mu_{x \to f}(x) = \prod_{g \in \partial x \setminus f} \mu_{g \to x}(x) \tag{3.19}$$

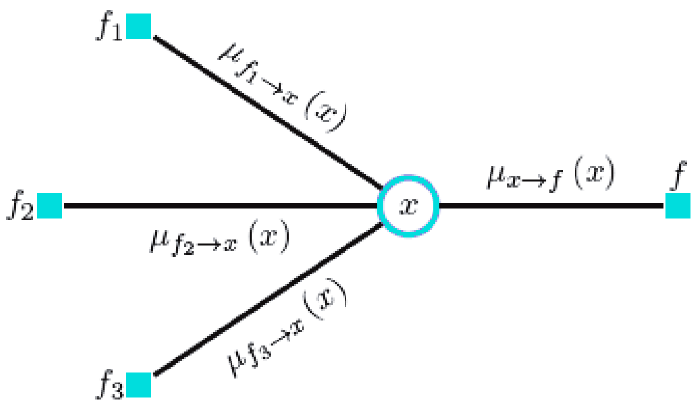


Figure 3.7: Variable to Factor

Factor to Variable message

$$\mu_{f\to x}(x) = \sum_{\chi_f \setminus x} \phi_f(\chi_f) \prod_{y\in\partial f\setminus x} \mu_{y\to f}(y) \tag{3.20}$$

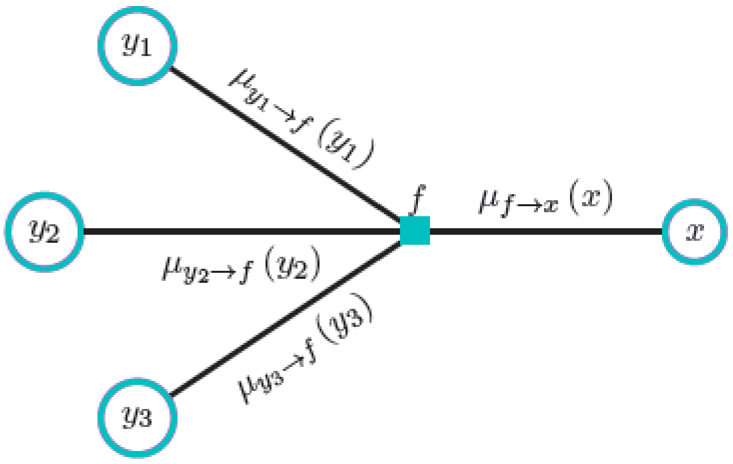


Figure 3.8: Factor to Variable messages

Marginal

$$p(x) \propto \prod_{f\in\partial x} \mu_{f\to x}(x) \tag{3.21}$$

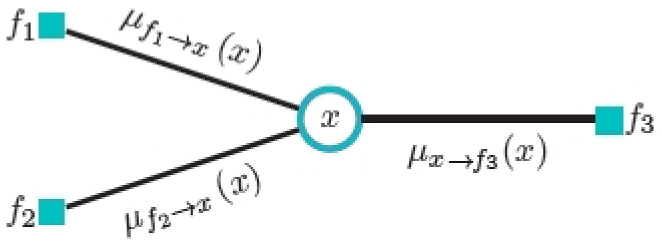


Figure 3.9: Marginal

For marginal inference, the important information is the relative size of the message states so that we may re-normalize messages as we wish. Since the marginal will be proportional to the incoming messages for that variable, the normalization constant is trivially obtained using the fact that the marginal must sum to 1. However, if we wish to also compute any normalization constant using these messages, we cannot normalize the messages since this global information will then be lost [9]. The small example of marginal inference is given on a Figure 3.9.

Chapter 4

Algorithm

Assume we have a probabilistic model on $\vec{x} = (x_1, ..., x_n) \in \chi^N$, where χ is a discrete set, which can be decomposed, such as

$$P(\vec{x}) \propto \prod_{a \in F} f_a(\vec{x}), \tag{4.1}$$

where each f_a only depends on the variables V_a. We can express constraint satisfaction problems in these kinds of models, where each f_a is associated with some constraint.

Crucial problem in computer science is finding good enough assignments to constraint satisfaction problems (CSPs), meaning finding values $\vec{x}$ in the support of $P(\vec{x})$. Assume we knew the value of $P(x_1 = 1)$ was greater than 0. Then we would know that some satisfying assignment where $x_1 = 1$ exist. Having that in mind, we could recursively try to find $\vec{x}$ in the support of $P(1, x_2, ..., x_n)$, and iteratively come up with a good assignment to our CSPs. Actually, we could sample uniformly from the distribution as follows: assign x_1 to 1 with probability $P(x_1 = 1)$, else assign it to 0 . Then, we iteratively sample from $P(x_2|x_1)$ for the model where x_1 is fixed to the value we assigned to it and repeat until we have assigned values to all of the $\{x_i\}_i$. A logical question arises: When can we try and efficiently compute the marginals

$$P(x_i) := \sum_{\vec{x} - x_i} P(\vec{x}) \tag{4.2}$$

for each i?

A popular algorithm for this problem exists when the corresponding graphic model of $P(\vec{x})$ is a tree. BP is only guaranteed to work for trees, but still we hope that it will give a useful answer if our factor graph is only "tree like".

4.1 Deriving BP

We will start by making two simplifying assumptions on our model $P(\vec{x})$. First, we will assume that $P(\vec{x})$ can be written in the form

$$P(\vec{x}) \propto \prod_{(i,j)\in E} f_{i,j}(x_i, x_j) \prod_{i\in[n]} f_i(x_i)$$

for some functions $\{f_{i,j}\}_{i,j}, \{f_i(x_i)\}_i$ and some "edge set" E (where the edges are undirected) 4.1. That is, we will only consider pairwise constraints. We will see later that this naturally corresponds to a physical interpretation, where each of the "particles" x_i interact mutually via pairwise forces. Belief propagation still works without this assumption, but the pairwise case is all we need for the stochastic block model.

