

Community Detection in Large-Scale Time-Varying Networks

A Modularity Based Approach

Dissertation presented by
Antoine ASPEEL

for obtaining the Master's degree in
engineer in applied mathematics

Supervisor
Jean-Charles DELVENNE

Readers
Michael FANUEL, Marco SAERENS

Academic year 2016-2017

Acknowledgments

First and foremost, I want to sincerely thank my supervisor Professor Jean-Charles Delvenne for his availability and the time spent on the many conversations we had during this year. His insightful advice and his intuition have been invaluable.

I would also like to thank Michael Fanuel and Professor Marco Saerens for agreeing to spend time in reading my thesis.

A very special thank goes to my friends, from Ardennes and elsewhere, who supported me during the writing of this thesis, especially to Kad, Jonas and Carol who offered me valuable advice during the proofreading of my thesis.

Last but not least, I would like to warmly thank my family for its unconditional support and encouragements during my whole academic cursus.

Contents

Introduction	1
1 State of the art	5
1.1 Graphs	5
1.2 Communities	5
1.3 Modularity	6
1.4 Relaxation	8
1.5 Summary	11
2 Embedding dimension	13
2.1 Visualization of a solution	13
2.2 Interpretation of a solution	14
2.3 Theoretical results	15
2.4 Numerical results	21
2.5 Summary	25
3 Static algorithms	27
3.1 Node by node algorithm	27
3.2 Total update algorithm	35
3.3 Riemannian trust-region algorithm	38
3.4 Discrete static algorithm	41
3.5 Summary	43
4 Dynamic algorithms	45
4.1 Fuzzy dynamic algorithm	46

4.2	Discrete dynamic algorithm	46
4.3	Summary	49
5	Comparison of the algorithms	51
5.1	Static algorithms	53
5.2	Discrete static algorithm	55
5.3	Discrete dynamic algorithm	60
5.4	Summary	64
6	Further work	67
6.1	Discussion	67
6.2	An extended notion of communities	68
	Conclusion	71
	Bibliography	73
A	Theoretical results and proofs	79
A.1	Proofs	79
A.2	Example of degenerated solution	83
A.3	Convergence results of the Riemannian trust-region algorithm	85
B	Pseudocode	87
B.1	Node by node algorithm	87
B.2	Total update algorithm	89
B.3	Fuzzy dynamic algorithm	91
B.4	Discrete dynamic algorithm	91

Notations

In this section, we present the notations used in this thesis.

Generally, vectors are written in bold lower case ($\mathbf{x}$) and matrices are written in bold upper case ($\mathbf{X}$).

The notations are listed below.

$\mathcal{G}$	Undirected graph
V	Set of vertices
E	Set of edges
n	Number of nodes
m	Number of edges (taking account their weights)
$\mathbf{A}$	Adjacency matrix (size $n \times n$)
$\mathbf{d}$	Vector of degrees (size $n \times 1$)
$\mathbf{B}$	Modularity matrix (size $n \times n$)
$\mathbf{L}$	Membership matrix for the discrete problem (Pd) (size $n \times k$)
$\mathbf{H}$	Membership matrix for the relaxed problem (P) (size $n \times k$)
$\mathbf{X}_{i,:}$	i -th row of matrix $\mathbf{X}$
$\mathbf{X}_{:,j}$	j -th column of matrix $\mathbf{X}$
Q_k	Modularity function
$\mathbf{0}_{mn}$	Null matrix of size $m \times n$
δ_{ij}	The Kronecker's delta (1 if $i = j$ and 0 otherwise)
$\langle \cdot, \cdot \rangle$	Euclidean inner product
$\ \cdot \ $	Euclidean norm
$\ \cdot \ _F$	Frobenius norm
$Tr(\cdot)$	Trace operator
k	The embedding dimension
$\delta(\mathcal{G})$	Minimal embedding dimension of the graph $\mathcal{G}$
c	Number of communities
$\mathbb{S}^{k-1}$	unit $(k - 1)$ -dimensional sphere
i or j	Index, generally from 1 to n
l	Index, generally from 1 to k
$:=$	Definitional equality

Introduction

In 1735, the famous mathematician Leonhard Euler solved the *Seven Bridges of Königsberg Problem*, a puzzle that asks how to walk in the streets of the city of Königsberg by crossing all the seven bridges of this city once and only once and finish at its starting position. His answer was that there is no solution to this problem [Alexanderson, 2006]. More surprising than the answer to this question are the tools used to answer it. This event lays the foundations of graph theory.

In the 21st century, graph theory is taught in most universities in engineering or in mathematics. It is also a current domain of research as a lot of questions are waiting for an answer and many problems cannot yet be solved efficiently.

Graph theory is a powerful tool to model various real-world problems such as finding the shortest path from a point to another [Dreyfus, 1969], crucial in the operation of a GPS; planning school schedule while taking into account availability of teachers, classes and classrooms [Sharma and Pandey, 2011]; finding the distribution of the tasks in a group of people that satisfies a maximum of people [Munkres, 1957]; etc. The number of such examples is huge and they span fields as varied as physics, biology, sociology and mathematics. The French mathematician Henri Poincaré said once “*La mathématique est l’art de donner le même nom à des choses différentes*”.¹ This is particularly obvious with graph theory: a unique mathematical object - a graph also called a network - can represent many various realities.

Informally, a graph is a mathematical representation of a set of objects, and the links between pairs of them. The objects are called the nodes or the vertices of the graph. The links are its edges. Figure 1 represents a graph. Each dot is one node and the links between them are the edges. They indicate which nodes are related to each other. The nodes can for example represent a group of 12 people that are in a same party, the edges indicating who spoke to whom.

¹In English: “*Mathematics is the art of giving the same name to different thing*”.

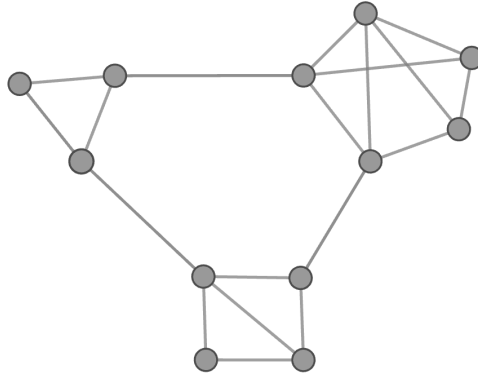


Figure 1: An example of graph.

Another important example is the webgraph. This graph describes the hyperlinks between pages of the world wide web. The nodes are the pages and the edges are the hyperlinks. This graph is used every day when someone makes research on Google [Page et al., 1999].

Facebook’s friendships graph depicted in figure 2 illustrates how the 21st century is marked by the growing importance of big data. The graphs become larger and larger, to the extent that dealing with them is challenging in terms of running time. To face this reality, all the algorithms developed in this thesis have as objective to be suitable for *large-scale* graphs, meaning that they have to be efficient.



Figure 2: Facebook’s friendships graph.

Each graph contains information about the situation of which it is the mathematical representation. An important task in applied mathematics is to make this information accessible. A graph has many properties, we focus on its *community structure*. Informally, a community is a set of nodes densely connected that are weakly connected with nodes outside of the community [Girvan and Newman, 2002].

A community structure is frequently observed in social networks - as Facebook -

induced by family, circle of friends, sport team, etc. [Girvan and Newman, 2002]. Revealing this structure can help to achieve a better understanding of human behavior. In the webgraph, communities correspond to groups of web pages that treat a same subject. In a network representing the links between buyers and products in an online website, the detection of communities enables the development of recommendation systems [Reddy et al., 2002]. In another area, a graph can be associated to any digitized image. Consider one node by pixel and add an edge if two neighboring pixels have similar color. In such a graph, community detection can help to recognize patterns - as objects - in the image. This variety of applications explains why community detection in graphs has already been studied a lot [Fortunato, 2010].

In this thesis, our interest is focused on *time-varying* graphs. These graphs have the property to evolve in time. If we look at the graph at two distinct moments, it can be different. Note that all the examples presented in the previous paragraph can be seen as time-varying graphs: someone adds a new friend on Facebook, a new webpage is created, an online purchase is ordered by a customer and finally if an image changes progressively, it is a video. The real-world networks often have a very fast evolution [Aktunc et al., 2015].

Actually, there is no community detection method that is satisfactory in all situations. More particularly, the case of time-varying networks has never been deeply studied. Most of the time, it is considered as a succession of static networks that are treated independently to each other. This approach which does not exploit the dynamic character of the situation is probably sub-optimal in terms of running time. An improvement could make community detection possible on graphs that evolve too fast for the static approach. The goal of this thesis is to fill this void by designing an efficient community detection algorithm in the case of large-scale time-varying networks.

Our approach is based on the maximization of a quality function called the modularity. The procedure is composed of two different steps: (i) we solve a particular relaxation of the modularity maximization problem and (ii) we use a clustering algorithm to recover a solution of the unrelaxed problem. All the algorithms designed here are iterative which give them the desired dynamic property. When the graph changes a bit, we expect that only few iterations will be required to recover the new communities.

In the first chapter, the problem is introduced and rigorously formalized. The relaxation that is studied involves hyperspheres.

Chapter 2 discusses the natural dimension of the hyperspheres that appears in the relaxation. Both theoretical and numerical results are presented. In addition, the behavior of the solutions is studied.

Chapter 3 presents four algorithms to solve the relaxed problem and proposes a community detection algorithm for the case of a static graph.

Chapter 4 presents a set of dynamic algorithms to solve the problem of tracking communities in time-varying networks. The idea is to use a static algorithm and to recover a solution of the initial problem from the relaxed solution.

Chapter 5 compares the algorithms with an existing efficient one and draws a verdict.

Chapter 6 opens new horizons for further works and proposes an alternative point of view on the notion of community structure.

Finally, the conclusion recalls the main results of this work.

Chapter 1

State of the art

There are different approaches to solve the problem of finding communities in a network. We focus here on one based on modularity. This section presents what has already been done on this subject and formally introduces the problem which is the subject of this thesis.

1.1 Graphs

The adjacency matrix of an undirected graph $\mathcal{G}$ is denoted by $\mathbf{A}$ and assumed to be symmetric with non negative entries.

For a graph $\mathcal{G} = (V, E)$, V is the set of vertices and E is the set of edges. The number of vertices is denoted by $n = |V|$ and the weighted number of edges is denoted by $m = \frac{1}{2} \sum_{i,j} \mathbf{A}_{ij}$ in the light of the handshake lemma.

We note $\mathbf{d}_i$ the weighted degree of the node i of $\mathcal{G}$, it is $\mathbf{d}_i = \sum_j \mathbf{A}_{ij}$.

1.2 Communities

The notion of community is intuitive. For example, the communities represented by colored ellipses in figure 1.1 seem natural. Paradoxically, it is quite difficult to formalize this notion mathematically and no definition is unanimous [Fortunato, 2010].

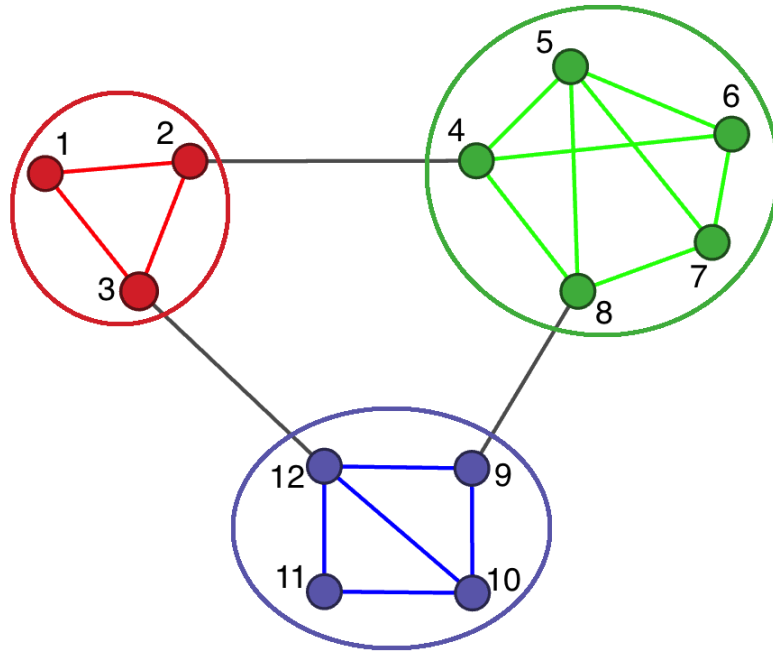


Figure 1.1: Graph of figure 1 with its community structure emphasized by colored ellipses. The nodes are numbered.

One would like the following properties for a *good* community structure [Girvan and Newman, 2002]:

- The set of communities forms a partition of the nodes, i.e. each node is in one and only one community;
- There are a lot of edges inside a community and only some edges between different communities.

1.3 Modularity

The modularity is a function over the set of partitions of the nodes that says *how good* a partition is, in the sense of communities. It was introduced in [Newman and Girvan, 2004] and is the most popular function to evaluate the quality of a partition of nodes into communities [Fortunato, 2010]. More precisely, the modularity says how much the nodes of a same community are connected to each other in comparison to what would be expected in a random graph. This random graph is called the null model. Such an approach comes from the idea that random graphs do not contain any community structure. The modularity compares the proportion of edges between the nodes of the graph and the nodes of the null model. We write $\mathbf{P}_{ij}$ the weight of

the edge between the nodes i and j in the null model. The modularity is

$$\frac{1}{2m} \sum_{i=1}^n \sum_{j=1}^n (\mathbf{A}_{ij} - \mathbf{P}_{ij}) \delta(\mathbf{c}_i, \mathbf{c}_j)$$

where $\mathbf{c}_i$ is the community to which i belongs and $\delta(\mathbf{c}_i, \mathbf{c}_j) = 1$ if $\mathbf{c}_i = \mathbf{c}_j$ and 0 otherwise. In other words, it is 1 if i and j belong to the same community and 0 otherwise. The factor $\frac{1}{2m}$ is a normalization factor.

A partition is *good* in the sense of communities if the number of edges inside a community is greater than the expected number of edges in the null model and then, the greater the modularity, the better the partition.

The choice of $\mathbf{P}_{ij}$ depends on the desired properties of the null model. A common choice is to impose that the expected degree of a node in the null model is the degree of this node in the original network. A similar condition is imposed on the number of edges, we want the expected number of edges in the null model to be the number of edges in the original graph. To have these properties, one can propose to choose m pairs of nodes independently, each node being chosen with a probability proportional to its degree. An edge is added between the nodes that have been selected.

This procedure leads to the following definition of the expected weight of an edge between node i and node j in the null model

$$\mathbf{P}_{ij} := \frac{\mathbf{d}_i \mathbf{d}_j}{2m}.$$

According to this definition, the modularity is given by

$$\frac{1}{2m} \sum_{i=1}^n \sum_{j=1}^n \left(\mathbf{A}_{ij} - \frac{\mathbf{d}_i \mathbf{d}_j}{2m} \right) \delta(\mathbf{c}_i, \mathbf{c}_j). \quad (1.1)$$

Thanks to the normalization factor $\frac{1}{2m}$, this quantity is between -1 and 1 . Note that in general the upper bound is not reached, i.e. in general, the best partition of the nodes gives a modularity strictly smaller than 1 . In fact, it was shown by [Brandes et al., 2007] that a modularity of 1 can never be reached, except in the case of a graph without any edges.

It is natural to define the modularity matrix as²

$$\mathbf{B} = \frac{1}{2m} \left(\mathbf{A} - \frac{\mathbf{d} \mathbf{d}^T}{2m} \right). \quad (1.2)$$

This matrix is symmetric: $\mathbf{B} = \mathbf{B}^T$. Intuitively, the modularity matrix can be seen as a centered version of the adjacency matrix.

The notation $\delta(\mathbf{c}_i, \mathbf{c}_j)$ in formula (1.1) can be replaced by a matrix $\mathbf{L}$ called the membership matrix. It indicates which node is in which community. More precisely,

$$\mathbf{L}_{il} = \begin{cases} 1 & \text{if node } i \text{ is in community } l \\ 0 & \text{otherwise} \end{cases}$$

²Note that another definition is sometimes used in literature where the factor $\frac{1}{2m}$ is missing.

and then each line corresponds to a node and each column corresponds to a community.

If it is known in advance that there is $k < n$ communities, it makes sense to choose $\mathbf{L} \in \mathbb{R}^{n \times k}$ instead of $\mathbb{R}^{n \times n}$.

As example, the membership matrix associated to the community structure presented in figure 1.1 is

$$\mathbf{L} = \begin{pmatrix} 1 & & & & & & & & \\ 1 & & & & & & & & \\ 1 & & & & & & & & \\ & 1 & & & & & & & \\ & 1 & & & & & & & \\ & 1 & & & & & & & \\ & 1 & & & & & & & \\ & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & 1 & & & & & & \\ & & 1 & & & & & & \\ & & 1 & & & & & & \end{pmatrix}.$$

For a given $k \in \mathbb{N}_0$, we define the objective function $Q_k(\mathbf{L})$ of a graph $\mathcal{G}$ for a given $\mathbf{L} \in \mathbb{R}^{n \times k}$ as

$$Q_k(\mathbf{L}) = \text{Tr}(\mathbf{L}^T \mathbf{B} \mathbf{L}) = \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{L}_{il} \mathbf{L}_{jl}$$

where $\text{Tr}(\cdot)$ is the trace operator, the sum of the diagonal elements of a matrix. It is a matrix formulation of the modularity presented in (1.1). Keep in mind that the greater $Q_k(\mathbf{L})$, the better the partition $\mathbf{L}$, in the sense of communities.