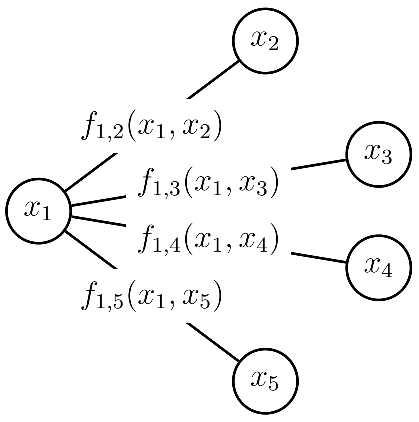


Figure 4.1: Graphical model associated with the pdf $P(x_1, x_2, x_3, x_4, x_5) \propto f_{1,2}(x_1, x_2)f_{1,3}(x_1, x_3)f_{1,4}(x_1, x_4)f_{1,5}(x_1, x_5)f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_4)f_5(x_5)$

The second assumption we need to notice that there is a natural correspondence between $P(\vec{x})$ and the graphical model T on n vertices, where (i, j) forms an edge in T if and only if and only if $(i, j) \in E$. In order words, edges (i, j) in T correspond to factors of the form $f_{i,j}$ in $P(\vec{x})$, and vertices in T correspond to variables in $\vec{x}$. Our second assumption is that the graphical model T is a tree.

Now, imagine we are given such a tree T which represents our probabilistic model. How do we compute the marginals? Suppose that we arbitrarily rooted our tree at vertex x_i. If we could compute the marginals of the children of x_i, we could stitch them together to compute the marginal x_i. In order words, in our graphical model, we should think about computing the marginals of roots of subtrees. A quick check shows that the base case is easy: Let's say we are given a graphical model which is a tree consisting of a single node x_i. This corresponds to some PDF $P(\vec{x}) = P(x_i) \propto f_i(x_i)$. So to compute $P(x_i)$, have just have to compute the marginalizing constant $Z = \sum_{x_i \in \chi} f_i(x_i)$, and then we have $P(x_i) = \frac{1}{Z} f(x_i)$. With the base case out of the way, we can try to solve the induction step: given a graphical model which is a tree rooted at x_i, and where we are given the marginals of the subtrees rooted at the children of x_i, how do we compute the marginal of the tree rooted at x_i? At the figure 4.2 we can see the graphic representation of this. To formalize the induction step, we will define some notation that will be useful to us later on. The main pieces of notation are $T^{i \to j}$, which is the subtree rooted atx_i with parent x_j, and the "messages" $\psi_{x_i}^{k \to i}$, which can be thought of as information which is passed from the child subtrees of x_i to the vertex x_i in order to compute the marginals correctly.

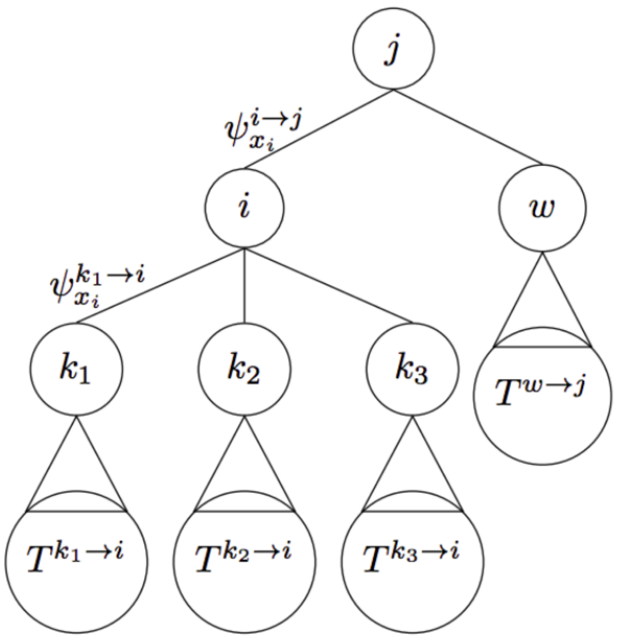


Figure 4.2: Message passing graph [12]

- We let ∂i denote the neighbors of vertex i in T. In general, we will switch between vertex i and the variable x_i represented by vertex i.

- We define $T^{i \to j}$ to be the subtree of T rooted at i, where i's parent is j. We need to specify i's parent to give an orientation to our tree (so we'd know in which direction to do recursion down the tree). Likewise, we let $V_{i \to j}$ be the set of vertices/variables which occur in subtree $T^{i \to j}$.

- Let the function $T^{i\to j}(V_{i\to j})$, which is a function of the variables in $V_{i\to j}$, be equal to the product of $T^{i\to j}$'s edges and vertices. Specifically, $T^{i\to j}(V_{i\to j}) = \prod_{(a,b)\in E'} f_{a,b}(x_a, x_b) \prod_{a\in V_{i\to j}} f_a(x_a)$, where E' is the set of edges which occur in subtree $T^{i\to j}$. We can think of $T^{i\to j}(V_{i\to j})$ as being the pdf of the "model" of the subtree $T^{i\to j}$.

- At last, we define $\psi^{i\to j}_{x_i} := \frac{1}{Z^{i\to j}} \sum_{V_{i\to j}-x_i} T(V_{i\to j})$, where $Z^{i\to j}$ is a normalizing constant chosen such that $\sum_{x_i\in\chi} \psi^{i\to j}_{x_i} = 1$. In particular, $\psi^{i\to j}_{x_i} : \chi \to \mathbb{R}$ is a function defined for each possible value of $x_i \in \chi$. We can interpret $\psi^{i\to j}_{x_i}$ in two ways: as the marginal of the root of the subtree $T^{i\to j}$. And we can think of it as a "message" from vertex i to vertex j. Later on we will see that this is a valid interpretation and why.

Now, let's see how we can express the marginal of the root of a tree as a function of the marginals of its subtrees. Imagine we are considering the subtree $T^{i\to j}$, so that vertex i has children $(\partial i) - j$. Then we can compute the marginal $\psi^{i\to j}_r$ directly:

$$\begin{aligned}
\psi^{i\to j}_{x_i} &\propto \sum_{V_{i\to j}-i} T^{i\to j}(V_{i\to j}) \\
&= \sum_{(\partial i)-j} \sum_{V_{i\to j}-i-\partial i} f_i(x_i) \prod_{k\in(\partial i)-j} f_{i,k}(x_i, x_k) T^{k\to i}(V_{k\to i}) \\
&= f_i(x_i) \sum_{(\partial i)-j} \prod_{k\in(\partial i)-j} f_{i,k}(x_i, x_k) \sum_{V_{k\to i}-k} T^{k\to i}(V_{k\to i}) \\
&\propto f_i(x_i) \sum_{(\partial i)-j} \prod_{k\in(\partial i)-j} f_{i,k}(x_i, x_k) \psi^{k\to i}_{x_k} \\
&= f_i(x_i) \prod_{k\in(\partial i)-j} \sum_{x_k\in\chi} f_{i,k}(x_i, x_k) \psi^{k\to i}_{x_k}
\end{aligned}$$

In the equation above we've got the first row by the definition of marginal probability. Next we broke up sums according to definition of $T^{i\to j}(V_{i\to j})$. Then, we swapped sums and products then again use definiton of marginal probability. At the last row, we again swapped sums with products.