The problem of modularity maximization is to find a membership matrix $\mathbf{L} \in \mathbb{R}^{n \times k}$ solution of the following discrete optimization problem:

$$Q_k^d = \max_{\mathbf{L} \in \mathbb{R}^{n \times k}} Q_k(\mathbf{L}) \text{ such that } \begin{cases} \mathbf{L}_{il} \in \{0, 1\} & \text{for } i = 1, \dots, n \text{ and } l = 1, \dots, k \\ \sum_{l=1}^k \mathbf{L}_{il} = 1 & \text{for } i = 1, \dots, n. \end{cases} \quad (\text{Pd})$$

The constraints ensure that $\mathbf{L}$ is a valid membership matrix, i.e. that it defines a partition of the nodes. When k is chosen equal to n , this problem is equivalent to find the partition of the nodes that gives the maximal modularity.

1.4 Relaxation

The problem (Pd) is computationally intractable, meaning that it is NP-hard as shown in [Brandes et al., 2006]. In this situation, a classic approach is to study a relaxation of the problem.

A natural relaxation is to replace the constraint $\mathbf{L}_{il} \in \{0, 1\}$ by $\mathbf{L}_{il} \in [0, 1]$. This

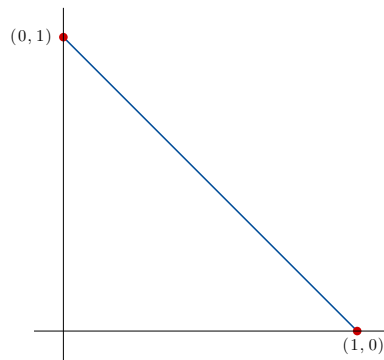
choice is natural in the sense that $[0, 1]$ is the convex hull of $\{0, 1\}$. This problem is

$$\max_{\mathbf{H} \in \mathbb{R}^{n \times k}} Q_k(\mathbf{H}) \text{ such that } \begin{cases} \mathbf{H}_{il} \in [0, 1] & \text{for } i = 1, \dots, n \text{ and } l = 1, \dots, k \\ \sum_{l=1}^k \mathbf{H}_{il} = 1 & \text{for } i = 1, \dots, n. \end{cases} \quad (1.3)$$

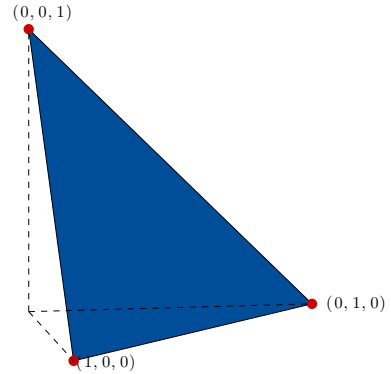
Note that we write $\mathbf{H}$ instead of $\mathbf{L}$ to emphasize the fact that $\mathbf{H}$ does not have 0 or 1 entries. This matrix will also be called the membership matrix.

This problem has already been studied in [Schobbens, 2014], [Vanderlinden, 2015] and [Van Den Eeckhaut, 2016]. The admissible domain of this problem can be identified as the canonical simplex in $\mathbb{R}^k$. Figure 1.2 presents this domain for $k = 2$ and for $k = 3$. One property of this domain is that it has borders and for this reason, the iterative algorithms studied to solve this problem are of the projective methods type:

1. From an iterate $\mathbf{x}^k$, compute a normed direction Δ^k ;
2. Choose a step length α^k ;
3. Set the new iterate as $\mathbf{x}^{k+1} := P(\mathbf{x}^k + \alpha^k \Delta^k)$ where $P(\cdot)$ is a projection operator on the feasible domain.



(a) $k = 2$, 1-simplex, a segment.



(b) $k = 3$, 2-simplex, a triangle.

Figure 1.2: Admissible domain of problem (1.3) for some values of k . Figure reprinted from [Van Den Eeckhaut, 2016, Figure 1.4].

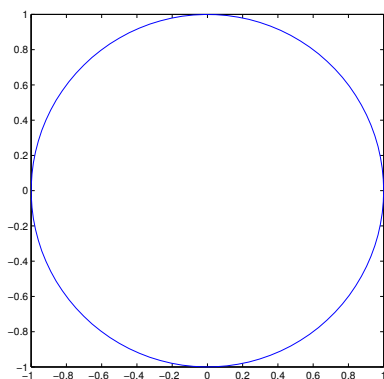
In practice, it is not obvious how to do the projection and this is the reason why we choose to explore an alternative relaxation that avoids the questions of projections.

We define the quadratic constrain relaxation as the following problem:

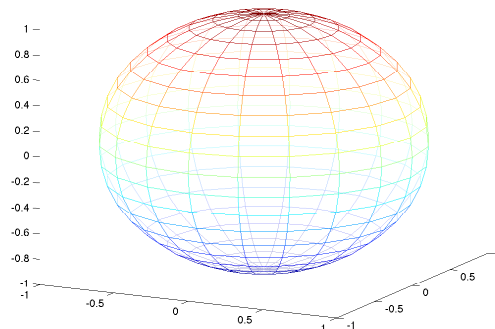
$$Q_k^* = \max_{\mathbf{H} \in \mathbb{R}^{n \times k}} Q_k(\mathbf{H}) \text{ such that } \sum_{l=1}^k (\mathbf{H}_{il})^2 = 1 \text{ for } i = 1, \dots, n. \quad (\text{P})$$

This is a quadratically constrained quadratic program. It was briefly studied by E. Schobbens in her thesis [Schobbens, 2014, Chapter 4].

We observe that $\sum_{l=1}^k (\mathbf{H}_{il})^2 = 1$ if and only if $\|\mathbf{H}_{i,:}\| = 1$. That corresponds to place each row of $\mathbf{H}$ (i.e. each node) on a $(k - 1)$ -dimensional sphere $\mathbb{S}^{k-1} := \{\mathbf{x} \in \mathbb{R}^k : \|\mathbf{x}\| = 1\} \subset \mathbb{R}^k$. It is why the parameter k is called the embedding dimension. Such spheres are depicted in figure 1.3.



(a) $k = 2$, $\mathbb{S}^1$, a unit circle.



(b) $k = 3$, $\mathbb{S}^2$, a usual unit sphere.

Figure 1.3: Admissible domain of problem (P) for some values of k , the embedding dimension.

For n nodes, the admissible set is the Euclidean product of n $(k - 1)$ -dimensional spheres. This domain is written $(\mathbb{S}^{k-1})^n$ which, equipped with an inner product, is a Riemannian manifold (when $k > 1$).³

Note that the admissible domain is compact and that the objective function is continuous, which means that by the extreme value theorem, the maximum exists and the optimum Q_k^* is well-defined.

This relaxation is unusual so we now check that it really is a relaxation of the discrete problem.

Lemma 1. *Problem (P) is a relaxation of problem (Pd). In other words, if $\mathbf{L}$ is a feasible solution of (Pd), then $\mathbf{L}$ is a feasible solution of (P).*

Proof. Let $\mathbf{L}$ be a feasible solution of (Pd). Then it satisfies $\sum_j \mathbf{L}_{ij} = 1$ and $\mathbf{L}_{ij} \in \{0, 1\}$.

Then for each line i , it exists an index $c_i \in \{1, \dots, k\}$ such that $\mathbf{L}_{i,j} = \delta_{i,c_i}$.

Thus $\sum_j \mathbf{L}_{ij}^2 = \sum_j \delta_{i,c_i}^2 = 1$ and then $\mathbf{L}$ is also feasible for (P). $\square$

A direct consequence is the following result.

Result 1. *For any graph and for any $k \in \mathbb{N}_0$, it holds $Q_k^* \geq Q_k^d$.*

³No knowledge about Riemannian manifolds is expected from the reader.

The relaxation (P) looks surprising but has the advantage of not having any border. The fact that it is a Riemannian manifold will allow us to use the tools of Riemannian optimization [Absil et al., 2008].

One can see that this problem is generally non convex and thus it cannot be hope to find an algorithm that will systematically find a global maximum. We focus on the search of local maxima with high objective value. Thankfully, it was shown in [Good et al., 2010] that modularity has the property that, on practical networks, it is observed that a large majority of the local maxima gives an objective value close to the optimal one. This is illustrated by figure 1.4.

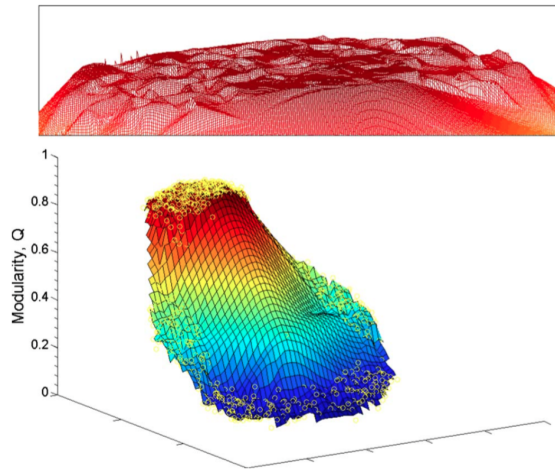


Figure 1.4: Image reprinted from [Good et al., 2010] that illustrates the fact that a big part of local maxima gives high values of modularity.

1.5 Summary

Before closing this section, the roadmap of this work is recalled in order to fix ideas. To track communities in time-varying networks, some iterative algorithms are designed to solve the relaxed problem (P). When a solution $\mathbf{H}$ of this problem have been found, it is converted into a solution $\mathbf{L}$ of the discrete problem (Pd). Such a solution represents the community structure of the graph. The conversion step from the relaxed problem to the discrete one is done using a clustering algorithm.

Chapter 2

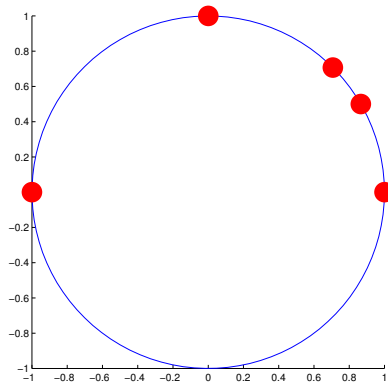
Embedding dimension

The goal of this chapter is to answer the following question : *What is a good value for the embedding dimension k ?* In addition, some ways to visualize and interpret the solutions of the relaxed problem are presented.

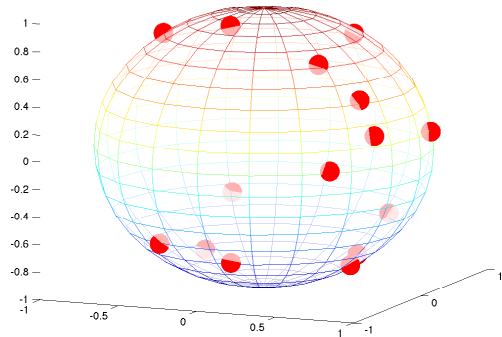
The results presented in this chapter will give some insight into numerical observations of the following chapters.

2.1 Visualization of a solution

This section presents a way to visualize a solution $\mathbf{H}$ of the problem (P). A natural way is to use the fact that each line of $\mathbf{H}$, i.e. each node, lies in $\mathbb{S}^{k-1}$ and then, if the embedding dimension k is 2 or 3, we can plot one point by line of $\mathbf{H}$ on a circle ($k = 2$) or on a usual sphere ($k = 3$). Some examples are depicted on figure 2.1.



(a) $k = 2$, $\mathbb{S}^1$, a circle.



(b) $k = 3$, $\mathbb{S}^2$, a usual sphere.

Figure 2.1: Representations of solutions $\mathbf{H}$ of (P). Each line of $\mathbf{H}$ is represented by a dot.

2.2 Interpretation of a solution

The relaxation occurring from (Pd) to (P) being not fully trivial, an interpretation of the solutions $\mathbf{H}$ needs to be explicitly developed. This requires some observations. A new matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$ is introduced and defined as

$$\mathbf{X} := \mathbf{H}\mathbf{H}^T.$$

With such a definition, one can observe that

$$\mathbf{X}_{ij} = \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} = \langle \mathbf{H}_{i,:}; \mathbf{H}_{j,:} \rangle \quad (2.1)$$

and then this matrix contains the inner products between all pairs of rows of $\mathbf{H}$. This matrix is sometimes called the Gramian matrix.

Firstly, observe that $-1 \leq \mathbf{X}_{ij} \leq 1$, indeed, from (2.1) and using the Cauchy-Schwarz inequality, it holds $|\mathbf{X}_{ij}| = |\langle \mathbf{H}_{i,:}; \mathbf{H}_{j,:} \rangle| \leq \|\mathbf{H}_{i,:}\| \|\mathbf{H}_{j,:}\| = 1$.

In addition, still using relation (2.1), it holds

$$\mathbf{X}_{ii} = \langle \mathbf{H}_{i,:}; \mathbf{H}_{i,:} \rangle = \|\mathbf{H}_{i,:}\|^2 = 1. \quad (2.2)$$

One could ask about the link between those inner products and the angles between the vectors. To answer to this question, the usual definition of the cosine of the angle between two vectors $\mathbf{x}$ and $\mathbf{y}$ is used:

$$\cos(\widehat{\mathbf{x}\mathbf{y}}) := \frac{\langle \mathbf{x}; \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}.$$

In the case of interest, it yields

$$\cos(\widehat{\mathbf{H}_{i,:}; \mathbf{H}_{j,:}}) = \frac{\langle \mathbf{H}_{i,:}; \mathbf{H}_{j,:} \rangle}{\|\mathbf{H}_{i,:}\| \|\mathbf{H}_{j,:}\|} = \langle \mathbf{H}_{i,:}; \mathbf{H}_{j,:} \rangle = \mathbf{X}_{ij}.$$

In a non-formal way, it holds

Nodes i and j are close community-wise if and only if $\mathbf{H}_{i,:}$ and $\mathbf{H}_{j,:}$ are close.

One can consider the limit case where $\mathbf{H}_{i,:} = \mathbf{H}_{j,:}$ for which i and j are *exactly* in the same community. Then when it only has $\mathbf{H}_{i,:} \approx \mathbf{H}_{j,:}$, one can consider that i and j are only *close* community-wise by a continuity argument.

Knowing this, one can say that i and j are close community-wise if and only if the angle $\widehat{\mathbf{H}_{i,:}; \mathbf{H}_{j,:}}$ is small or equivalently if its cosine is close to 1 which is itself equivalent to their inner product being close to 1.

Taking into consideration these observations, it is natural to consider that $\mathbf{X}_{ij}$ indicates whether nodes i and j are close community wise: the greater $\mathbf{X}_{ij}$ is, the closer nodes i and j are.

This leads to another way to visualize a solution, where one assigns a color gradient for values of $\mathbf{X}_{ij}$ from -1 to 1 . The representation obtained for the same solution as 2.1a is depicted on figure 2.2a. The i -th node in the matrix is the i -th leftmost node on figure 2.1a. As indicated by (2.2), we observe that all diagonal values are 1. All entries of the first row except the first one are less than 0 indicating that node 1 is far from all the others community-wise. On the other hand, entries of nodes 3, 4 and 5 form a block of high value indicating that they can form a community. This observation is coherent with figure 2.1a. Note that such patterns appear only if the nodes are well ordered. The figure 2.2b is a shuffled version of 2.2a in which the pattern is hidden. To make appear patterns the rows of $\mathbf{H}$ need to be ordered in counterclockwise (or clockwise) order. Note that this notion is not trivial to adapt when $k > 2$.

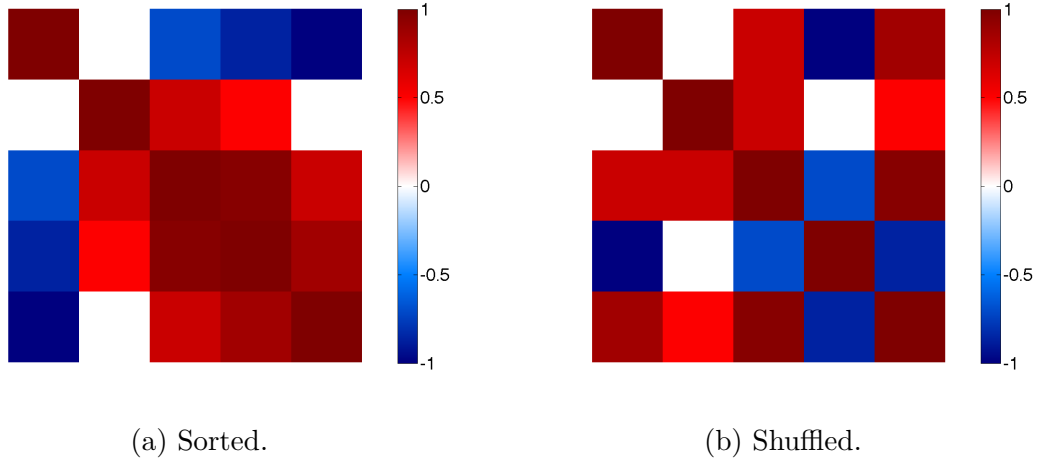


Figure 2.2: Representation of the matrix $\mathbf{X}$ corresponding to matrix $\mathbf{H}$ depicted on figure 2.1a.

2.3 Theoretical results

This section is devoted to the theoretical results that we have been able to prove.

2.3.1 Definitions

The embedding dimension k has been introduced in chapter 1 as the number of communities in the discrete program (Pd) but for the relaxed problem (P) its role is not obvious. We introduce the minimal embedding dimension $\delta(\mathcal{G})$ that can be seen as the *natural* value of k for the graph $\mathcal{G}$. Its definition will be made precise later. If one chooses k too high, the algorithms could be slowed down because there is too much entries in $\mathbf{H}$. On the contrary, if k is chosen too small, the solution obtained can be poor. The objective of this section is to try to shed light on those questions. A theoretical approach is firstly developed and secondly, some numerical experiments are presented.

It looks natural to require that the minimal embedding dimension $\delta(\mathcal{G})$ maximizes $Q_k^*(\mathcal{G})$ with respect to k in order to obtain a modularity as high as possible. That means to find a value of k such that the optimal modularity is maximal. In addition, if more than one value of k give the same value of maximal modularity, the smallest one is preferred. According to this definition, the minimal embedding dimension is the smallest one such that the problem (P) can achieve all its potential. Formally, it can be written for a graph $\mathcal{G}$,

$$\delta(\mathcal{G}) := \min_{k \in \mathbb{N}_0} \operatorname{argmax} Q_k^*(\mathcal{G}). \quad (2.3)$$

Note that it is not obvious that the minimal embedding dimension δ is well-defined. Indeed, one can argue that maybe in some cases, $Q_k^* < Q_{k+1}^*$ for all k and then $\delta = \infty$.

Result 3 shows that $\delta(\mathcal{G}) \leq n$ and then $\delta(\mathcal{G})$ is well-defined. To prove it, we need some lemmas and intermediate results.

The following result reveals a particular property of problem (P).

Result 2. *Problem (P) is invariant under rotations and reflections of rows of $\mathbf{H}$.*

The proof of this result can be found in appendix A.1.1.

Result 2 states that problem (P) is invariant under any linear transformation of the form

$$\mathcal{R} : \mathbb{R}^{n \times k} \rightarrow \mathbb{R}^{n \times k} : \mathbf{H} \mapsto \mathcal{R}(\mathbf{H}) := \mathbf{H}\mathbf{R}^T$$

with $\mathbf{R} \in \mathbb{R}^{k \times k}$ orthogonal, i.e. $\mathbf{R}^T \mathbf{R} = \mathbf{I}$.

Lemma 2. *For any graph and for any $k \in \mathbb{N}_0$, it holds that $Q_k^* \leq Q_{k+1}^*$.*

Appendix A.1.2 presents the proof of lemma 2.

Lemma 3. *For any graph with n nodes and for any $k > n$, $Q_k^* = Q_n^*$.*

It has been chosen to keep the proof of this lemma in the body of the text because it sheds light on how the rank of a solution is linked to the embedding dimension.

Proof. By lemma 2, it holds that $Q_k^* \geq Q_n^*$. It remains to show $Q_k^* \leq Q_n^*$. We proceed by contradiction. Assume $Q_k^* > Q_n^*$. Let $\mathbf{H}^* \in \mathbb{R}^{n \times k}$ be an optimal solution, i.e. $Q_k(\mathbf{H}^*) = Q_k^*$. We define $r := \operatorname{rank}(\mathbf{H}^*)$.

The following definition of the rank of a matrix is used: $r = \dim(\mathcal{L})$ where $\mathcal{L} := \operatorname{span}\{\mathbf{H}_{1,:}^*, \dots, \mathbf{H}_{n,:}^*\}$ is the vector space spanned by the rows of $\mathbf{H}^*$.

One can define an operator $\mathcal{R}(\cdot)$ that rotates a set in $\mathbb{R}^k$ around the origin such that $\mathcal{R}(\mathcal{L}) = \mathbb{R}^r \times \{0\}^{k-r}$. Note that $\mathcal{R}(\mathbb{S}^{k-1}) = \mathbb{S}^{k-1}$ thanks to the symmetry of the sphere. Otherwise, it is not true for spheres of smaller dimension, i.e. in general $\mathcal{R}(\mathbb{S}^{r-1}) \neq \mathbb{S}^{r-1}$. Another property of $\mathcal{R}(\cdot)$ is that for any sets A and B , we have

$\mathcal{R}(A) \cap \mathcal{R}(B) = \mathcal{R}(A \cap B)$. This property follows from the physical interpretation of a rotation.

Using those properties yields

$$\begin{aligned} \mathcal{R}(\mathbb{S}^{k-1} \cap \mathcal{L}) &= \mathcal{R}(\mathbb{S}^{k-1}) \cap \mathcal{R}(\mathcal{L}) \\ &= \mathbb{S}^{k-1} \cap (\mathbb{R}^r \times \{0\}^{k-r}) \\ &= \mathbb{S}^{r-1} \times \{0\}^{k-r}. \end{aligned}$$

Thanks to the constraint of (P) and the definition of $\mathcal{L}$, for any $i \in V$, it holds that

$$\mathbf{H}_{i,:}^* \in \mathbb{S}^{k-1} \cap \mathcal{L}$$

and then

$$\mathcal{R}(\mathbf{H}_{i,:}^*) \in \mathcal{R}(\mathbb{S}^{k-1} \cap \mathcal{L}) = \mathbb{S}^{r-1} \times \{0\}^{k-r}. \quad (2.4)$$

Remember that the problem (P) is invariant under rotations (see result 2), the matrix $\mathcal{R}(\mathbf{H}^*)$ where each row is rotated is also an optimal solution, i.e. $Q_k(\mathcal{R}(\mathbf{H}^*)) = Q_k(\mathbf{H}^*)$.

The matrix $\mathbf{H}_{small} \in \mathbb{R}^{n \times r}$ is defined as the r first columns of $\mathcal{R}(\mathbf{H}^*)$. We show that $Q_r(\mathbf{H}_{small}) = Q_k(\mathcal{R}(\mathbf{H}^*))$. To do this, use the fact that $(\mathcal{R}(\mathbf{H}^*))_{il} = 0$ when $l > r$ as shown in (2.4). One can write

$$\begin{aligned} Q_k(\mathcal{R}(\mathbf{H}^*)) &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^k (\mathcal{R}(\mathbf{H}^*))_{il} (\mathcal{R}(\mathbf{H}^*))_{jl} \\ &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^r (\mathcal{R}(\mathbf{H}^*))_{il} (\mathcal{R}(\mathbf{H}^*))_{jl} \\ &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^r (\mathbf{H}_{small})_{il} (\mathbf{H}_{small})_{jl} \\ &= Q_r(\mathbf{H}_{small}). \end{aligned}$$

Note that $r \leq n$ because r is the rank of $\mathbf{H}^* \in \mathbb{R}^{n \times k}$ and then, using lemma 2 yields $Q_r^* \leq Q_n^*$.

Putting all this together gives $Q_k^* = Q_k(\mathbf{H}^*) = Q_k(\mathcal{R}(\mathbf{H})) = Q_r(\mathbf{H}_{small}) \leq Q_r^* \leq Q_n^*$. It is a contradiction with $Q_k^* > Q_n^*$ and the proof is complete. $\square$

A corollary of this lemma is the well-definedness of $\delta(\mathcal{G})$.

Result 3. *For any graph $\mathcal{G}$ with n nodes, $\delta(\mathcal{G}) \leq n$ and the minimal embedding dimension is well-defined.*

Proof. It follows directly from the definition of the minimal embedding dimension (2.3) and from lemma 3. $\square$

Note that this result is not surprising keeping in mind that the parameter k was initially introduced as the number of communities in the discrete problem (Pd).

Thanks to this result, the following alternative formula for the minimal embedding dimension of $\mathcal{G}$ holds,

$$\delta(\mathcal{G}) = \min_{k \in \{1, \dots, n\}} k \text{ such that } Q_k^*(\mathcal{G}) = Q_n^*(\mathcal{G}).$$

It follows from this relation that $Q_{\delta(\mathcal{G})}^*(\mathcal{G}) = Q_n^*(\mathcal{G})$.

2.3.2 Link with the rank

In linear algebra, the rank of a matrix refers to the dimension of the vector space spanned by its lines. One can hope to construct a link between the minimal embedding dimension and the rank of an optimal solution. A deeper understanding of the solutions of (P) will permit to understand the behavior of algorithms developed in the next chapter.

If a feasible solution $\mathbf{H} \in (\mathbb{S}^{k-1})^n$ of problem (P) lives on a (possibly rotated) $(r-1)$ -dimensional sphere ($r \leq k$), the vector space spanned by its rows is r dimensional and then the rank of the matrix $\mathbf{H}$ is r . This means that the rank of a feasible solution indicates the minimal dimension of the sphere to embed it.

One can look for the relation $\delta(\mathcal{G}) = \text{rank}(\mathbf{H}^*)$ where $\mathbf{H}^*$ is an optimal solution. The objective function and the constraints of (P) being nonlinear, one can expect to find only a pretty weak result. As it will be seen later, such a relation with equality is too strong and is wrong. Instead, a weaker result of the form $\delta(\mathcal{G}) \leq \text{rank}(\mathbf{H}^*)$ is proved.

Result 4. *Let $\mathcal{G}$ be a graph. If $k \geq \delta(\mathcal{G})$, then for any optimal solution $\mathbf{H}^* \in \mathbb{R}^{n \times k}$ of problem (P), it holds that $\delta(\mathcal{G}) \leq \text{rank}(\mathbf{H}^*)$. In addition, there exists some $\mathbf{H}^*$ for which equality holds.*

The proof of this result is similar to the proof of lemma 3. It can be found in appendix A.1.3.

One can observe that the assumption $k \geq \delta(\mathcal{G})$ is automatically fulfilled by setting $k := n$ thanks to result 3.

Degenerated solutions

As mentioned before, equality does not hold all the time in result 4. An example for which $\text{rank}(\mathbf{H}^*) > \delta(\mathcal{G})$ is constructed in appendix A.2. Such optimal solution $\mathbf{H}^*$ is called *degenerated*.

2.3.3 Bounds

It could be interesting to know which range of values of $\delta(\mathcal{G})$ can be expected. Indeed, it would allow fixing a good value of k in our algorithms, i.e. find a good dimension for

the embedding of the graph. More formally, one can ask for bounds on the minimal embedding dimension that depend only on the number of nodes. Such bounds are of the form $f(n) \leq \delta(\mathcal{G}) \leq g(n)$.

Firstly, we show that the lower bound $\delta(\mathcal{G}) \geq 1$ is tight in the sense that there exists graphs with arbitrarily many nodes such that equality holds.

The idea of the proof is to construct a family of graphs with arbitrarily many nodes for which $\delta = 1$. These graphs are the complete graphs with n nodes denoted K_n . We show that an optimal solution for these graphs is to put all lines of $\mathbf{H}$ on the same point of the sphere. This looks natural because in K_n all nodes are linked to each other and then they form one unique community. The proof ends by noting that if an optimal solution is such that all the nodes coincide, it can be embedded in a 0-sphere, i.e. with $k = 1$ and then the minimal embedding dimension is 1.

Lemma 4. *For the complete graph K_n and for any $k \in \mathbb{N}_0$, $Q_k^* = 0$ and $\delta(K_n) = 1$.*

The proof of lemma 4 is presented in appendix A.1.4.

The desired result is then obtained.

Result 5. *For any $n \in \mathbb{N}_0$, there exists a graph $\mathcal{G}$ with n nodes such that $\delta(\mathcal{G}) = 1$.*

Proof. It is a direct corollary of lemma 4. □

One can ask for a similar result for an upper bound of the form $\delta(\mathcal{G}) \leq g(n)$. Indeed, such a result would be helpful to accelerate the algorithms presented in chapter 3 by setting $k := g(n)$. We have such a result but the inequality could not be tight.

Note that result 3 already states that $\delta(\mathcal{G}) \leq n$. This bound can be improved thanks to a result from [Pataki, 1998] and [Barvinok, 1995] (proved independently) that holds in a more general setting.

Result 6. *For any graph $\mathcal{G}$ with n nodes, $\frac{1}{2}\delta(\mathcal{G})(\delta(\mathcal{G}) + 1) \leq n$.*

It is shown in appendix A.1.5 that assumptions of the original theorem apply to problem (P).

To find a more explicit dependency, one can develop

$$\begin{aligned}
 \frac{1}{2}\delta(n)(\delta(n) + 1) &\leq n \\
 \Leftrightarrow \delta(n)(\delta(n) + 1) - 2n &\leq 0 \\
 \Leftrightarrow \delta(n)^2 + \delta(n) - 2n &\leq 0 \\
 \Leftrightarrow \delta(n) &\leq \frac{-1 + \sqrt{1 + 8n}}{2}.
 \end{aligned}$$

Finally, for large values of n , the right hand side of this expression can be simplified

$$\frac{-1 + \sqrt{1 + 8n}}{2} \approx \frac{-1 + \sqrt{8n}}{2} = \sqrt{2n} - \frac{1}{2}.$$

These bounds are depicted on figure 2.3 and illustrate the fact that the minimal embedding dimension grows slowly with the number of nodes of the graph n . In the light of this section, it makes sense to choose an embedding dimension k such that $1 \leq k \leq \frac{-1 + \sqrt{1 + 8n}}{2}$.

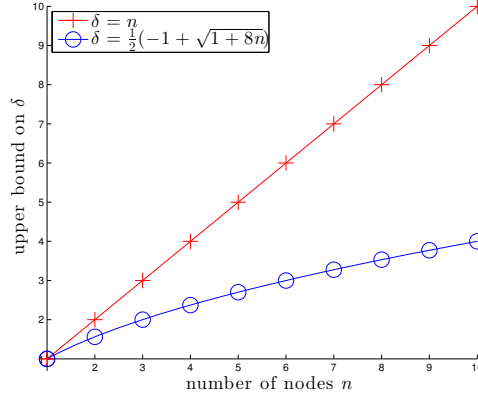


Figure 2.3: Presentation of the different upper bounds on $\delta(n)$.

2.3.4 A sufficient condition for global optimality

It was mentioned in the introduction that the problem (P) is generally non-convex and then that we have to be satisfied with local maxima. Result [Boumal et al., 2016, Theorem 2] provides a sufficient condition for global optimality. This condition is based only on local information, which is rather surprising given the non-convexity of the problem.

This result appeals to the notion of *second order critical point*. This point $\mathbf{H}$ fulfills the requirements

$$\begin{aligned} \text{grad } Q_k(\mathbf{H}) &= \mathbf{0}, \\ \text{Hess } Q_k(\mathbf{H}) &\preceq \mathbf{0}. \end{aligned}$$

Note that the operators grad and Hess refer to the Riemannian gradient and the Riemannian Hessian, respectively. These notions are not formally defined but can be viewed as equivalent to the classical Euclidean ones but on a Riemannian manifold.

The result [Boumal et al., 2016, Theorem 2] is the following.

Result 7. *If $\frac{k(k+1)}{2} > n$, then for almost all $\mathbf{B}$, any second order critical point is globally optimal.*

Proof. It is an application of [Boumal et al., 2016, Theorem 2]. Note that the assumptions of this theorem are fulfilled, indeed, in [Boumal et al., 2016, Section 4], the authors present the problem (P) as a typical example. $\square$

2.4 Numerical results

To have a better idea of what kind of values for $\delta(\mathcal{G})$ are actually observed in practical cases, this section illustrates what is the typical behavior of an optimal solution of problem (P).

We study Zachary’s karate club graph, ($n = 34$, $m = 78$) [Zachary, 1977]. This graph is depicted on figure 2.4. Figure 2.5a presents the solution obtained numerically with the embedding dimension $k = 3$. Each dot corresponds to a node, i.e. a line of $\mathbf{H}$. One can remark that the number of dots is less than n , which is due to the fact that some points coincide because they are very close community-wise.

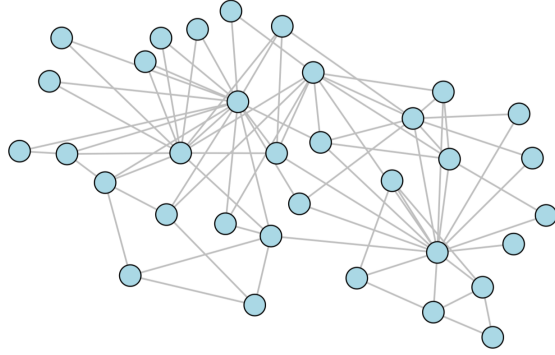


Figure 2.4: Representation of the Zachary’s karate club graph reprinted from [Caimo and Mira, 2015, Figure 10].

In addition, a black circle centered at the origin that passes through all the points has been added. The fact that such a circle exists indicates that $\delta(\mathcal{G}) < 3$. Figure 2.5b presents the solution obtained by setting the embedding dimension to $k = 2$. One can observe that this solution is similar to the one obtained on the case $k = 3$.

An important fact that this example reveals is that in practice, $\delta(\mathcal{G}) \ll n$. This phenomenon has been already observed and the minimal embedding dimension is generally so small that [Burer et al., 2002] propose a heuristic algorithm to solve a similar optimization program by setting $k = 2$. None of the graphs considered during research on this thesis had a minimal embedding dimension greater than 3.