As we look at the update formula we have derived, we can now realise why the $\{\psi^{k\to i}_{x_k}\}_{k\in\partial i-j}$ are called "messages" **to** vertex i: they forward information about the child subtrees to their parent i.

All of the above is algebraic way of deriving belief propagation. A way that is maybe more intuitive to get the same result is following: imagine fixing the value of $x_i = a$ in the the subtree $T^{i\to j}$, and then drawing from each of the marginals of the children of x_i conditioned on the value $x_i = a$. We can consider the marginals of each of the children independently, because the children are independent of each other when conditioned on the value of x_i. Converting words to equations, this means that if x_i has children $x_{k_1}, ..., x_{k_d}$, then the marginal probability of $(x_i, x_{k_1}, ..., x_{k_d})$ in the subtree $T^{i\to j}$ is proportional to $f_i(x_i)\prod_{\substack{k:(i,k)\in E,\\ k\neq j}} \psi_{x_k}^{k\to i} f_{i,k}(x_i, x_k)$. We can then write

$$\begin{aligned}\psi_{x_i}^{i\to j} &\propto \sum_{(\partial i)-j} f_i(x_i) \prod_{\substack{k:(i,k)\in E,\\ k\neq j}} \psi_{x_k}^{k\to i} f_{i,k}(x_i, x_k)\\ &= f_i(x_i) \prod_{\substack{k:(i,k)\in E,\\ k\neq j}} \sum_{x_k\in\chi} \psi_{x_k}^{k\to i} f_{i,k}(x_i, x_k)\end{aligned}$$

And we get back to what we had before. This equation we'll call the "update" or the "message passing" equation. The key assumption was that if we condition on x_i, then the children of x_i are independent. It's useful to have this in mind when thinking about how BP behaves on more general graphs.

A similar calculation yields that we can calculate the marginal of our original probability distribution $\psi_{x_i}^{i} := P(x_i)$ as the marginal of the subtree with no parent, i.e.

$$\psi_{x_i}^{i} \propto f_i(x_i) \prod_{k:(i,k)\in E} \sum_{x_k\in\chi} \psi_{x_k}^{k\to i} f_{i,k}(x_i, x_k)$$

However, instead of computing every $\psi_{x_i}^{i\to j}$ neatly with recursion, we might try something else: let's instead randomly initialize each $\psi_{x_i}^{i\to j}(x_i \in \chi)$ with anything we want. Then, let's update each $\psi_{x_i}^{i\to j}$ in parallel with our update equations. We will keep doing this in successive steps until each $\psi_{x_i}^{i\to j}$ has converged to a fixed value. By viewing belief propagation as a recursive algorithm, it's easy to see that all of the $\psi_{x_i}^{i\to j}$'s will have their correct values after at most d steps. This can be because (after arbitrarily rooting our tree at any vertex) the leaves of our recursion will be set to the correct value after 1 step. After two steps, the parents of the leaves will be updated as functions of the leaves, and they will have the correct values as well. Specifically:

Proposition: Imagine we initialize messages $\psi_{x_i}^{i\to j}$ arbitrarily and update them in parallel according to our update equations. If T has diameter d, then after d steps each$\psi_r^{i\to j}$ converges, and we recover the correct marginals.

By computing everything in parallel in steps instead of recursively, we are computing a lot of "garbage" updates we never use. However, the advantage is, that this procedure is now well defined for general graphs. Suppose $P(\vec{x})$ violated assumption (2), so that the corresponding graph were not a tree. Then we could still try to compute the messages $\psi_{x_i}^{i\to j}$ with parallel updates. We are also able to do this in a local "message passing" kind of way, which may be more intuitive. Maybe, the messages will converge after a reasonable amount of iterations. And maybe, ideally, they will converge to something which gives us information about the marginals $\{P(x_i)\}_i$. In fact, we will see that is exactly what happens in the stochastic block model.

At the end of Chapter 2, we stated that the case of $P(\{q_i\}_i)$ being invariant under permutations of the labeling $\{q_i\}_i$ must be considered with caution. In community detection, when all the group sizes $n_1, ..., n_q$ are equal the right thing to do would be to continue using the marginals, but only after we took care of the symmetry of the problem by randomly fixing certain values of the vertices to have certain labels. There's a way belief propagation does this implicitly: remember that we start belief propagation by randomly initializing the messages. We use the random initialization of the messages so we can "break the symmetry" of the problem.

4.2 Belief Propagation

We saw from the Chapter 2 that in order to maximize the correlation of the labeling we come up with, we should pick the labeling which maximize the marginals of P. So we have some marginals that we want to compute. Now we are going to apply BP to this problem in the "sparse" regime where $c_{a,b} = Np_{a,b} = \mathcal{O}(1)$ (other algorithms, like approximate message passing, can be used for "dense" graph problems). Suppose we are given a random graph with edge list E. What does graph associated with our probabilistic model look like? Well, in this case, every variable is connected to every other variable because $P(\{q_i\}_i)$ includes a factor $f_{i,j}(x_i, x_j)$ for every $(i,j) \in [n] \times [n]$, so we have a complete graph. Never the less, some of the connections between variables are much weaker. In full, our BP update equations are:

$$\begin{aligned}\psi_{t_i}^{i\to j} =& (\text{approximately } \mathcal{O}(n^2) \text{ factors of order } 1 - \mathcal{O}(\tfrac{1}{N}) \text{ from non-edges})\\ &\times (\text{approximately } \mathcal{O}(n) \text{ factors "close to 0" from edges})\end{aligned}$$

$$= \frac{1}{Z^{i\to j}} n_{t_i} \left(\prod_{\substack{(i,k)\notin E\\ k\neq j}} \left[\sum_{t_k \in [q]} \left(1 - \frac{c_{t_1,t_k}}{N}\right) \psi_{t_k}^{k\to i} \right] \right) \times \left(\prod_{\substack{(i,k)\in E\\ k\neq j}} \left[\sum_{t_k\in[q]} c_{t_i,t_k} \psi_{t_k}^{k\to i} \right] \right)$$

$$= \frac{1}{Z^{i\to j}} n_{t_i} \prod_{\substack{(i,k)\notin E\\ k\neq j}} \left[1 - \frac{1}{N} \sum_{t_k\in[q]} c_{t_1,t_k} \psi_{t_k}^{k\to i} \right] \prod_{\substack{(i,k)\in E\\ k\neq j}} \left[\sum_{t_k\in[q]} c_{t_i,t_k} \psi_{t_k}^{k\to i} \right]$$