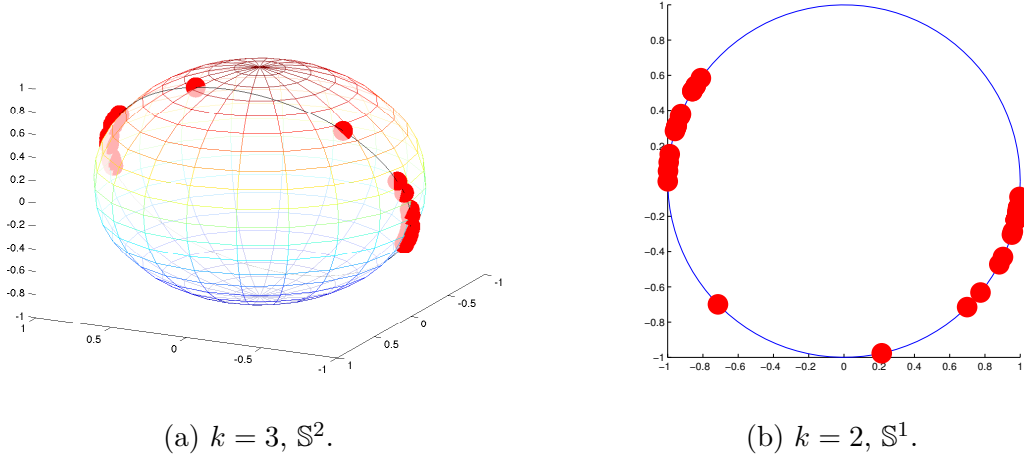


Figure 2.5: Representations of solutions of (P) obtained numerically on the Zachary's karate club graph for different values of the embedding dimension k .

A natural way to make a deeper numerical analysis of the minimal embedding dimension is to compute Q_k^* for increasing values of k . The minimal embedding dimension $\delta(\mathcal{G})$ is the smallest k from which the objective value stops to increase.

This method does not indicate if the solutions obtained when $k > \delta(\mathcal{G})$ are degenerated or not. To answer this question, one can propose to compute the rank of a numerical solution with a large k , i.e. greater than $\delta(\mathcal{G})$. If the rank is greater than the minimal embedding dimension, the solution is degenerated. The problem of this method is that numerical solutions are not exact and thus the rank will probably be too high. Indeed, a noisy low rank matrix is generally a full rank matrix. Some experiments show that the numerical solutions are always full rank. Alternatively, one can look at the *condition number* of the solution $\mathbf{H}$, $\kappa(\mathbf{H})$ instead of its rank. Qualitatively, the higher $\kappa(\mathbf{H})$, the more $\mathbf{H}$ is singular and the more $\mathbf{H}$ is a degenerated solution.

An alternative way to recognize the degenerated solutions is to check if the solution spans the whole space. More precisely, the idea is to look if the rows of $\mathbf{H} \in \mathbb{R}^{n \times k}$ considered as points on $\mathbb{S}^{k-1}$ can be intersected by a $(k-1)$ -dimensional hyperplane $\mathcal{L}$ that passes through the origin. Indeed, if it is the case, the points are in fact on a possibly rotated $(k-2)$ -dimensional sphere and the solution does not spans the whole space, i.e. $\text{rank}(\mathbf{H}) < k$, similarly to the one observed on figure 2.5a. One can construct the hyperplane that passes through the origin and as close as possible to all the rows of $\mathbf{H}$ in the sense of the least square error and can check the norm of the resulting error. The infinity norm $\|\mathbf{e}\|_\infty := \max_{1 \leq i \leq n} |\mathbf{e}_i|$ is preferable in order to be sure that all nodes are close to the hyperplane. Note that this procedure makes sense only when $k > 1$.

Now, we compute those diagnostic values on Zachary's karate club graph to test them and understand how they work. Figure 2.6 presents the modularity when k increases. We observe that modularity increases when k comes from 1 to 2 and then stays constant. This indicator shows that $\delta = 2$. Considering the degeneration of the numerical solutions, figure 2.7a indicates $\kappa(\mathbf{H}^*)$ in a logarithmic y -scale. This

number increases very fast with k . That means that the solutions are of very low rank and then that they are *not too* degenerated. The second indicator, based on the interpolation error, is depicted on figure 2.7b. It confirms the conclusions deduced from the condition number. When $k = 2$, there is a non negligible error when the hyperplane, i.e. the line, passes through the points but as soon as $k > 2$, there is a very small error, indicating that the solution is not full rank, i.e. $\text{rank}(\mathbf{H}) < k$.

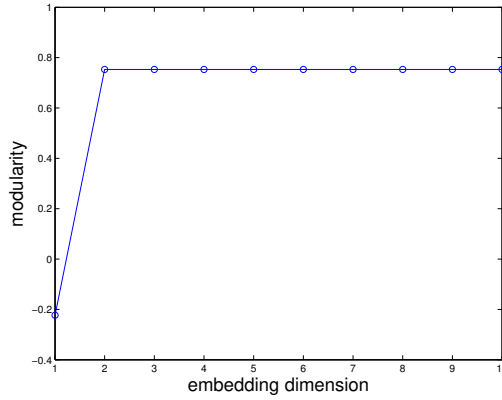
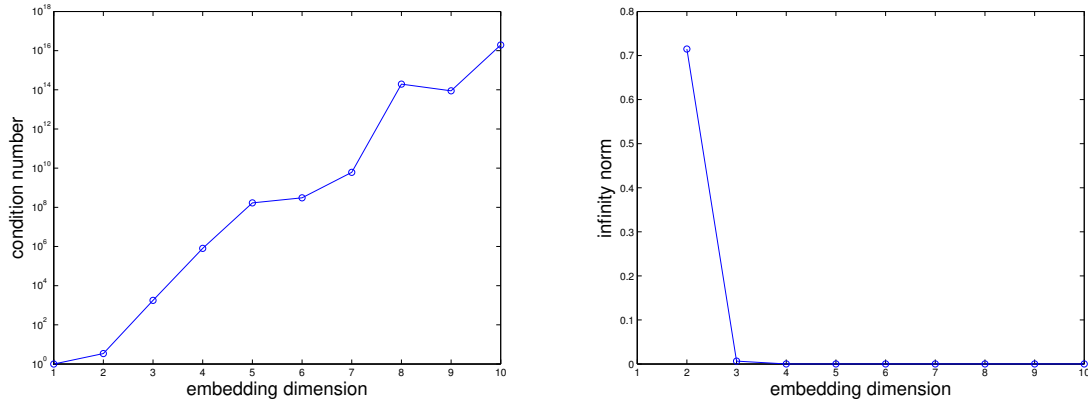


Figure 2.6: Evolution of modularity Q_k^* as a function of the embedding dimension k on Zachary's karate club graph.



(a) Evolution of the condition number $\kappa(\mathbf{H})$. (b) Evolution of the infinity norm of the interpolation error. The y -scale is logarithmic.

Figure 2.7: Evolution of the diagnostic values with the embedding dimension k in order to learn about the minimal embedding dimension. The graph tested is Zachary's karate club graph.

A similar analysis is done on a greater graph. We choose Polblog's graph ($n = 1224$, $m = 3937$), which will be presented in chapter 5.

Figure 2.8 indicates that again in this case, $\delta(\mathcal{G}) = 2$.

Figure 2.9a shows that, in this case, the solutions obtained with $k > \delta(\mathcal{G})$ are generally degenerated. Indeed, the condition number is less than 10 when it was greater than 10^{10} in the non degenerated case of Zachary's Karate Club's graph. That means that $\text{rank}(\mathbf{H}^*) > \delta(\mathcal{G})$ so degeneracy occurs, at least for numerical solutions. Figure 2.9b shows a non-negligible error (greater than 0.7) that indicates that the numerical solution naturally spans the whole space. In other words $\text{rank}(\mathbf{H}^*) = k$.

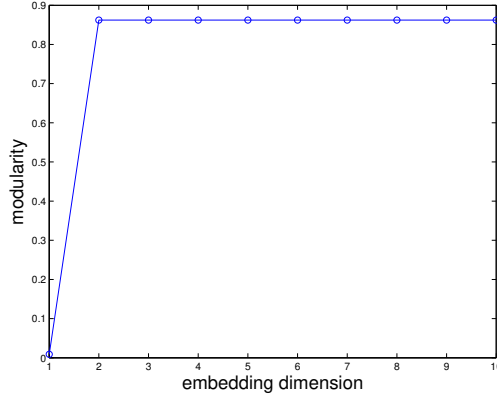
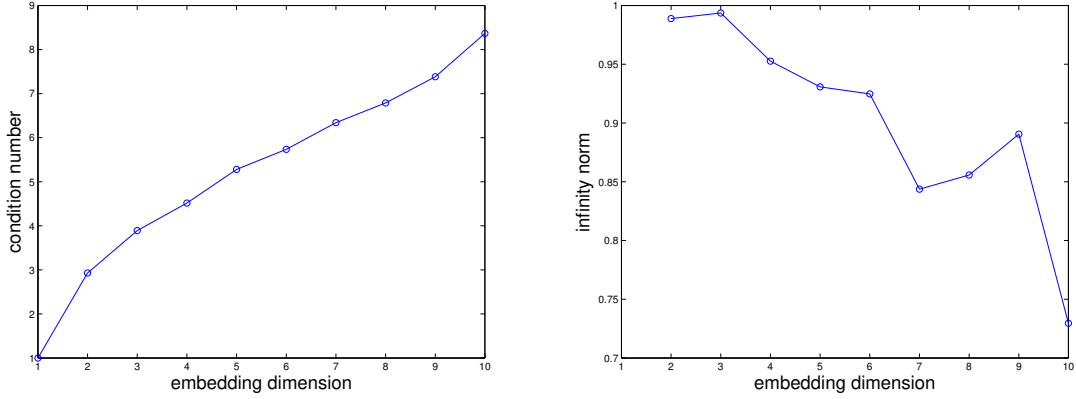


Figure 2.8: Evolution of modularity Q_k^* as a function of the embedding dimension k on Polblog's graph.



(a) Evolution of the condition number $\kappa(\mathbf{H})$. (b) Evolution of the infinity norm of the interpolation error. The y-scale is not logarithmic.

Figure 2.9: Evolution of the diagnostic values with the embedding dimension k in order to learn about the minimal embedding dimension. The graph tested is Polblog's graph.

The fact that the numerical solutions sometimes have a natural tendency to fulfill the whole space indicates that it makes sense to set the embedding dimension k sufficiently large, i.e. $k > \delta(\mathcal{G})$. Indeed, if the nodes fulfill the whole $(k - 1)$ -dimensional sphere, the communities (nodes close on the sphere) will probably be more distinguishable from

each other than if all the nodes are flattened on a circle. In other words, communities are probably more separated from each other when k is larger than the minimal embedding dimension, making the identification of communities easier. If the objective is to find a fuzzy solution, i.e. a solution of problem (P), it makes sense to set $k := 2$ or 3 or more generally $k := \delta(\mathcal{G})$. Otherwise, if one is looking for a solution of problem (Pd) obtained by solution of (P) followed by a clustering, a larger value of k looks reasonable.

2.5 Summary

To conclude this chapter, the main results are summarised. The minimal embedding dimension is defined. Its well-definedness is proved. It is shown that there exists graphs with arbitrarily many nodes that have a minimal embedding dimension of 1. A proof of the fact that the minimal embedding dimension is bounded from above by the number of nodes is proposed. A better bound from [Pataki, 1998] or [Barvinok, 1995] is then mentioned.

Another result is that, under some weak assumptions, the rank of an optimal solution is greater than the embedding dimension.

Finally, some numerical experiments show that the minimal embedding dimension is generally as small as 2 or 3. If one wants to solve the problem (P), one can set the embedding dimension k very small. Otherwise, different experiments highlight the fact that when the embedding dimension is greater than the minimal one, the numerical solutions sometimes tend to span the whole space.

The main contributions of this chapter are the lemmas and results for which the proof does not refer to a reference. The definition of the minimal embedding dimension and the proof of its well-definedness are part of it.

Chapter 3

Static algorithms

This chapter focuses on the presentation of four methods to solve the relaxed problem (P). All the algorithms presented here are iterative in order to have the following dynamic property: if the solution of the problem is known but that later, the graph changes a bit, one can recover a new good solution after only a few iterations. These algorithms will be used to solve the problem of community detection in time-varying networks. When a solution of the relaxed problem (P) is found, a solution of the discrete problem (Pd) is recovered from it using a clustering algorithm. This idea is presented in the static case at the end of this chapter.

3.1 Node by node algorithm

This algorithm is a *random block-coordinate ascent method* where each block is a row of $\mathbf{H}$. It proceeds as follows. Assume that there is a feasible $\mathbf{H} \in \mathbb{R}^{n \times k}$, it chooses a node $N \in V$ randomly and updates $\mathbf{H}_{N,:}$ such that $Q_k(\mathbf{H})$ is maximal and such that this new $\mathbf{H}$ is feasible. This corresponds to find the best position of the node on the sphere. This is why the algorithm is called node by node (NBN). It is natural to choose the rows of $\mathbf{H}$ as blocks because the constraints of problem (P) are naturally expressed on them.

3.1.1 Preliminaries

Let's compute the optimal value of $\mathbf{H}_{N,:}$ that is needed for the update.

The notation $z_{ij} := \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} \in \mathbb{R}$ is used. Note that $z_{ij} = z_{ji}$. The objective

function is

$$\begin{aligned}
Q_k(\mathbf{H}) &= \sum_{i=1}^n \sum_{j=1}^n z_{ij} \\
&= \sum_{i=1, i \neq N}^n \sum_{j=1}^n z_{ij} + \sum_{j=1}^n z_{Nj} \\
&= \sum_{i=1, i \neq N}^n \sum_{j=1, j \neq N}^n z_{ij} + \sum_{i=1, i \neq N}^n z_{iN} + \sum_{j=1, j \neq N}^n z_{Nj} + z_{NN}.
\end{aligned}$$

By the symmetry of z_{ij} and using $z_{NN} = \mathbf{B}_{NN} \sum_{l=1}^k \mathbf{H}_{Nl}^2 = \mathbf{B}_{NN}$ where the constraint of (P) was used, it yields

$$Q_k(\mathbf{H}) = \sum_{i=1, i \neq N}^n \sum_{j=1, j \neq N}^n z_{ij} + \mathbf{B}_{NN} + 2 \sum_{i=1, i \neq N}^n \mathbf{B}_{iN} \sum_{l=1}^k \mathbf{H}_{Nl} \mathbf{H}_{il}.$$

One can remark that the two first terms are independent of $\mathbf{H}_{N,:}$ and thus, maximizing $Q_k(\mathbf{H})$ with respect to $\mathbf{H}_{N,:}$ is equivalent to maximizing only the last term with respect to these same variables. In addition, note that this last term is linear in our variables $\mathbf{H}_{N,:}$, indeed, it can be written (the constant multiplicative factor is omitted because it does not influence the maximum)

$$\sum_{l=1}^k \sum_{i=1, i \neq N}^n \underbrace{\mathbf{B}_{iN} \mathbf{H}_{il}}_{(\mathbf{c}^N)_l} \mathbf{H}_{Nl}$$

As a consequence, the one node problem that has to be solved can be written as

$$\mathbf{H}_{N,:} = \arg \max_{\mathbf{x} \in \mathbb{R}^{1 \times k}} \mathbf{c}^N \mathbf{x}^T \text{ such that } \|\mathbf{x}\| = 1 \quad (3.1)$$

with

$$(\mathbf{c}^N)_l = \sum_{i=1, i \neq N}^n \mathbf{B}_{iN} \mathbf{H}_{il} \in \mathbb{R}^{1 \times k}. \quad (3.2)$$

Note that $\mathbf{c}^N$ is a row vector.

From a geometric interpretation, we can see the objective as the orthogonal projection of $\mathbf{x}$ on the line spanned by $\mathbf{c}^N$. To maximize this quantity, we choose $\mathbf{x}$ parallel to $\mathbf{c}^N$ and oriented toward the same direction. Finally, taking the constraint into account yields the following analytic solution

$$\mathbf{H}_{N,:} = \frac{\mathbf{c}^N}{\|\mathbf{c}^N\|}$$

where it was assumed that $\mathbf{c}^N \neq \mathbf{0}$. Note that the existence of analytic formula for the global maximum of the multivariate non linear program (3.1) is remarkable.

3.1.2 Remark about the assumption $\mathbf{c}^N \neq \mathbf{0}$

Due to the assumption $\mathbf{c}^N \neq \mathbf{0}$, the nodes N such that $\mathbf{c}^N = \mathbf{0}$ cannot be updated. Note that this is not restrictive in the sense that if $\mathbf{c}^N = \mathbf{0}$, the objective function of the one node problem (3.1) is constant.

The algorithm goes through the list of nodes and updates the corresponding value in $\mathbf{H}$. This is done by a *loop*. It has to ensure that for the values of N such that this assumption is violated, this update is skipped.

A naive way to do is to verify if $\mathbf{c}^N \neq \mathbf{0}$ before doing the normalization. This can be done using an *if ... do* statement. Unfortunately, it is well known that doing this in a *loop* significantly slows down the algorithm.

Another option is to restrict the set of values of N in the *loop* to the ones for which $\mathbf{c}^N \neq \mathbf{0}$. In order to do this, we have to know *a priori* for which values of N this appears. One can define $V := \{1, \dots, n\}$ and $\bar{N} := V \setminus \{N\}$. The situation $\mathbf{c}^N = \mathbf{0}$ appears in two different cases.

The first one is when each column of $\mathbf{H}_{\bar{N}, \cdot}$ is orthogonal to $\mathbf{B}_{N\bar{N}}$ (see relation (3.2)) which appears with probability 0 if $\mathbf{H}$ is chosen randomly. After some iterations $\mathbf{H}$ is not random and thus this argumentation is not very robust but in practice, this pathological case has never occurred. This is why this case is ignored.

The second case for which $\mathbf{c}^N = \mathbf{0}$ is when $\mathbf{B}_{N\bar{N}} = \mathbf{0}$. This case occurs independently of the value that $\mathbf{H}$ takes. It is only a property of the data thus we limit ourselves to only removing the values of N for which $\mathbf{B}_{N\bar{N}} = \mathbf{0}$. One can note that this second case is a particular setting of the first one.

To implement this idea, a naive way to do is to compute $\mathbf{B}$ and to check for which values of N , $\mathbf{B}_{N\bar{N}} \neq \mathbf{0}$, but computing $\mathbf{B}$ is expensive (it can be done in $O(n^2)$). Thankfully, the following result allows to accomplish this task efficiently.

Result 8. *Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ and let $N \in V$ be such that $\mathbf{A}_{NN} = 0$ and $\mathbf{A}_{Nj} \geq 0$ for all $j \in V$. Then $\mathbf{d}_N \neq 0$ if and only if $\mathbf{B}_{N\bar{N}} \neq \mathbf{0}$.*

This result indicates that the problematic nodes are exactly the isolated ones.

Before proving it, note that it allows to check if $\mathbf{B}_{N\bar{N}} \neq \mathbf{0}$ in $O(1)$ by checking if $\mathbf{d}_N \neq 0$. Doing this, we can find the set of valid indices in $O(n)$ without computing $\mathbf{B}$ explicitly instead of $O(n^2)$ with a direct approach.

Note that if $\mathbf{A}_{NN} \neq 0$, this result does not state that $\mathbf{B}_{N\bar{N}} = \mathbf{0}$ when $\mathbf{d}_N \neq 0$. It does not give any information about this case.

Proof. Let $N \in V$, we prove the equivalent proposition: $\mathbf{d}_N = 0$ if and only if $\mathbf{B}_{N\bar{N}} = \mathbf{0}$. We prove the two directions separately.

- $\mathbf{d}_N = 0 \Rightarrow \mathbf{B}_{N\bar{N}} = \mathbf{0}$: Assume $\mathbf{d}_N = 0$, thanks to the assumption $\mathbf{A}_{ij} \geq 0$, it

holds that $\mathbf{A}_{N,:} = \mathbf{0}$. We compute

$$\mathbf{B}_{N,:} = \frac{1}{2m} \left(\mathbf{A}_{N,:} - \frac{\mathbf{d}_N \mathbf{d}}{2m} \right) = \mathbf{0}$$

where we have used $\mathbf{A}_{N,:} = \mathbf{0}$ and $\mathbf{d}_N = 0$. Noting that $\bar{N} \subset V$, we have $\mathbf{B}_{N\bar{N}} = \mathbf{0}$ and we are done.

- $\mathbf{B}_{N\bar{N}} = \mathbf{0} \Rightarrow \mathbf{d}_N = 0$: Assume $\mathbf{B}_{N\bar{N}} = \mathbf{0}$, we have

$$\begin{aligned} \mathbf{0} &= \mathbf{B}_{N\bar{N}} = \frac{1}{2m} \left(\mathbf{A}_{N\bar{N}} - \frac{\mathbf{d}_N \mathbf{d}_{\bar{N}}}{2m} \right) \\ \Leftrightarrow \mathbf{A}_{N\bar{N}} &= \frac{\mathbf{d}_N \mathbf{d}_{\bar{N}}}{2m} \\ \Rightarrow \sum_{j \in \bar{N}} \mathbf{A}_{Nj} &= \frac{\mathbf{d}_N}{2m} \sum_{j \in \bar{N}} \mathbf{d}_j \\ \Leftrightarrow \underbrace{\sum_{j \in V} \mathbf{A}_{Nj}}_{\mathbf{d}_N} - \underbrace{\mathbf{A}_{NN}}_0 &= \frac{\mathbf{d}_N}{2m} \left(\underbrace{\sum_{j \in V} \mathbf{d}_j}_{2m} - \mathbf{d}_N \right) \\ \Leftrightarrow \mathbf{d}_N &= \mathbf{d}_N - \frac{\mathbf{d}_N^2}{2m} \\ \Leftrightarrow 0 &= \frac{\mathbf{d}_N^2}{2m} \\ \Leftrightarrow \mathbf{d}_N &= 0. \end{aligned}$$

□

3.1.3 Comments on the implementation

From an implementation point of view, it is crucial to choose an efficient way to compute $\mathbf{c}^N$. Indeed, it is the most expensive part of the computations. The notation $\bar{m}$ refers to the number of non-zero elements in $\mathbf{A}$. The definition (1.2) of the modularity matrix $\mathbf{B}$ highlights the fact that this matrix is not sparse, even when $\mathbf{A}$ is sparse. Therefore, to compute efficiently a matrix product involving $\mathbf{B}$ we do not use $\mathbf{B}$ explicitly. For the same reason, $\mathbf{c}^N$ is computed vectorially. Firstly, observe that one can write $\mathbf{c}^N = \mathbf{B}_{N,:} \mathbf{H} - \mathbf{B}_{NN} \mathbf{H}_{N,:}$. Using the definition of $\mathbf{B}$ (1.2), the following formula follows

$$\mathbf{c}^N = \frac{1}{2m} \mathbf{A}_{N,:} \mathbf{H} - \frac{1}{4m^2} \mathbf{d}_N (\mathbf{d}^T \mathbf{H}) - \frac{1}{2m} \left(\mathbf{A}_{NN} - \frac{\mathbf{d}_N^2}{2m} \right) \mathbf{H}_{N,:}. \quad (3.3)$$

The terms can be computed, from left to right, in $O(\bar{m}k)$, $O(nk)$ and $O(k)$ and thus this formula is $O(k\bar{m})$. Note that the fact that $O(m) \geq O(n+1) = O(n)$ if the graph is assumed to be connected has been used.

One can proceed similarly to evaluate $Q_k(\mathbf{H})$. Firstly, recall that $Q_k(\mathbf{H}) = \text{trace}(\mathbf{H}^T \mathbf{B} \mathbf{H})$. Using the definition of $\mathbf{B}$ (1.2) yields the following formula

$$Q_k(\mathbf{H}) = \text{trace} \left(\frac{1}{2m} \mathbf{H}^T \mathbf{A} \mathbf{H} - \frac{1}{4m^2} (\mathbf{H}^T \mathbf{d})(\mathbf{d}^T \mathbf{H}) \right). \quad (3.4)$$

The product $\mathbf{H}^T \mathbf{A}$ is $O(k\bar{m})$, the product of the result with $\mathbf{H}$ is $O(nk)$. Concerning the second term, each parenthesis is $O(nk)$, the product between them is $O(k)$. Finally, compute the trace is $O(k)$. We conclude that this formula is $O(k\bar{m})$.

We verify with numerical experiments that these formulas are faster than a computation that uses $\mathbf{B}$ explicitly. We use MATLAB on the Power Europe's graph ($n = 2783$, $\bar{m} = 7524$) that will be introduced in chapter 5 and we compute $\mathbf{c}^1$ with $k = 3$. The naive formula takes $1.7 \cdot 10^{-2}$ seconds when formula (3.3) proceeds in $3.4 \cdot 10^{-4}$.

Finally, a stopping criterion is needed, we use the Frobenius norm of the difference between two successive iterates. Recall that, the Frobenius norm of a matrix $\mathbf{H} \in \mathbb{R}^{n \times k}$ is defined as

$$\|\mathbf{H}\|_F := \sqrt{\sum_{i=1}^n \sum_{l=1}^k \mathbf{H}_{il}^2}.$$

In order to improve the computation of this quantity, one can observe the following. Let $\mathbf{H}$ be an iterate and $\tilde{\mathbf{H}}$ be the next one, then

$$\begin{aligned} \|\mathbf{H} - \tilde{\mathbf{H}}\|_F^2 &= \sum_{i=1}^n \sum_{l=1}^k (\mathbf{H}_{il} - \tilde{\mathbf{H}}_{il})^2 \\ &= \sum_{i=1}^n \sum_{l=1}^k (\mathbf{H}_{il}^2 + \tilde{\mathbf{H}}_{il}^2 - 2\mathbf{H}_{il}\tilde{\mathbf{H}}_{il}) \\ &= \sum_{i=1}^n \left(\underbrace{\sum_{l=1}^k \mathbf{H}_{il}^2}_1 + \underbrace{\sum_{l=1}^k \tilde{\mathbf{H}}_{il}^2}_1 - 2 \sum_{l=1}^k \mathbf{H}_{il}\tilde{\mathbf{H}}_{il} \right) \\ &= 2n - 2 \sum_{i=1}^n \sum_{l=1}^k \mathbf{H}_{il}\tilde{\mathbf{H}}_{il} \\ &= 2n - 2\text{Tr}(\mathbf{H}^T \tilde{\mathbf{H}}). \end{aligned}$$

Note that to compute $\text{Tr}(\mathbf{H}^T \tilde{\mathbf{H}})$, it is not necessary to compute the complete product because only the diagonal elements are needed. Using those observations, for a tolerance $\epsilon > 0$ on this stopping criterion, one can write

$$\begin{aligned} \|\mathbf{H} - \tilde{\mathbf{H}}\|_F &> \epsilon \\ \Leftrightarrow \|\mathbf{H} - \tilde{\mathbf{H}}\|_F^2 &> \epsilon^2 \\ \Leftrightarrow 2n - 2\text{Tr}(\mathbf{H}^T \tilde{\mathbf{H}}) &> \epsilon^2 \\ \Leftrightarrow \text{Tr}(\mathbf{H}^T \tilde{\mathbf{H}}) &< n - \frac{\epsilon^2}{2}. \end{aligned}$$

Even if this formulation does not change the complexity of the computations, it reduces the number of operations.

Let's present the node by node algorithm.

1. Provide an initial feasible solution $\mathbf{H}$ of problem (P);
2. Find the set I of non isolated nodes;
3. For a random $N \in I$:
 - (a) Compute $\mathbf{c}^N$ from $\mathbf{H}$ using formula (3.3);
 - (b) Set $\mathbf{H}_{N,:} := \mathbf{c}^N / \|\mathbf{c}^N\|$;
4. Repeat step 3 until a stopping criterion is reached.

A pseudocode version of the node by node algorithm is available in appendix B.1.

3.1.4 Comments about the algorithm

If one wants a deterministic algorithm, step 3 can be replaced by a deterministic order on the elements of I .

In the light of the previous considerations, if the stopping criterion is the Frobenius norm of two successive iterates $\|\mathbf{H} - \mathbf{H}_{prev}\|_F > \epsilon$, one can write $Tr(\mathbf{H}^T \tilde{\mathbf{H}}) < n - \frac{\epsilon^2}{2}$. Even if the time complexity is the same, doing this requires fewer operations and is then faster. Recall that one can compute only the diagonal entries of the matrix product.

Note that it is also possible to replace this stopping criterion by any other matrix norm or by a criterion on the objective value: $|Q_k(\mathbf{H}) - Q_k(\mathbf{H}_{prev})| > \epsilon$. If one chooses the last, Q_k is computed using formula (3.4).

3.1.5 Complexities

Computations of complexities are detailed in appendix B.1.

It is shown that the time complexity of the node by node algorithm is $O(Kk\bar{m})$ where K and $\bar{m}$ are respectively the number of times that a node is updated and the number of non zero entries in matrix $\mathbf{A}$.

Otherwise, the space complexity is $\Theta(\max\{kn, \bar{m}\})$.

3.1.6 Behavior analysis

In this section, the typical behavior of the NBN algorithm on real-world examples is studied. More precisely, it focuses on the evolution of the modularity during the iterations of the NBN algorithm. An iteration refers to the procedure where each

node is updated exactly one time. Note that the time used to compute one iteration is not presented here and thus, this section is not relevant to study the efficiency of the algorithm. Such an analysis is the topic of section 5.1.

We choose the Power Europe's graph⁴ ($n = 2783$, $m = 3937$) and set $k = 3$. The initial iterate $\mathbf{H}^0$ is produced randomly. The results are presented in figure 3.1. One can observe that the increase in modularity in the first iterations is very important.

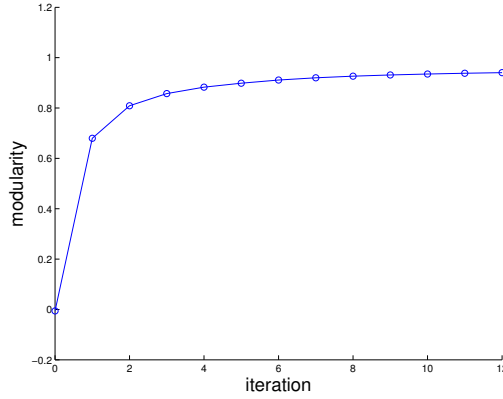


Figure 3.1: Evolution of the modularity during the iterations of the NBN algorithm on the Power Europe's graph. The embedding dimension is $k = 3$.

3.1.7 Preconditioning

In this section, a variant of the NBN algorithm is presented. It corresponds to precondition the data of the problem before calling the NBN algorithm. The idea is to modify the diagonal of matrix $\mathbf{B}$. This is made possible by the following result that is mentioned in [Fanuel, 2017].

Result 9. *Let $\mathbf{H} \in \mathbb{R}^{n \times k}$ be a feasible solution of (P) and let $\mathbf{D} \in \mathbb{R}^{n \times n}$ a diagonal matrix. The notation $Q_k^{\mathbf{B}}(\mathbf{H}) := \text{Tr}(\mathbf{H}^T \mathbf{B} \mathbf{H})$ is used to emphasize the dependency in $\mathbf{B}$. Then $Q_k^{\mathbf{B}+\mathbf{D}}(\mathbf{H}) = Q_k^{\mathbf{B}}(\mathbf{H}) + \text{Tr}(\mathbf{D})$.*

Proof. It follows from the definition of Q_k that

$$\begin{aligned}
 Q_k^{\mathbf{B}+\mathbf{D}}(\mathbf{H}) &= \sum_{i=1}^n \sum_{j=1}^n (\mathbf{B}_{ij} + \mathbf{D}_{ij}) \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} \\
 &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} + \sum_{i=1}^n \mathbf{D}_{ii} \underbrace{\sum_{l=1}^k \mathbf{H}_{il}^2}_{=1} \\
 &= Q_k^{\mathbf{B}}(\mathbf{H}) + \text{Tr}(\mathbf{D}).
 \end{aligned}$$

□

⁴See chapter 5 for details about this graph.

This result states that replacing matrix $\mathbf{B}$ by a matrix $\tilde{\mathbf{B}} := \mathbf{B} + \mathbf{D}$ with $\mathbf{D}$ a diagonal matrix does not change the solutions of the problem (P), indeed, $Tr(\mathbf{D})$ does not depend of $\mathbf{H}$.

We can use this property of the problem to precondition it in order to obtain a better behavior.

A natural choice is $\mathbf{D} := -\text{diag}(\mathbf{B})$ where $\text{diag}(\mathbf{B})$ is a diagonal matrix with the same diagonal as $\mathbf{B}$. Doing this, $\tilde{\mathbf{B}}$ is $\mathbf{B}$ but with its diagonal elements set to 0. It simplifies the formula $\mathbf{c}^N = \mathbf{B}_{N,:}\mathbf{H} - \mathbf{B}_{NN}\mathbf{H}_{N,:}$ to

$$\mathbf{c}^N = \tilde{\mathbf{B}}_{N,:}\mathbf{H} - \underbrace{\tilde{\mathbf{B}}_{NN}\mathbf{H}_{N,:}}_{=0} = \tilde{\mathbf{B}}_{N,:}\mathbf{H}.$$

A naive way to do is to construct a matrix $\tilde{\mathbf{B}}$ (it is $O(n^2)$ as for $\mathbf{B}$) that is not sparse and to compute directly $\mathbf{c}^N = \tilde{\mathbf{B}}_{N,:}\mathbf{H}$. We will prefer a non direct approach. First, we define

$$\tilde{\mathbf{A}}_{ij} := \begin{cases} \mathbf{A}_{ij} & \text{if } i \neq j \\ \frac{(\mathbf{d}_i)^2}{2m} & \text{if } i = j. \end{cases} \quad (3.5)$$

Observe that if $\mathbf{A}$ is sparse, $\tilde{\mathbf{A}}$ is still sparse (probably a bit less). Note that constructing $\tilde{\mathbf{A}}$ is not expensive (it can be done in $O(n)$). Thanks to this definition, one can write

$$\tilde{\mathbf{B}} = \frac{1}{2m} \left(\tilde{\mathbf{A}} - \frac{\mathbf{d}\mathbf{d}^T}{2m} \right) \quad (3.6)$$

and $\mathbf{c}^N$ can be computed efficiently by the following formula

$$\mathbf{c}^N = \frac{1}{2m} \tilde{\mathbf{A}}_{N,:}\mathbf{H} - \frac{1}{4m^2} \mathbf{d}_N(\mathbf{d}^T\mathbf{H})$$

that uses matrix $\tilde{\mathbf{A}}$.

The behavior of this algorithm with the embedding dimension $k = 3$ is presented in figure 3.2 for the Power Europe's graph. One can observe that each iteration is significantly less efficient than an iteration of the classic NBN algorithm.

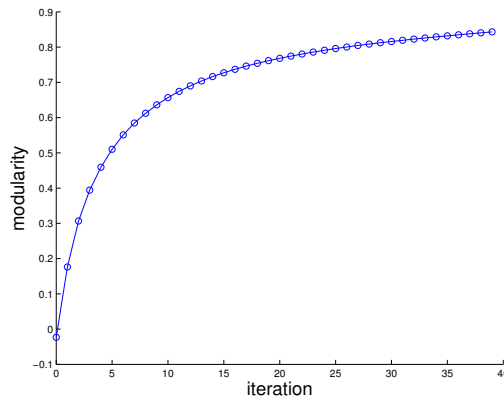


Figure 3.2: Evolution of the modularity during the iterations of the preconditioned NBN algorithm on the Power Europe's graph. The embedding dimension is $k = 3$.