Likewise

$$\psi_{t_i}^{i} = \frac{1}{Z^{i}} n_{t_i} \prod_{(i,k)\notin E} \left[1 - \frac{1}{N} \sum_{t_k\in[q]} c_{t_1,t_k} \psi_{t_k}^{k\to i} \right] \prod_{(i,k)\in E} \left[\sum_{t_k\in[q]} c_{t_i,t_k} \psi_{t_k}^{k\to i} \right]$$

we want to approximate these equations so that we only have to pass messages along the edges E, instead of the complete graph. This will make our analysis simpler, and allow the more efficient running of the belief propagation algorithm. The first observation is: Imagine we have two nodes $j, j' \in [N]$ such that $(i,j), (i,j') \notin E$. Then we see that $\psi_{t_i}^{i\to j} = \psi_{t_i}^{i\to j'} + \mathcal{O}\left(\frac{1}{N}\right)$, since the only difference between these two variables are two factors of order $(1 - \mathcal{O}\left(\frac{1}{N}\right))$ which appear in the first product of the BP equations. Thus, we basically send the same messages to non-neighbors j of i in our random graph. In general though, we have:

$$\approx \frac{1}{Z^i} n_{t_i} \prod_{k \in [N]} \left[1 - \frac{1}{N} \sum_{t_k \in [q]} c_{t_1,t_k} \psi_{t_k}^{k \to i} \right] \prod_{\substack{(i,k) \in E \\ k \neq j}} \left[\sum_{t_k \in [q]} c_{t_i,t_k} \psi_{t_k}^{k \to i} \right]$$

$$\approx \frac{1}{Z^i} n_{t_i} e^{-h_{t_i}} \prod_{k \in (\partial i) - j} \left[\sum_{t_k \in [q]} c_{t_i,t_k} \psi_{t_k}^{k \to i} \right]$$

The first approximation comes from dropping non-edge constraints on the first product, and is reasonable because we expect the number of neighbors of to be constant. We have also defined a variable

$$h_{t_i} := \frac{1}{N} \sum_{k \in [N]} \sum_{t_k \in [q]} c_{t_i,t_k} \psi_{t_k}^{k \to i}$$

and we have used the approximation $e^{-x} \approx 1 - x$ for small x. The term h_{t_i} is so called "auxiliary external field". We will use this approximation of the BP equation to find solutions for community detection problem. The advantage of this is that the computation time is $\mathcal{O}(|E|)$ instead of $\mathcal{O}(N^2)$, so we can also apply BP to large sparse graphs (which is computationally challenging). At the same time we see how a large dense graphical model with sparse edges still behaves like a sparse tree from the perspective of Belief Propagation. Furthermore, we are hoping that the BP equations will converge and give us good approximations of the marginals. We will only consider factored block models, which are considered to have "hard" setting of parameters. These models satisfy the condition that each community has the same average degree c. We require

$$\sum_{d=1}^{q} c_{a,d} n_d = \sum_{d=1}^{q} c_{b,d} n_d = c$$

A meaningful conclusion for this setting of parameters is that

$$\psi_{t_i}^{i \to j} = n_{t_i}$$

is always a fixed point of the BP equations, which is known as a factored fixed point which can be confirmed by inserting the fixed-point conditions into belief propagation equations. When BP reaches such a fixed point, we get overlap is equaling 0 ($Q = 0$) and the algorithm fails. Still, we can hope that if we randomly initialize $\{\psi^{i \to j}\}_{(i,j) \in E}$, then BP can converge to some non-trivial fixed point which would give some information about the original labeling of the vertices.

Chapter 5

Algorithm implementation

5.1 Implementation of BP in Python

Now that we know all the theory behind, we can start implementing Belief Propagation community detection algorithm in Python programming language.

First we will implement the algorithm on the synthetically made small network of two cliques. We do this just to be sure that the algorithm works on something that can be easily concluded by just looking at it.

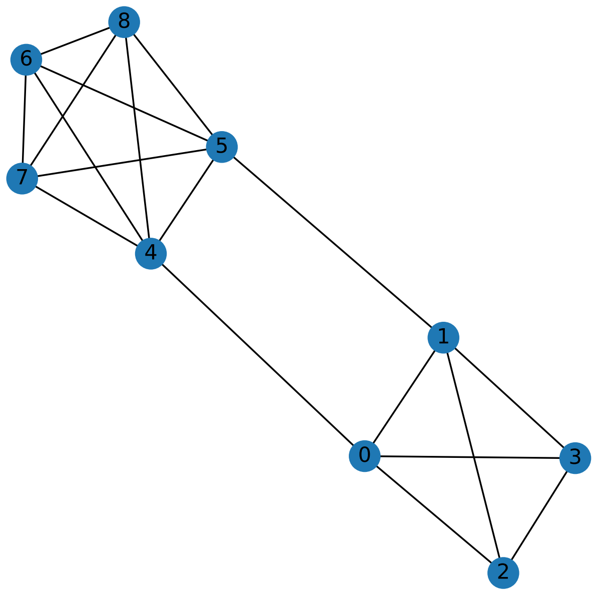


Figure 5.1: Synthetic network of two cliques

On the figure 5.1 we can see how our network looks like. We start by picking at random parameters for the stochastic block model.

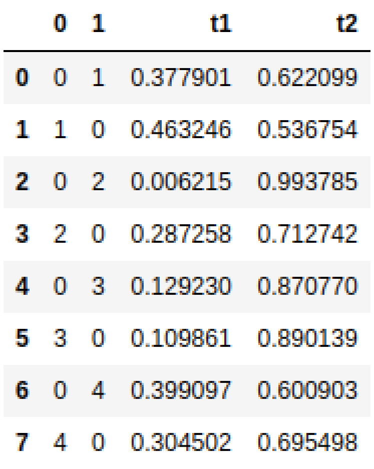
$$q = 2, n_a = 0.5, n_b = 0.5, c_{ab} = \begin{bmatrix} 0.6 & 0.1 \\ 0.1 & 0.3 \end{bmatrix} * 9 \tag{5.1}$$

In the equation 5.1 q is a number of communities we are trying to detect, n_a and n_b are fractions of nodes in communities a and b respectively and c_{ab} is affinity matrix multiplied by the number of nodes, which is in our case 9. Next, we generate random messages between neighbors.

```
E=Z.get_edgelist()
E1=np.asarray(E)
B=E1[:,::-1]
np.random.seed(1)
e=[]
for i in range(E1.shape[0]):
    e.append(E1[i])
    e.append(B[i])
edges=pd.DataFrame(e)
for i in range(1,q+1):
    edges['t'+str(i)]=np.random.rand(edges.shape[0])
b=[]
for i in range(edges.shape[0]):
    b.append(edges.iloc[i,-q:]*(1/np.sum(edges.iloc[i,-q:])))
edges.iloc[:,-q:]=pd.DataFrame(b)
```

Listing 5.1: Generation od random messages between the neighbor nodes

This will give us the following table.



	0	1	t1	t2
0	0	1	0.377901	0.622099
1	1	0	0.463246	0.536754
2	0	2	0.006215	0.993785
3	2	0	0.287258	0.712742
4	0	3	0.129230	0.870770
5	3	0	0.109861	0.890139
6	0	4	0.399097	0.600903
7	4	0	0.304502	0.695498

Figure 5.2: Table of random edge messages

As shown in the table on the figure 5.2, sum of messages that node i is sending to its neighbor j needs to be equal to 1. This is because node i must belong to 1 of q communities. In other words node i is sending with what probability it is belonging to each of the communities.

Next, we repeat the same procedure for the nodes that are not connected by edge in our network.

```python
ne=list(nx.non_edges(H))
nE1=np.asarray(ne)
nB=nE1[:,::-1]
ne=[]
for i in range(nE1.shape[0]):
    ne.append(nE1[i])
    ne.append(nB[i])
nedges=pd.DataFrame(ne)
for i in range(1,q+1):
    nedges['t'+str(i)]=np.random.rand(nedges.shape[0])
b=[]
for i in range(nedges.shape[0]):
    b.append(nedges.iloc[i,-q:]*(1/np.sum(nedges.iloc[i,-q:])
  ))
nedges.iloc[:,-q:]=pd.DataFrame(b)
```

Listing 5.2: Generation of random messages between the non-neighbor nodes

Now that we have randomly generated the messages we can start by first part of the algorithm, inferring the group assignment. Our algorithm will keep iterating until it achieves convergence of the messages.

```python
def BP(edges, nedges, cab, n):

    old=pd.DataFrame(np.zeros(edges.iloc[:,-q:].shape))
    retries=0
    while (retries <= 200) and not (abs(edges.iloc[:,-q]-old.
  iloc[:,-q]) < 0.0000001).all():
        #For neighbours
        ln=[]
        for i in range(len(list(Z.vs))):
            psi=np.ones(q)
            for j in set(edges.loc[edges[1]==i].index):
                psi = np.multiply(psi,np.dot(cab,np.asarray(
  edges.iloc[j,-q:]))) #product of matrices cab and vectors
  of messages
            ln.append(psi)
        #nonedge
        l=[]
        for i in range(len(list(Z.vs))):
            psi=np.ones(q)
            for j in set(nedges.loc[nedges[1]==i].index):
                psi = np.multiply(psi,np.dot(1-cab/len(list(Z.vs)),np.asarray(nedges.iloc[j,-q:])))
            l.append(psi)
        L=np.multiply(ln,l)
        #All together
        M_i=np.multiply(L,n)
        Zmi=np.sum(M_i,axis=1)
        M=(np.multiply(M_i,np.tile(1/Zmi, (q, 1)).T))
        Psi_marginal=pd.DataFrame(M)
        Psi_marginal['q_i']=M.argmax(axis=1)
        Psi_i=Psi_marginal

        psi_i_j=[]
        for i in range(len(list(Z.vs))):
            for j in set(edges.loc[edges[1]==i].index):
                psi_i_j.append(np.multiply(L[i],1/np.dot(cab,(edges.iloc[j,-q:]))))
        Psi_i_j=np.multiply(n,psi_i_j)

        Zi_j=np.sum(Psi_i_j,axis=1)
        P=(np.multiply(Psi_i_j,np.tile(1/Zi_j, (q, 1)).T))

        old=pd.DataFrame(edges.iloc[:,-q:])
        edges.iloc[:,-q:]=P


        for i in range(len(list(Z.vs))):
            for j in set(nedges.loc[nedges[0]==i].index):
                nedges.iloc[j,-q:]=np.asarray(Psi_i)[i,0:q]
        retries += 1
    return Psi_i
```

Listing 5.3: BP algorithm - calculating the messages between the nodes

Since we are dealing with a trivial network, we get results even without applying the second part of the algorithm. After calculating the messages with the same parameters of the stochastic block model we began with, we plot the network with its communities 5.3.

After our messages converge with the criteria of 0.0000001, we can proceed to the next step of our algorithm, learning the parameters of the stochastic block model.

$$c_{aa} = \frac{M_{aa}}{\frac{N_a*(N_a-1)}{2}} * N, c_{ab} = \frac{M_{ab}}{N_a * N_b} * N, \tag{5.2}$$

In equation 5.2 m_{ab} is the number of edges between groups a and b. Next block of code is related to learning these parameters. The convergence criteria is again 0.0000001

```
edges['e1']=0
edges['e2']=0
for i in range(len(list(Z.vs))):
    for j in set(edges.loc[edges[0]==i].index):
        edges.iloc[j,-2]=Psi_i['q_i'][i]
for i in range(len(list(Z.vs))):
    for j in set(edges.loc[edges[1]==i].index):
        edges.iloc[j,-1]=Psi_i['q_i'][i]

count = np.zeros((q,q))
comb= list(itertools.product(range(q), repeat=2))

for i in range(edges.shape[0]):
    for j in range(len(comb)):
        if edges['e1'][i]==comb[j][0] and edges['e2'][i]==
    comb[j][1]:
            count[comb[j][0]][comb[j][1]]+=1
c= np.zeros((q,q))

for i in range(len(comb)):

    if comb[i][0]==comb[i][1]:
        if N[comb[i][0]] == 0 or N[comb[i][0]] == 1:
            c[comb[i][0]][comb[i][1]] ==0
        else:
            c[comb[i][0]][comb[i][1]]=count[comb[i][0]][comb[
    i][1]]/(N[comb[i][0]]*(N[comb[i][1]]-1)/2) * len(list(Z.vs
    ))
    else:
        if N[comb[i][0]] == 0 or N[comb[i][1]] == 0:
            c[comb[i][0]][comb[i][1]] ==0
        else:
            c[comb[i][0]][comb[i][1]]= count[comb[i][0]][comb
    [i][1]]/(N[comb[i][0]]*(N[comb[i][1]]))*len(list(Z.vs))
```