This algorithm is written NBN-p to emphasis the preconditioning. Finally, recall that the choice $\mathbf{D} = -\text{diag}(\mathbf{B})$ is intuitive but arbitrary.

3.1.8 Improvements

A number of tricks can be used to try to improve the NBN algorithm. One can try to find a good *preconditioning* matrix $\mathbf{D}$ (see previous section). One can also study an *accelerated ascent* [Nesterov and Stich, 2017b] or a *parallelization* of the block-updates [Tappenden et al., 2015]. Some researches also indicate that assigning a non-uniform probability to all blocks can improve the performance of the algorithm [Richtárik and Takáč, 2013]. All these tricks can even be combined together.

3.2 Total update algorithm

The second algorithm is based on the idea of the node by node algorithm but here, instead of updating nodes one after another, the total update algorithm updates all nodes at the same time. Doing this, the equivalent of $\mathbf{c}^N \in \mathbb{R}^{1 \times k}$ in the node by node algorithm is a matrix $\mathbf{C} \in \mathbb{R}^{n \times k}$ in the total update algorithm. For this matrix the following holds $\mathbf{C}_{Nl} = (\mathbf{c}^N)_l = \mathbf{B}_{N,:} \mathbf{H}_{:,l} - \mathbf{B}_{NN} \mathbf{H}_{Nl}$. In matrix notation, it is

$$\mathbf{C} = \mathbf{B}\mathbf{H} - \text{diag}(\mathbf{B})\mathbf{H} = (\mathbf{B} - \text{diag}(\mathbf{B}))\mathbf{H} = \tilde{\mathbf{B}}\mathbf{H} \quad (3.7)$$

where $\text{diag}(\mathbf{B})$ is a diagonal matrix with the same diagonal as $\mathbf{B}$ and where $\tilde{\mathbf{B}} := \mathbf{B} - \text{diag}(\mathbf{B})$.

One can observe some similarities with the preconditioned NBN algorithm presented in section 3.1.7. Note that preconditioning the problem before using the total update algorithm is useless. Indeed, even if we change the diagonal of $\mathbf{B}$, the formula (3.7) naturally set it to 0.

The matrix $\tilde{\mathbf{A}}$ is defined as in (3.5) and then (3.6) still holds. Thanks to this definition, $\mathbf{C}$ can be computed by the following formula

$$\mathbf{C} = \frac{1}{2m} \tilde{\mathbf{A}}\mathbf{H} - \frac{1}{4m^2} \mathbf{d}(\mathbf{d}^T \mathbf{H}). \quad (3.8)$$

In this section, $\bar{m}$ is the number of non-zero elements in $\tilde{\mathbf{A}}$. Using the sparsity of $\tilde{\mathbf{A}}$, the first term is computed in $O(k\bar{m})$ and the second one in $O(kn + kn) = O(kn)$ and thus this formula is $O(k \max\{\bar{m}, n\}) = O(k\bar{m})$ instead of $O(kn^2)$ with the explicit approach.

When $\mathbf{C}$ is computed we normalize each of its rows and thus each row has to be non zero. This condition is $\mathbf{C}_{N,:} \neq \mathbf{0}$ but by construction of $\mathbf{C}$, it holds that $\mathbf{C}_{N,:} = \mathbf{c}^N$ and then the comments in section 3.1.2 apply and only isolated nodes are removed.

The total update algorithm is then:

1. Provide an initial feasible solution $\mathbf{H}$ of problem (P);
2. Find the set I of non isolated nodes;
3. Construct the matrix $\tilde{\mathbf{A}}$ using formula (3.5);
4. Compute $\mathbf{C}$ from $\mathbf{H}$ using formula (3.8);
5. For any $N \in I$, set $\mathbf{H}_{N,:} := \mathbf{C}_{N,:} / \|\mathbf{C}_{N,:}\|$;
6. Go back to step 4 until a stopping criterion is reached.

A pseudocode version of the total update algorithm is available in appendix B.2.

3.2.1 Comments about the algorithm

One can observe that this algorithm is deterministic.

About the stopping criterion, if one chooses to use $\|\mathbf{H} - \mathbf{H}_{prev}\|_F > \epsilon$, it can be written $Tr(\mathbf{H}^T \mathbf{H}) < n - \frac{\epsilon^2}{2}$ as explained on the NBN algorithm. If one prefers, one can use any matrix norm or a stopping criterion on the objective value: $|Q_k(\mathbf{H}) - Q_k(\mathbf{H}_{prev})| > \epsilon$.

3.2.2 Complexities

Details about time and space complexities are developed in appendix B.2.

The time complexity of the total update algorithm is $O(Kk\bar{m})$ where K and $\bar{m}$ are respectively the number of repetitions of steps 4 and 5, and the number of non zero elements in $\tilde{\mathbf{A}}$. This complexity is better than the one of the NBN algorithm because here, each node has been updated K times, while for NBN, each node has been updated only K/n times because for the NBN algorithm, K is the total number of updates of the individual nodes.

The space complexity is $\Theta(\max\{kn, \bar{m}\})$.

3.2.3 Matrix expansion

One can see that, in fact, what is done is something close to the simultaneous iteration method applied on $\tilde{\mathbf{B}}$. The only difference is that here, the normalization is done on each row instead than on each column. It is well known that under some assumptions the method of simultaneous iteration converges towards a matrix $\mathbf{H}^*$ such that $\text{Col}\{\mathbf{H}^*\} = \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_k)$ where the $\mathbf{v}_i$ are the eigenvectors of $\tilde{\mathbf{B}}$ with the greatest associated eigenvalues (in absolute value) [Van Dooren and Absil, 2015, Proposition 4.5]. In the light of this, it is natural to look for a link between the iterates of the total update algorithm and some eigenvectors.

To emphasize the current iteration, each matrix is superscripted by it: we write $\mathbf{C}^i := \tilde{\mathbf{B}}\mathbf{H}^i$.

We define $\mathbf{N}^i := \text{diag}(\|\mathbf{C}_{1,:}^i\|^{-1}, \dots, \|\mathbf{C}_{n,:}^i\|^{-1})$ the normalization matrix.⁵ It is a diagonal matrix such that each row of $\mathbf{H}^{i+1} = \mathbf{N}^i \mathbf{C}^i$ is of unit norm.

A complete iteration of the Total Update algorithm is $\mathbf{H}^{i+1} = \mathbf{N}^i \tilde{\mathbf{B}} \mathbf{H}^i$ and then

$$\mathbf{H}^{i+1} = \mathbf{N}^i \tilde{\mathbf{B}} \mathbf{N}^{i-1} \tilde{\mathbf{B}} \dots \mathbf{N}^1 \tilde{\mathbf{B}} \mathbf{N}^0 \tilde{\mathbf{B}} \mathbf{H}^0.$$

If matrices $\mathbf{N}^j$ and $\tilde{\mathbf{B}}$ would commute, we would find $\mathbf{H}^{i+1} = \mathbf{N}^i \mathbf{N}^{i-1} \dots \mathbf{N}^0 \tilde{\mathbf{B}}^i \mathbf{H}^0$ and $\tilde{\mathbf{B}}^i$, the i -th power of $\tilde{\mathbf{B}}$ would make eigenvectors appear during convergence. Unfortunately, these matrices do not commute in general and then finding a link with eigenvectors or with spectral clustering (see [von Luxburg, 2007]) seems non trivial.

3.2.4 Behavior analysis

As for the NBN algorithm, the behavior of the algorithm during iterations on the Power Europe's graph is studied. The value of the embedding dimension k is set to 3 and the initial iterate $\mathbf{H}^0$ is the same as in the case of the NBN algorithm. Evolution of the modularity throughout iterations is presented in figure 3.3. We observe a pretty slow convergence at the beginning but also on the final iterations. Recall that this representation does not take into account the time needed to perform an iteration.

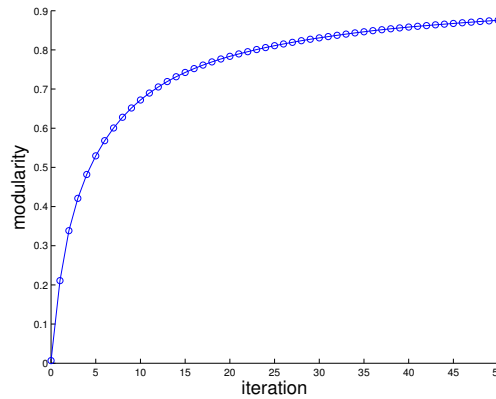


Figure 3.3: Evolution of the modularity during the iterations of the total update algorithm on the Power Europe's graph. The embedding dimension is $k = 3$.

The behavior is in fact very similar to the one of the preconditioned NBN algorithm, this is shown in figure 3.4.

⁵If $\|\mathbf{C}_{j,:}^i\| = 0$, we set $\mathbf{N}_{jj}^i = 1$.

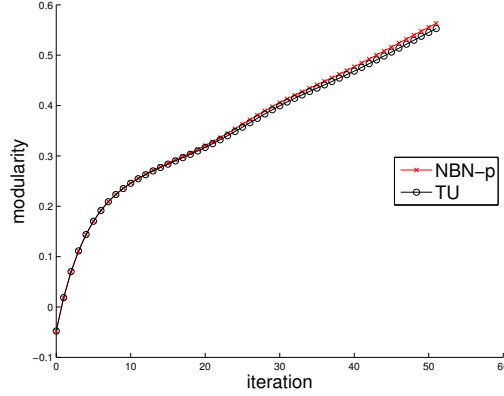


Figure 3.4: Comparison of the evolution of the modularity during the iterations of the TU algorithm and the preconditioned NBN algorithm. The computations are executed on the Power Europe’s graph with $k = 3$.

3.3 Riemannian trust-region algorithm

This algorithm is developed in [Absil et al., 2008] and uses tools of optimization on Riemannian manifolds. In [Boumal et al., 2016] the idea of using this algorithm to solve a problem more general than the one studied here is mentioned.

An advantage of optimization on manifolds is that the feasible domain appears as the *natural* domain to search solutions. With such an approach, the optimization methods for unconstrained problems can be used for constrained ones because the constraints are made invisible.

The idea of the Riemannian trust-region algorithm (RTR) is to generalize the trust-region methods in $\mathbb{R}^n$ to Riemannian manifolds. In the case of problem (P), the manifold is $(\mathbb{S}^{k-1})^n$ and it corresponds to the case of an oblique manifold. It is a Riemannian manifold when it is equipped with the canonical inner product $\langle \mathbf{H}_1, \mathbf{H}_2 \rangle_{(\mathbb{S}^{k-1})^n} := \text{Tr}(\mathbf{H}_1^T \mathbf{H}_2)$ as a Riemannian metric.

For the sake of simplicity, we only present the idea behind trust-region methods in $\mathbb{R}^n$. A precise description of the Riemannian case is fully developed in [Absil et al., 2008, Chapter 7, Algorithms 10 and 11].

Our implementation of this algorithm uses the MANOPT toolbox of MATLAB (see [Boumal et al., 2014]) that contains a `trustregions` solver. The code was initially written by Nicolas Boumal⁶ in 2012 and updated in 2015 by himself. This implementation has already been used by Elisabeth Schobbens in her thesis [Schobbens, 2014].

Note that the algorithm does not work for $k = 1$ since the admissible set in that case is $\mathbf{H}_{ij} \in \{-1, 1\}$, which is not a Riemannian manifold.

⁶Instructor in applied mathematics at Princeton University.

3.3.1 Trust-region methods in $\mathbb{R}^n$

The trust-region method in $\mathbb{R}^n$ are presented⁷. In this section, n is the dimension and does not refer to the number of nodes. The goal is to find the maximum of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. If $\mathbf{x}$ is the current iterate, the method constructs $m(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}$, the second order Taylor expansion of f at $\mathbf{x}$

$$m(\mathbf{h}) := f(\mathbf{x}) + \mathbf{h}^T \nabla f(\mathbf{x}) + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h}$$

where ∇f is the gradient of f and $\nabla^2 f$ is the Hessian matrix of f . This function can be seen as a quadratic model to approximate f close to $\mathbf{x}$.

Secondly, a trust-region radius $\Delta > 0$ is defined.

The idea is to find $\mathbf{h}$ solution of the so called trust-region subproblem

$$\max_{\mathbf{h} \in \mathbb{R}^n} m(\mathbf{h}) \text{ such that } \|\mathbf{h}\| \leq \Delta. \quad (3.9)$$

One can interpret this program as solving an approximation of the problem. The constraint enforces the fact that the approximation is valid only close to $\mathbf{x}$. Note that solve problem (3.9) is not trivial and can require an iterative algorithm. Such an algorithm is called an inner iteration.

Finally, when a $\mathbf{h}$ has been found, the quality of the result is assessed using the ratio

$$\rho = \frac{f(\mathbf{x}) - f(\mathbf{x} + \mathbf{h})}{m(\mathbf{0}) - m(\mathbf{h})}.$$

Depending on the value of ρ , the new candidate $\mathbf{x} + \mathbf{h}$ is accepted or rejected. More precisely, the following breaking rule is used:

- If ρ is very small, $m(\cdot)$ is a too bad approximation: the new candidate is rejected and the trust-region radius is reduced;
- If ρ is small but not too much, the new candidate is accepted but the step is reduced;
- If ρ is close to 1, the new candidate is accepted and the trust-region radius can be expanded;
- If $\rho \gg 1$, the model is inaccurate but the new candidate increases the cost function and then, it is accepted. One can hope that this will hold if the trust-region radius is increased and then, it is increased.

This procedure is repeated toward a stopping criterion is reached.

⁷This section follows [Absil et al., 2008, Section 7.2.1] devoted to the trust-region methods in $\mathbb{R}^n$.

3.3.2 Comments on the algorithm

In general, trust-region method works even if only a *sufficiently good* approximation of the Hessian is used. In the case of problem (P), we use the exact one because it is available and because it gives better convergence properties. This means that this is a second order method.

To solve the trust-region subproblem (3.9), a truncated conjugate gradient method is used. See [Absil et al., 2008, Algorithm 11] for details in the Riemannian case.

A quantitative version of the breaking rule can be found in [Absil et al., 2008, Algorithm 10].

An interesting comment about the cost of an iteration is indicated in [Boumal et al., 2016]:

“Essentially, each iteration of RTR requires evaluation of one cost and one gradient, a bounded number of Hessian-vector applications, and one projection from $\mathbb{R}^{n \times k}$ to $(\mathbb{S}^{k-1})^n$.”

3.3.3 Convergence

Some convergence results are presented. All of them come from [Absil et al., 2008, Section 7.3] that is devoted to the convergence analysis of the Riemannian trust-region method. In appendix A.3, the results are formally presented and the assumptions are verified.

Those results give theoretical guarantees that the RTR algorithm has some desirable properties: it converges to local maxima for any initial iterate and reach finally (under some assumptions) a quadratic convergence.

3.3.4 Behavior analysis

As previously, the behavior analysis is done on the Power Europe’s graph with $k = 3$. The evolution of the modularity throughout the iterations is presented in figure 3.5. The first iterations are efficient and the algorithm looks to converge in few iterations.

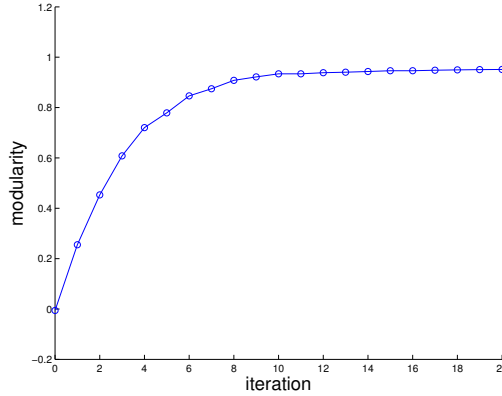


Figure 3.5: Evolution of the modularity during the iterations of the RTR algorithm on the Power Europe’s graph. The embedding dimension is $k = 3$.

Before presenting the discrete static algorithm, note that all the static algorithms that have been presented hold to solve problem (P) independently of the definition of $\mathbf{B}$ provided that it is a symmetric matrix.

3.4 Discrete static algorithm

In this section, we consider that the user does not care about a fuzzy solution, he wants a partition of the nodes into communities, i.e. each node is in one and only one community. It means that the solution is feasible for the problem (Pd).

In order to obtain a solution $\mathbf{L}$ of (Pd) from a fuzzy solution $\mathbf{H}$ of (P), a clustering algorithm is used: each cluster will correspond to a community.

A popular and efficient clustering algorithm is the k -means algorithm. A drawback of this algorithm is that it takes in input the number of clusters, i.e. communities. Thus, in this section, it is assumed that the number of communities c is known in advance.

Note that one can prefer to use, instead of k -means, another algorithm that does not take the number of clusters in input. This option is rejected because this kind of algorithm is generally significantly slower.

3.4.1 k -means algorithm

In this section, we present the k -means algorithm initially presented in [MacQueen et al., 1967].

From a data set $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ and from a number of clusters $k \leq n$ fixed a priori, the k -means algorithm returns a partition $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k\}$ of the data set. This can be written $\cup_{i=1}^k \mathcal{P}_i = \{1, \dots, n\}$ and $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset$ if $i \neq j$.

The mean of the data clustered in $\mathcal{P}_i$ is written $\mathbf{m}_i$. It is

$$\mathbf{m}_i := \frac{1}{|\mathcal{P}_i|} \sum_{j \in \mathcal{P}_i} \mathbf{x}_j. \quad (3.10)$$

Optimality criterion

The k -means clustering problem is to find the partition $\mathcal{P}$ solution of

$$\min_{\mathcal{P}} \sum_{i=1}^k \sum_{j \in \mathcal{P}_i} \|\mathbf{x}_j - \mathbf{m}_i\|^2.$$

It was shown that this problem is NP-hard (see for example [Aloise et al., 2009]). The k -means algorithm is an heuristic approach to tackle this problem.