Listing 5.4: Caltulating the stochastic block model parameters in Python

Since, again, we are dealing with a very trivial network, BP achieves convergence of SBM parameters after only 2 iterations. Final results and group assignments are presented on the pictures bellow, equation 5.3 and figure 5.3 respectively.

$$n = \begin{bmatrix} 0.55555556 \\ 0.44444444 \end{bmatrix} c_{ab} = \begin{bmatrix} 0.9 & 0.9 \\ 0.9 & 9.0 \end{bmatrix} * 9 \tag{5.3}$$

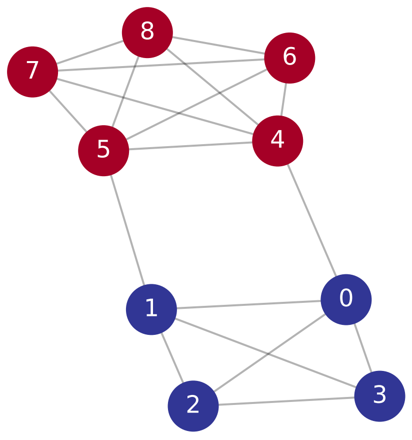


Figure 5.3: Network after inferring group assignment

We have seen, from the results given above, that BP had no problem inducing the parameters of SBM and assigning vertices to communities. Now, we move to a more complex problem.

5.2 Zachary's karate club

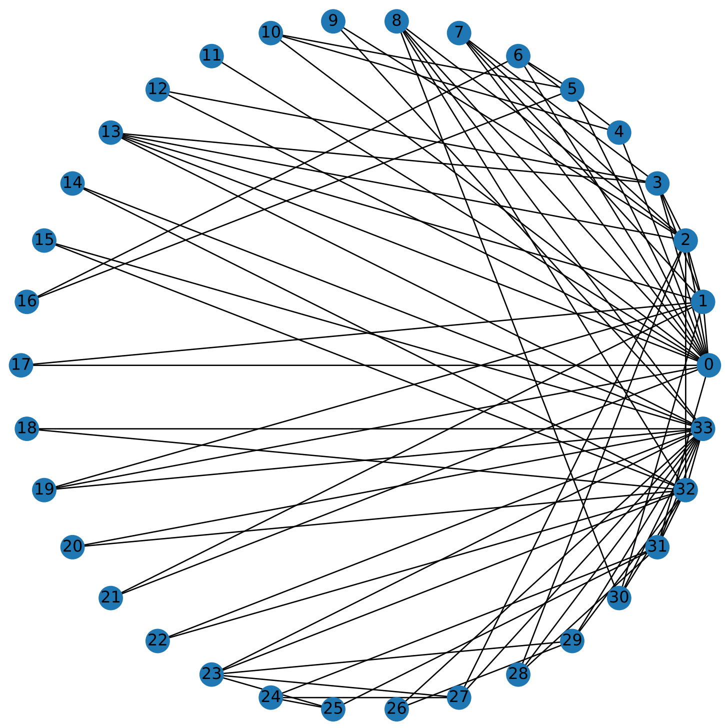


Figure 5.4: Zachary's karate club social network

So far we've seen how efficient as our algorithm on a small, trivial network. Now we will test it on bigger, more complicated network. Network we'll be using is called Zachary's karate club [13].

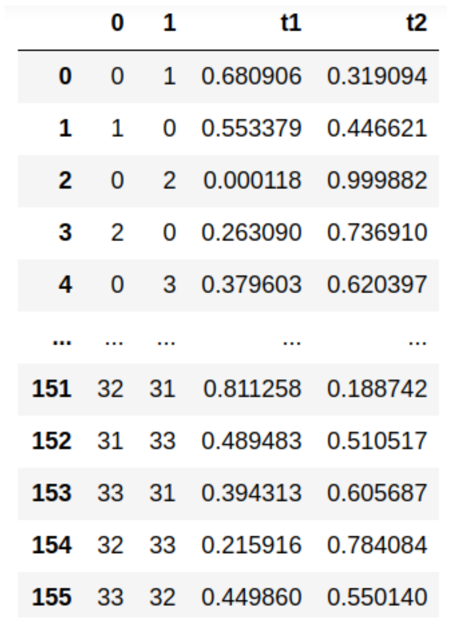
Zachary's karate club is a social network representing university karate club. The network is constructed of 34 members of a karate club (34 vertices), and 78 links that represent the friendship between the members. This network is depicted on the figure 5.4. Vertex 0 in a network represents the instructor, while vertex 33 stands for the president of the karate club.

In case when we initialize the algorithm with $q = 2$ communities, we can observe this as an analysis of which karate club members will center around the president of the club and which will center around the instructor. Let's start.

Again, we start by random selection of the parameters for the stochastic block model 5.4.

$$q = 2, n_a = 0.5, n_b = 0.5, c_{ab} = \begin{bmatrix} 0.3 & 0.01 \\ 0.01 & 0.3 \end{bmatrix} * 34 \tag{5.4}$$

Next, we move to generation of random messages between neighbor and non-neighbor nodes using the same code we used in the example with synthetic network. Resulting table is given on the figure 5.5.



	0	1	t1	t2
0	0	1	0.680906	0.319094
1	1	0	0.553379	0.446621
2	0	2	0.000118	0.999882
3	2	0	0.263090	0.736910
4	0	3	0.379603	0.620397
...	...	...	...	...
151	32	31	0.811258	0.188742
152	31	33	0.489483	0.510517
153	33	31	0.394313	0.605687
154	32	33	0.215916	0.784084
155	33	32	0.449860	0.550140

Figure 5.5: Zachary's karate club neighbor random messages

We can now start with the first part of BP, inducing community assignments. This time, our code is running longer than the last because we are dealing with a bigger network.

```
old=pd.DataFrame(np.zeros(edges.iloc[:,-q:].shape))
retries=0
while (retries <= 200) and not (abs(edges.iloc[:,-q]-old.iloc[:,-q]) < 0.0000001).all():
    #For neighbours
    ln=[]
    for i in range(len(list(Z.vs))):
        psi=np.ones(q)
        for j in set(edges.loc[edges[1]==i].index):
            psi = np.multiply(psi,np.dot(cab,np.asarray(edges.iloc[j,-q:])))
        ln.append(psi)
    #nonedge
    l=[]
    for i in range(len(list(Z.vs))):
        psi=np.ones(q)
        for j in set(nedges.loc[nedges[1]==i].index):
            psi = np.multiply(psi,np.dot(1-cab/len(list(Z.vs)
    ),np.asarray(nedges.iloc[j,-q:])))
        l.append(psi)
    L=np.multiply(ln,l)
    #All together
    M_i=np.multiply(L,n)
    Zmi=np.sum(M_i,axis=1)
    M=(np.multiply(M_i,np.tile(1/Zmi, (q, 1)).T))
    Psi_marginal=pd.DataFrame(M)
    Psi_marginal['q_i']=M.argmax(axis=1)
    Psi_i=Psi_marginal

    psi_i_j=[]
    for i in range(len(list(Z.vs))):
        for j in set(edges.loc[edges[1]==i].index):
            psi_i_j.append(np.multiply(L[i],1/np.dot(cab,(
    edges.iloc[j,-q:]))))
    Psi_i_j=np.multiply(n,psi_i_j)

    Zi_j=np.sum(Psi_i_j,axis=1)
    P=(np.multiply(Psi_i_j,np.tile(1/Zi_j, (q, 1)).T))

    old=pd.DataFrame(edges.iloc[:,-q:])
    edges.iloc[:,-q:]=P


    for i in range(len(list(Z.vs))):
        for j in set(nedges.loc[nedges[0]==i].index):
            nedges.iloc[j,-q:]=np.asarray(Psi_i)[i,0:q]
    retries += 1
```
Listing 5.5: Calculating the messages for Zachary's carate club network

5.3 Results and discussion

After 50 iterations BP reaches convergence of the messages, meaning the difference between the iterations gets close to 0. Before we perform the second part of the algorithm, calculating parameters $c_{a,b}$ and n_a, n_b let's take a look at our network 5.6. As we can see from the figure this already looks pretty good. This is because of our initial choice of SBM parameters. Let's move on with the second part of BP. After a number of iterations BP achieves convergence and get first set of parameters for the BP 5.5. We are now left with 18 members of karate club belonging to one of the communities(e.g.

centered around president) and 16 to another. Since in Python counting starts from 0 the instructor is represented by node0 and the president by the node 33. Having this in mind we can clearly see from the figure 5.6 that our communities center around vertices 0 and 33.

$$n = \begin{bmatrix} 0.51389995 \\ 0.48610005 \end{bmatrix} c_{ab} = \begin{bmatrix} 8.000000 & 1.298611 \\ 1.298611 & 8.783333 \end{bmatrix} \tag{5.5}$$

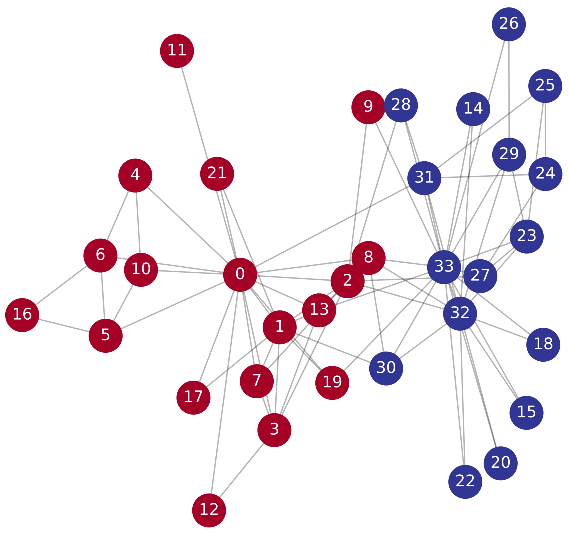


Figure 5.6: Zachary's karate club community assignment

Now, we'll see how BP behaves with different initialization. Our initialization parameters are represented in the equation 5.6.

$$q = 2, n_a = 0.7, n_b = 0.3, c_{ab} = \begin{bmatrix} 0.7 & 0.5 \\ 0.5 & 0.1 \end{bmatrix} * 34 \tag{5.6}$$

After BP convergence we get results 5.7. As we can conclude from this, with different initialization parameters we get completely different community assignments of the vertices with now 30 members of the club belonging to first community and only 4 to the second 5.7.

$$n = \begin{bmatrix} 0.88242956 \\ 0.11757044 \end{bmatrix} c_{ab} = \begin{bmatrix} 17.0 & 11.90000 \\ 11.9 & 2.57931 \end{bmatrix} \tag{5.7}$$

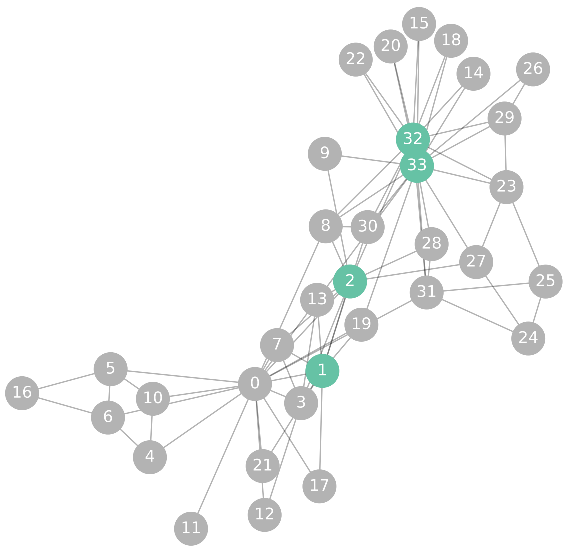


Figure 5.7: Zachary's karate club community assignment disassortative initialization parameter

At the end let's take a look at the figure 5.8. Fixed point "(i)" corresponds to the case of assortative network structure, meaning $c^i_{ab} > c^i_{aa}$. On the other hand, second fixed point "(ii)" splits the nodes based on their degree, putting nodes with the higher degree in one group, and the nodes with lower degrees in the other group, that is putting president and the instruction at the same group.