The algorithm

The k -means algorithm consists of three steps:

Initialization step: We define k means $\mathbf{m}_1, \dots, \mathbf{m}_k$. There are different ways to do this initialization, if a good estimate of the means is known in advance, it can be used. If not, an easy way to generate it is to select k points in the data set and use them for the initial means.

Assignment step: Each data point $\mathbf{x}_j$ is assigned to the cluster with the nearest mean. It is

$$\mathcal{P}_i = \{\mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i\| \leq \|\mathbf{x}_j - \mathbf{m}_l\| \text{ for } l = 1, \dots, k\}$$

for $j = 1, \dots, k$. Note that if $\mathbf{x}_j$ can be assigned to more than one cluster, only one of them is chosen.

Update step: Each mean $\mathbf{m}_i$ is updated by the mean of the new cluster $\mathcal{P}_i$ using formula (3.10).

Repetition: The assignment step and the update step are repeated until the $\mathbf{m}_i$ do not move anymore.

Convergence

This algorithm converges toward a local optimum in a finite number of steps. Unfortunately, there is no guarantee that it converges toward a global optimum. The classical solution to this drawback is to run several times the algorithm with different initial means and take the best solution.

Complexity

The time complexity of the k -means algorithm is linear in the number of nodes n [Steinbach et al., 2000].

3.4.2 About isolated nodes

It is crucial to take care of isolated nodes. Indeed, as shown by the following result, they do not influence the objective function and then, their corresponding line of $\mathbf{H}$ is arbitrary. That means that such line can be added to any cluster during k -means and then degrade the quality of the modularity obtained post-clustering. The solution to this problem is simple: suppress the isolated nodes from the adjacency matrix and consider them as communities of single node.

Result 10. *Suppress an isolated node to a graph does not change Q_k^* .*

The proof of this result can be found in appendix A.1.6.

3.4.3 The algorithm

Let's present the discrete static algorithm.

To recall, each line of $\mathbf{H} \in \mathbb{R}^{n \times k}$ corresponds to a node and then, the clustering is executed considering each line of $\mathbf{H}$ as a vector in $\mathbb{R}^k$.

The discrete static algorithm is

1. Suppress isolated nodes of the graph. One can consider them as single-node communities;
2. Find a solution $\mathbf{H}$ of problem (P). One can use one of the static algorithms presented in previous sections;
3. Apply the k -means algorithm on the lines of $\mathbf{H}$ for c clusters (c is given).

3.5 Summary

This chapter presents four algorithms to solve problem (P). The Node by node algorithm with a standard version (NBN) and a preconditioned one (NBN-p); the total update algorithm (TU) and finally the Riemannian trust-region algorithm (RTR). Finally, in the discrete static algorithm, these static algorithms are used to find a solution $\mathbf{H}$ of (P) that is transformed into a solution of the discrete problem (Pd) by a clustering algorithm. Such a discrete solution indicates the community structure of the graph.

Note that all the static algorithms can be used to solve problem (P) independently of the definition of the matrix $\mathbf{B}$, provided that it is a symmetric matrix. Similarly, the discrete static algorithm may be used to solve the discrete problem (Pd) under the same assumption on $\mathbf{B}$ and if the number of rows of $\mathbf{L}$ is known.

The new contributions of this work are the NBN algorithm: even if the idea of random block-coordinate ascent methods is old, the choice of the blocks and the analytic solution of the one node problem are original contributions. In addition, the result 8 that permits to identify efficiently the values of N such that $\mathbf{c}^N \neq 0$ is a contribution. Concerning the NBN-p algorithm, the fact that the problem (P) is invariant under the addition of a diagonal matrix was known, but the choice of the diagonal matrix and the idea to use it to precondition the problem is new. The total update algorithm is an innovation. Recall that the Riemannian trust-region algorithm comes from [Absil et al., 2008].

Chapter 4

Dynamic algorithms

This chapter focuses on the algorithms designed to track communities in a time-varying graph.

In the case of a static graph, the Louvain algorithm (see [Blondel et al., 2008]) that will be presented in section 4.2.1 is really fast and we do not expect to outperform it. However, the Louvain algorithm has the failure of not being dynamic in the sense that if the solution for a graph $\mathcal{G}$ is found using Louvain and that after this, $\mathcal{G}$ is slightly modified, the previous solution cannot be used to find communities in the new graph. It is due to the fact that the Louvain algorithm constructs a whole hierarchy of communities, i.e. it constructs communities but also communities of communities and so on. A consequence is that adapting the communities requires destructing the whole community hierarchy.

Recall that the main idea of this thesis to track communities is to solve the relaxed problem (P) with one of the static algorithms presented in the previous chapter and then, recover a solution of the discrete problem (Pd) from the solution of the relaxed problem that has been found. This last step is made using a clustering algorithm, we use the k -means algorithm (see section 3.4.1). The solution of problem (Pd) contains the community structure of the graph.

It is assumed that the graphs evolve step-wise, i.e. the time is discretized. Note that in many practical cases, the time is naturally discrete. For example, on the Facebook friendship's graph, when a friend is added or removed or when an account is created or suppressed, it is all discrete events that occurs punctually and not progressively. Otherwise, note that this discrete time elapses non-uniformly in comparison to the *real time*. Otherwise, one can prefer to update the state of the graph periodically.

Two cases of interest are considered. The first one is the fuzzy case where one looks for a solution of (P) and the second one is the discrete case where one looks for a solution of (Pd) that reveals the community structure of the graph.

4.1 Fuzzy dynamic algorithm

It is assumed here that the user is satisfied with a fuzzy solution, i.e. a solution $\mathbf{H}$ of (P) that lives in $(\mathbb{S}^{k-1})^n$. Recall that the way to interpret a fuzzy solution was developed in section 2.2.

In this case, it is pretty easy to construct a dynamic algorithm. The only difficulty resides in the fact that the size of the adjacency matrix can evolve in time. In the case where nodes are added, it is assumed that they are added at the end of the adjacency matrix, i.e. they receive the last numbers.

Let's present the following fuzzy dynamic algorithm:

1. Update $\mathbf{A}$;
2. Adapt the size of $\mathbf{H}$ to the current size of the graph: add a normed random row if a node is added; suppress the corresponding row if a node disappears;
3. Solve the problem (P) using a static algorithm with the previous solution as initial iterate.

These three steps are repeated at each update of the graph. A random feasible matrix $\mathbf{H}$ can be used for the initialization.

The static algorithm can be any algorithm presented in chapter 3 or another one with the same specifications.

A pseudocode version of this algorithm is available in appendix B.3.

4.2 Discrete dynamic algorithm

This discrete dynamic case is similar to the discrete static case but now, the graph evolves in time. Another distinction is that we take off the assumption that the number of communities is known in advance. Indeed, in a lot of real and practice problems, we have no precise idea of the number of communities that will emerge from the algorithm. Instead of this strong assumption, we assume a weaker one: the number of communities does not evolve *too much* from a time to another. If the graph does not change too much between two steps, it looks natural that the number of clusters c does not evolve neither. This reasoning makes this assumption reasonable.

4.2.1 Louvain algorithm

In this section the Louvain algorithm of [Blondel et al., 2008] is presented. It is an heuristic process that forms communities trying to maximize the modularity.

It is an iterative algorithm that repeats successively two phases.

Phase 1: Initially, each node is assigned to a different community i.e. there is n communities, one by node.

For each node $i \in V$, consider the adjacent vertices j of i . Compute the gain of modularity that would occur if i was removed of its community and added to the community of j . Then i is assigned to the community with the greatest gain of modularity only if this gain is positive. Otherwise, if no change of community of i increases the modularity, i stays in its original community. Repeat this process until the modularity can not be increased by moving one node. Note that each node will generally be considered several times.

Phase 2: In phase 2, a graph with as many nodes as the communities at the end of phase one is constructed. In this new graph, the algorithm assigns an edge between nodes i and j with weight given by the sum of the weights of the edges between the corresponding communities in the original graph. In addition, a self-loop is assigned to a node with weight given by the sum of weights of edges between nodes of the corresponding community at the end of phase 1.

Repetition: A combination of phases 1 and 2 is called a pass. At the end of phase 2, the phase 1 can be applied on the new graph constructed during phase 2. The algorithm terminates when phase 1 does not move any nodes from its community.

Comments on the algorithm

This algorithm is very fast: applied on a graph with 118 million nodes took only 152 minutes [Blondel et al., 2008]. The limit size of the problem is given by the storage capacity and not by the computation time. This is due to the fact that, in practice, the number of nodes decreases very fast. The main part of the running time occurs during the first passes.

In addition to this rapidity, the algorithm is well known for its very good accuracy. Indeed, the found solutions are of high modularity.

An interesting property of this algorithm is that it returns the complete hierarchical structure of the graph. Indeed, after the first pass, it finds communities; after the second pass, it finds communities of communities, etc.

Note that another advantage of this algorithm is that it does not require knowing the number of communities in advance.

Complexity

The time complexity of the Louvain algorithm is unknown but some computer simulations suggest that it is linear on typical sparse data [Blondel et al., 2008].

4.2.2 Discrete dynamic algorithm

The idea is to approximate the number of clusters using the number of communities given by the Louvain algorithm. In order to do this, the Louvain algorithm is called to refresh the number of clusters once from time to time. In the light of this, a parameter $T \in \mathbb{N}_0$ is introduced. After that the Louvain algorithm has been called to estimate the number of communities c , this estimation is considered as acceptable for the next $T - 1$ steps. During these steps, the value of c is given in argument to k -means. Doing this, the number c of clusters is updated all T steps.

As mentioned for the discrete static algorithm, the position of an isolated node $i \in V$ on the sphere $\mathbb{S}^{k-1}$ does not impact the objective value (see result 10) and then, they can be incorporated in an arbitrary cluster and degrade the quality of the post-clustering solution $\mathbf{L}$. To cope with this issue, it is assumed that during the whole evolution of the graph, it does not contain any isolated node. In the light of section 3.4.2 about isolated nodes, this assumption is not restrictive.

Note that the number of clusters to give to the k -means algorithm is the number of communities c and not the embedding dimension k .

Concerning the initialization step of the k -means algorithm, it makes sense to initialize the means as the means at the previous time. Doing this would speed up the clustering. When no initial means are known because the number of clusters has changed, k -means is replicated r times to ensure a good clustering.

The discrete time $t \in \mathbb{N}$ begins at $t = 0$. It is assumed that an initial iterate $\mathbf{H}$ that is feasible for the relaxed problem (P) is available for the initial time $t = 0$.

Putting all these ideas together, one finds the following discrete dynamic algorithm:

1. Update $\mathbf{A}$;
2. Adapt the size of $\mathbf{H}$ to the current size of the graph: add a normed random row if a node is added; suppress the corresponding row if a node disappears;
3.
 - If $t \bmod T = 0$, call the Louvain algorithm to update the number of communities c ; the solution outputted by Louvain is the current discrete solution $\mathbf{L}$;
 - If $t \bmod T = 1$, solve the problem (P) using a static algorithm with the previous solution $\mathbf{H}$ as initial iterate. Then use k -means without initial means but with r replications. Keep in memory the final means of k -means;
 - If $t \bmod T > 1$, solve the problem (P) using a static algorithm with the previous solution $\mathbf{H}$ as initial iterate. Then use the k -means algorithm with the means found at the previous call of k -means.

These steps are repeated at each update of the graph.

A pseudocode version of this algorithm is available in appendix B.4.

4.2.3 Comments on the algorithm

If T is chosen too large, the estimation of the number of communities c will be bad and the quality of $\mathbf{L}$ will be poor but if T is too small, the algorithm will be unnecessarily slowed down.

Note that the k -means algorithm can be replaced by any clustering algorithm. Similarly, the Louvain algorithm can be replaced by any clustering algorithm that does not need to know c in advance. If one does this, he has to take care of update $\mathbf{H}$ and after to give $\mathbf{H}$ to the clustering algorithm.

Concerning ϵ , the tolerance on the stopping criterion of the static algorithm, it makes sense not to choose it too small. Indeed, during the last iterations, the rows of $\mathbf{H}$ will not change a lot. That means that even if the fuzzy solution $\mathbf{H}$ is better, the post clustering solution $\mathbf{L}$ will probably stay the same and the algorithm would be slowed down unnecessarily.

In this algorithm, the user provides an initial solution $\mathbf{H}^0$. In practice, the user can have no idea of what is a *good* initial solution and then he must resolve to use a random initial solution. Note that in that case, the first step can be much longer than the others because more iterations will be needed in the static algorithm. If the graph evolves pretty fast (the real time between two discrete steps is short), the algorithm can be too slow and not be able to follow the communities in real time. If the graph's evolution is fast but not too much, this behavior can be dampened progressively and the algorithm will catch up the delay. Otherwise, if the evolution is slower, the algorithm will track communities in real-time. This remark also applies on the Fuzzy case.

At time $t = 0$, it holds that $t \bmod T = 0$ and then c is updated. It is necessary because initially, c is unknown. Note that during such a step, the matrix $\mathbf{H}$ is not updated because Louvain gives $\mathbf{L}$ directly. It would not be surprising if for the direct next step (when $t \bmod T = 1$), the number of iterations and then the running time were greater than the normal. In addition to this, k -means is repeated r times, this can also increase the running time.

4.3 Summary

This chapter has presented some methods to solve problems (P) and (Pd) with dynamic algorithms. The last one corresponds to track communities in time-varying networks, it is made by the discrete dynamic algorithm that can be seen as the goal of this thesis. This heuristic process is a personal contribution of this work.

Chapter 5

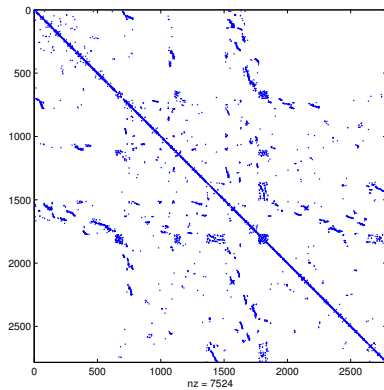
Comparison of the algorithms

In this section, the efficiency of the algorithms are compared. The performance of an algorithm takes into account the quality of the outputted solution, i.e. the modularity's height, but also the running time required to achieve this solution. All the algorithms are implemented in MATLAB.

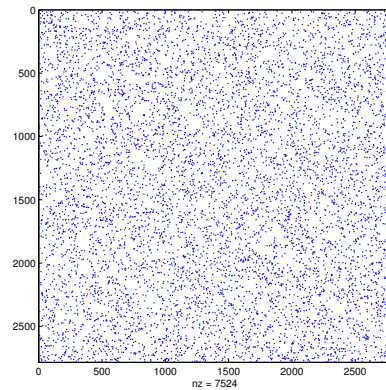
This chapter is split into three sections, the first one compares the different static algorithms, in this case, the outputted solutions does not form a partition of the nodes. The second section focuses on the discrete static algorithm. It is launched separately with the different static algorithm. The results are compared to the one obtained using the Louvain algorithm. Finally, in the last section, the different versions of the discrete dynamic algorithms are compared. It is similar to the previous section but in the case of time-varying networks.

A special attention is given to the sparsity of the graphs that are studied. The sparsity of a matrix is defined as the number of null entries of the matrix over the total number of entries. A sparsity of 100% is reached by the null matrix $\mathbf{0}_{n,n}$.

Two different graphs are introduced. The first one is the *Power Europe's graph* [Rosas-Casals et al., 2007]. It is a real-world network that represents the electrical lines between producers and consumers in Europe. The adjacency matrix of this graph is depicted on figure 5.1a, it contains $n = 2783$ nodes and $m = 3937$ edges. Its sparsity is of 99.90%. This graph is our sparse example. In order to prevent any cheating, the adjacency matrix is shuffled in order to destroy the community structure that appears on figure 5.1a. The shuffled version of the adjacency matrix is presented on figure 5.1b.



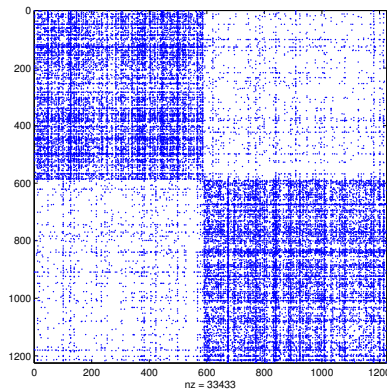
(a) Structured order.



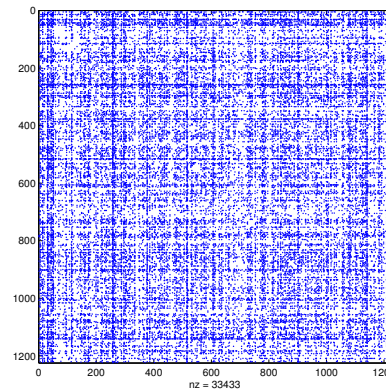
(b) Shuffled order.

Figure 5.1: Nonzero entries of the adjacency matrix of the Power Europe's graph.

The second graph is the *Polblog's graph* [Adamic and Glance, 2005], it is also a real-world network. The nodes are political bloggers and edges represent the links between the blogs. It dates backs to two months before the U.S. presidential election of 2004. Its adjacency matrix is depicted on figure 5.2a and a shuffled version is on figure 5.2b. This graph has $n = 1224$ nodes and $m = 9545$ edges. Its sparsity is of 97.77%. It is less sparse than the Power Europe graph. The Polblog's graph is considered as a *not too sparse* example.