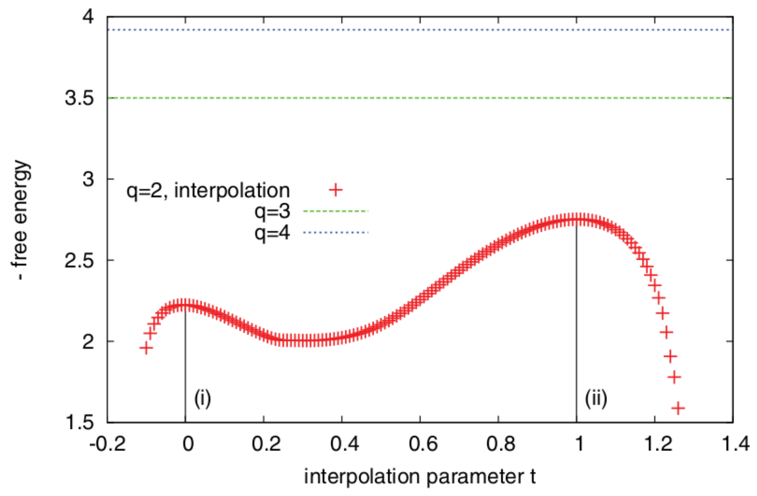


Figure 5.8: The likelihood of the partitions when $q = 2$ [1]

Although counter intuitive this second division is not an accident, it splits the nodes into "leaders" and "students". We plot the negative free energy 5.8 achieved by interpolating between the two fixed points according to a parameter t:

$$c_{ab}(t) = (1-t)c_{ab}^{(i)} + tc_{ab}^{(ii)}$$

We did not discuss the free energy in this paper, but it could be considered to represent the likelihood of the partitions. The higher the value of free energy, the higher the likelihood of the partition. As we can see from the figure 5.8, dissassortative structure ($c_{ab} < c_{aa}$) represented by the fixed point "(ii)" is the more likely division.

$$f_{BP}(q, \{n_a\}, \{c_{ab}\}) = -\frac{1}{N}\sum_i logZ^i + \frac{1}{N}\sum_{(i,j)\in E} logZ^{ij} - \frac{c}{2} \tag{5.8}$$

Conclusion

One of the most important aspects of the network is presence of community structure. Group organizations can be found all around us, such as families and friendships, villages and cities, nations, etc. We can also observe different kinds of groups and communities in the nature, for example proteins that have the same specific function in a cell, predators and prays, etc.

We have implemented a belief propagation algorithm to detect these kinds of organizations in the family of sparse graphs. We considered a case where we synthetically made the network. Simple network with 2 cliques with sparse edges between one another. We initialized the algorithm with random stochastic block model parameters and random messages between the connected nodes and nodes that are not connected by an edge. BP converged almost immediately and we were able to graphically see the results where one of the cliques was in one of the communities and second clique in another community. This experiment was to make sure our implementation works.

We then moved to a more complex problem. A well known network for detecting communities, Zachary's karate club. This network is a non tree graph with sparse edges. It contains 34 nodes and 78 edges. It represents the university karate club where nodes are students, instructor and a president of the club and the edges are friendship between these members. With 2 different initialization parameters where number of communities was set to 2 we were able to obtain two different structures of the network. One was of assortative structure and other disassortative. This could be explained as follows: in the case of assortative network we have a situation where the club was divided between those centered around the instructor and those centered around the president of the club. This is more intuitive for us. But dissassortative structure is dividing the club members between the leaders and the students, counting the number of connections of each node, meaning our instructor and the president would belong to the same community.

The last thing we did, we analyzed the likelihood of network assignments of the Zachary's karate club network. The more likely assignment of the groups is the one where we get dissassortative structure of the network, and by this we conclude our work.

When looking at the real world data, this approach would be a useful clustering algorithm. We would only need to find a way of representing our data by a network.

Bibliography

[1] Aurelien Decelle, Florent Krzakala, Cristopher Moore and Lenka Zdeborová, *Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications*, Physical Review E 84, 066106, (2011).

[2] Christopher M. Bishop, *Pattern recognition and Machine Learning*, Chapter 8, Springer Science+Business Media, LLC (2006).

[3] Ernesto Estrada and Philip A. Knight, *A first Course in Network Theory*, Chapter 2, OXFORD University Press (2015).

[4] A. Clauset, C. Moore, and M. Newman, *Hierarchical structure and the prediction of missing links in networks*, Nature 453, 98 (2008).

[5] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, *Machine Learning Research* **9**, 1981 (2008).

[6] B. Karrer and M. E. J. Newman, *Stochastic blockmodels and community structure in networks*, Physical Review E 83, 016107 (2011).

[7] A. Condon, R.M. Karp, *Algorithms for graph partitioning on the planted partition model*, Random Struct. Algor. 18 (2001) 116–140

[8] M. E. Dyer and A. M. Frieze, *The solution of some random NP-hard problems in polynomial expected time*, J. Algorithm 10, 451 (1989)

[9] David Barber *Bayesian Reasoning and Machine Learning*, Chapter 4, (2010)

[10] Zhao Yang, René Algesheimer and Claudio J. Tessone *A Comparative Analysis of Community Detection Algorithms on Artificial Networks*, Scientific Reports, (2016)

[11] A. Clauset *Generative models for Complex Network Structure*, NetSci (2013)

[12] https://windowsontheory.org/2018/10/20/belief-propagation-and-the-stochastic-block-model/

[13] W. W. Zachary, J. Anthropol *An Information Flow Model for Conflict and Fission in Small Groups* Res. 33, 452 (1977).

[14] https://en.wikipedia.org/wiki/Zachary27s_karate_club

[15] Jess Banks, Cristopher Moore, Joe Neeman, Praneeth Netrapalli, *Information-theoretic thresholds for community detection in sparse networks* AJMLR: Workshop and Conference Proceedings vol 49:1–34, 2016.

[16] Cristopher Moore, *The Computer Science and Physics of Community Detection: Landscapes, Phase Transitions, and Hardness*, 2017.

[17] Andrea Montanari, Federico Ricci-Tersenghi, Guilhem Semerjian, *Clusters of solutions and replica symmetry breaking in random k-satisfiability*, 2008.

Biography

Jelena Mihajlović was born on the 9th of April 1992 in Novi Sad. She finished elementary school "Jovan Jovanović Zmaj" in Sremska Kamenica and the Gymnasium "Isidora Sekulić" in Novi Sad. She received her BSc degree in Applied Mathematics in 2016 from the Faculty of Sciences, University of Novi Sad, Serbia and she continued her Master studies in the field of Data Science at the same faculty. In December 2018, Jelena got her first job as a Data Scientist in "Continental Automotive", and as of January 2020, she is working in "Node.io".