(a) Structured order.



(b) Shuffled order.

Figure 5.2: Nonzero entries of the adjacency matrix of the Polblog's graph.

One can observe that the Power Europe's graph has more nodes but that the Polblog's graph has more edges. This property permits highlighting the limiting element of an algorithm: is it mainly slowed down by the number of nodes or by the number of edges?

Finally, to separate the best algorithms, a third bigger graph is used. It is the *US Power Grid's graph*. Similar to the Power Europe's graph, it is a representation of the

electrical network in the United States. Its adjacency matrix is presented on figure 5.3a and a shuffled version is on figure 5.3b. It is significantly bigger than the two first graphs because it is composed of $n = 4941$ nodes and $m = 6594$ edges. It is a very sparse graph, indeed, its sparsity is 99.94%.

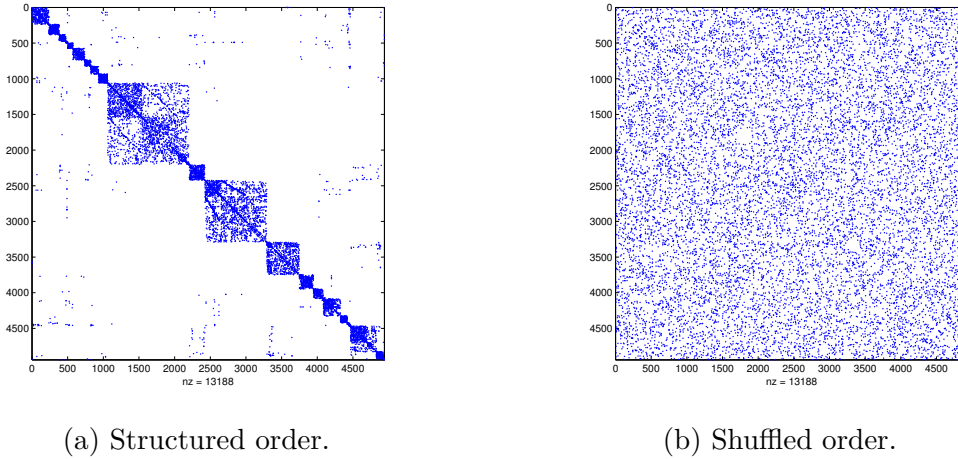


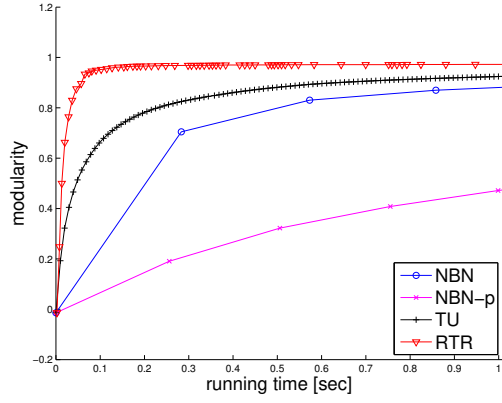
Figure 5.3: Nonzero entries of the adjacency matrix of the US Power Grid's graph.

5.1 Static algorithms

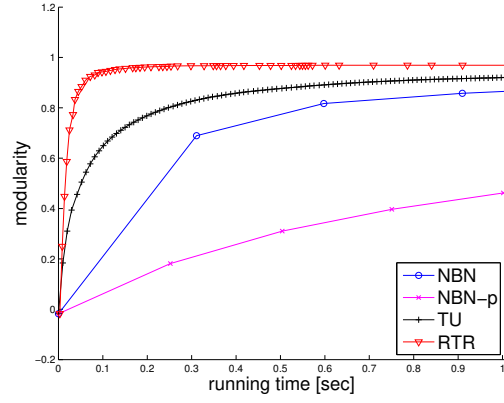
This section focuses on the static algorithms presented in chapter 3. They are compared in the fuzzy case and finally in the discrete case where the solution has been clustered. In this last case, the algorithms are compared to the Louvain algorithm.

One way to do the comparison for the fuzzy case is to run all the algorithm and to compare the time used by each one and the modularity that is found. Such an approach works globally but hides the internal efficiency of the algorithm in the sense that it does not reveal the performances *at the beginning* or *at the end* of the iterations. Indeed, maybe that some algorithms are faster for the first iterations while others are better when the current solution is close to the optimal one. In order to reveal those properties, we compare modularity throughout time.

Figure 5.4 presents a graph modularity versus time throughout iterations with the different algorithms with the embedding dimension $k = 2$. The computations are applied to the power Europe graph ($n = 2783$, $m = 3937$) and all algorithms start with the same initial solution $\mathbf{H}^0$ that has been generated randomly. Figure 5.5 presents the same thing but with $k = 10$. We observe that at any time, the RTR algorithm gives a better modularity than all the other algorithms presented here. We conclude that it outperforms the others. Similar results are obtained with other graphs.

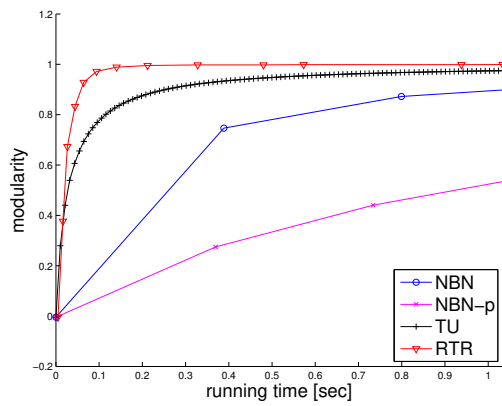


(a) Power Europe's graph.

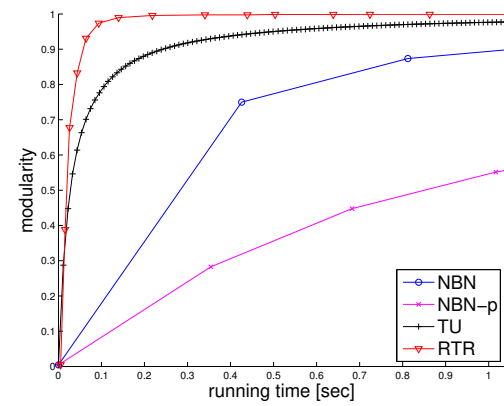


(b) Polblog's graph.

Figure 5.4: Evolution of the modularity Q_k^* achieved by the different static algorithms in the relaxed case versus the time required to obtain it. The embedding dimension k is fixed to 2.



(a) Power Europe's graph.



(b) Polblog's graph.

Figure 5.5: Evolution of the modularity Q_k^* achieved by the different static algorithms in the relaxed case versus the time required to obtain it. The embedding dimension k is fixed to 10.

5.2 Discrete static algorithm

A really efficient algorithm for community detection in networks is the Louvain algorithm that is presented in section 4.2.1. This algorithm is compared to the discrete static algorithm of section 3.4. We use the MATLAB implementation of the Louvain algorithm written by Antoine Scherrer. It is known to be suboptimal, indeed, a C++ implementation exists and is faster. Otherwise, all the other algorithms are in MATLAB so it makes sense to do the comparison in the same programming language.

The Louvain algorithm does not require the number of communities in advance and, it is used to know the number of communities to consider in the discrete static algorithm. The discrete static algorithm uses a static algorithm, these four algorithms presented in chapter 3 are tested.

It is set that the static algorithms stop when the step length of two successive iterates has a norm smaller than a tolerance, it is $\|\mathbf{H} - \mathbf{H}_{prev}\|_F < 10^{-4} \cdot \sqrt{nk}$ where $\|\cdot\|_F$ refers to the Frobenius norm. The last factor is used to be invariant under the size of matrix $\mathbf{H}$. In addition, the Riemannian trust-region algorithm has a second stopping criterion based on the Frobenius norm of the gradient. The tolerance is fixed at $10^{-10} \cdot \sqrt{nk}$.

5.2.1 Power Europe's graph ($n = 2783$, $m = 3937$)

Table 5.1 presents the time and the modularity for our different algorithms and for Louvain applied on the Power Europe's graph. They are executed for values of the embedding dimension k from 2 to 30. Only even values of k are tested to save time. Note that the number of communities given to k -means is the number of communities found by Louvain. The best performances of each algorithm are highlighted by shaded cells.

For the Power Europe's graph, the Louvain algorithm finds a solution of modularity 0.73 in 73.15 seconds. In comparison, the discrete static algorithm finds some solutions of modularity about approximately 0.90 independently of which static algorithm is used. It is a significant improvement of 23.3%.

When the NBN algorithm is used as the static algorithm, the discrete static algorithm finds solutions very slowly because it needs more than 280 seconds in the faster case. It is slower than the Louvain algorithm but gives a higher modularity. Concerning the dependency in the embedding dimension k , the high of the found modularity grows when k increases from 2 to 3 and stays more or less constant after. The running time is about the same range for any value of k , except for $k = 2$ for which the running time is approximately five times longer.

The NBN-p variant of the discrete static algorithm finds a solution of modularity 0.90 in 45.36 seconds. This solution is better than the one found by the Louvain algorithm and it is faster. For the NBN-p algorithm, the solutions of modularity 0.9 are found only when $k \geq 18$. Note that the running time grows with the embedding dimension k .

k	NBN		NBN-p		TU		RTR		Louvain	
	mod	time	mod	time	mod	time	mod	time	mod	time
2	0.61	1391.75	0.42	26.72	0.57	23.39	0.66	2.88	0.73	73.15
4	0.89	347.95	0.77	26.63	0.89	25.26	0.89	6.70		
6	0.89	322.82	0.83	30.70	0.89	19.87	0.89	10.60		
8	0.89	331.21	0.85	31.96	0.90	17.42	0.89	9.83		
10	0.89	338.04	0.86	35.02	0.90	16.08	0.89	11.79		
12	0.89	310.79	0.88	38.61	0.90	16.04	0.89	10.33		
14	0.89	280.97	0.88	40.71	0.90	14.37	0.89	13.24		
16	0.90	345.04	0.88	42.31	0.91	14.21	0.89	10.16		
18	0.89	346.22	0.90	45.36	0.91	13.22	0.89	16.60		
20	0.88	317.14	0.90	45.61	0.91	11.61	0.89	10.55		
22	0.89	341.39	0.90	46.23	0.91	11.74	0.89	13.05		
24	0.89	311.81	0.90	48.28	0.91	11.99	0.89	12.10		
26	0.89	323.29	0.90	51.32	0.91	12.59	0.89	14.27		
28	0.89	323.07	0.90	52.64	0.91	11.11	0.89	16.20		
30	0.89	355.34	0.90	58.31	0.90	11.65	0.89	23.64		

Table 5.1: Comparison on the Power Europe’s graph of the different versions of the discrete static algorithm and the Louvain algorithm. Both modularity and running time (in seconds) are presented for some values of the embedding dimension k . Note that the Louvain algorithm is independent of k . The shaded cells indicate the best performances of each algorithm.

The discrete static algorithm with the TU algorithm as static algorithm gives a modularity of 0.91 in 11.11 seconds. It is a quality approximately 23% higher than the one found by Louvain and the solution is found more than six times faster. In addition, one can observe that both the quality of the solution and the running time are not very sensitive to the value of the parameter k provided that it is not too small. This robustness makes that a user do not have to adjust the value of k but can just set it sufficiently large. Finally, for sufficiently small values of k , the running time decreases when k becomes larger. It is unexpected because the number of variables increases and then, a longer running time could be expected. A possible explanation of this phenomenon is that, when the embedding dimension k increases, there are more directions to explore and then, an update step can reach a better solution.

One can observe that the discrete static algorithm using the RTR algorithm finds solutions of modularity slightly smaller than the other version of the discrete static algorithm. It can be explained by the additional stopping criterion. Unfortunately, if it is suppressed, it occurs that the algorithm stops very late so this stopping criterion must be kept. A solution of modularity 0.89 is found in only 6.70 seconds with $k = 4$. For greater values of k , the modularity does not change but the running time generally increase. A consequence of this observation is that the value of k is a bit sensitive. If it is set too high, the algorithm is slowed down.

The results obtained with the TU and the RTR variants are encouraging in comparison to the results obtained with the Louvain algorithm.

5.2.2 Polblog's graph ($n = 1224$, $m = 9545$)

The same tests are executed on the Polblog's graph, i.e. the less sparse graph. The results are presented on table 5.2.

The results on the Polblog's graph are not in the advantage of the discrete static algorithm, in every case the modularity obtained is smaller than 0.43, the one obtained by the Louvain algorithm in 7.95 seconds.

Both the NBN and NBN-p variants of the discrete static algorithm give smaller modularity in more time than Louvain. They are totally outperformed. Note that the modularity find by the NBN-p algorithm is systematically less than 0.2.

For the TU variant, the algorithm is systematically faster than Louvain. Unfortunately, the quality of the solution is significantly worse than with Louvain. Indeed, the best case is when $k = 12$ and the modularity is about only 0.39. Note that as previously, the running time is not significantly affected by the value of k when $k \geq 4$.

Finally, the RTR variant takes generally more time, i.e. more than 15 seconds but finds solutions of better quality than the TU algorithm. Except when $k = 2$, this algorithm is outperformed in time and in modularity by the Louvain algorithm. The case $k = 2$ gives a modularity of 0.40 in 1.92 seconds which is not too bad.

One can ask what produces such a difference between the results on the Power Europe's

k	NBN		NBN-p		TU		RTR		Louvain	
	mod	time	mod	time	mod	time	mod	time	mod	time
2	0.39	15.37	0.03	14.81	0.33	6.23	0.40	1.92	0.43	7.95
4	0.39	31.31	0.07	15.05	0.38	4.97	0.39	16.73		
6	0.42	19.61	0.11	15.33	0.39	4.26	0.39	19.14		
8	0.39	19.34	0.12	15.74	0.37	3.63	0.38	22.07		
10	0.41	18.51	0.11	16.16	0.36	4.53	0.39	17.65		
12	0.42	14.19	0.16	15.47	0.39	3.13	0.39	25.77		
14	0.42	17.47	0.16	18.42	0.37	3.63	0.41	30.61		
16	0.41	13.92	0.15	17.93	0.36	3.54	0.42	22.57		
18	0.40	10.92	0.15	19.02	0.37	3.54	0.42	24.74		
20	0.40	12.80	0.17	18.58	0.35	3.59	0.39	40.02		
22	0.40	11.79	0.17	18.04	0.36	3.13	0.38	41.16		
24	0.41	14.95	0.18	19.34	0.38	3.23	0.39	44.13		
26	0.42	15.29	0.18	20.31	0.38	3.25	0.42	51.74		
28	0.41	11.75	0.19	20.72	0.35	3.37	0.39	31.38		
30	0.42	13.83	0.18	22.28	0.36	3.16	0.42	24.01		

Table 5.2: Comparison on the Polblog’s graph of the different versions of the discrete static algorithm and the Louvain algorithm. Both modularity and running time (in seconds) are presented for some values of the embedding dimension k . Note that the Louvain algorithm is independent of k . The shaded cells indicate the best results of each algorithm.

graph and on the Polblog's graph. We suspect the sparsity of the graph to be the discriminating factor. This intuition comes from the fact that the Louvain algorithm makes loops on the number of nodes n when the expansive part of TU and RTR algorithms are matricial products with sparse matrix. The number of operations to execute such products are controlled by the number of edges of the graph $\bar{m}$. This means that on a very sparse graph, the TU and the RTR algorithms could take advantage on the Louvain algorithm.

5.2.3 US Power Grid's graph ($n = 4941$, $m = 6594$)

This explanation is tested for the two best algorithm, the TU and the RTR variants, on a greater and sparser graph: the US Power Grid's graph. Results are presented on table 5.3.

k	TU		RTR		Louvain	
	mod	time	mod	time	mod	time
2	0.64	51.84	0.66	10.08	0.75	327.59
4	0.91	39.28	0.91	24.15		
6	0.92	31.10	0.91	26.58		
8	0.92	30.16	0.90	38.89		
10	0.92	27.08	0.91	46.52		
12	0.92	24.94	0.91	51.40		
14	0.93	21.19	0.91	49.86		
16	0.92	20.95	0.91	79.38		
18	0.93	21.99	0.91	70.81		
20	0.92	19.88	0.90	70.10		
22	0.93	20.59	0.91	109.24		
24	0.92	20.27	0.91	60.49		
26	0.93	19.51	0.91	128.38		
28	0.92	18.70	0.91	105.29		
30	0.93	21.37	0.92	91.33		

Table 5.3: Comparison on the US Power Grid's graph of the two best versions of the discrete static algorithm and the Louvain algorithm. Both modularity and running time (in seconds) are presented for some values of the embedding dimension k . Note that the Louvain algorithm is independent of k . The shaded cells indicate the best results of each algorithm.

As soon as the embedding dimension k is greater than 4, both the TU and the RTR variants outperform the Louvain algorithm in both modularity and running time. The Louvain algorithm finds a solution of modularity 0.75 in 327 seconds when the others find faster a modularity greater than 0.9. As on the Power Europe's graph, the modularity that is reached does not depend too much of the embedding dimension k provided that it is sufficiently large for both variants. When RTR is used, one can observe that the running time generally grows with k . This means that the value of

k has to be adjusted. On the contrary, when TU is used, an increasing of k has no impact or a positive impact on the running time. This makes the use of the TU variant easier than the RTR one. In addition, when $k \geq 8$, the TU variant outperforms the RTR variant in both modularity and running time. Finally, one can observe that the best performance of TU is better than the best performance of RTR.

Coming back to the comparison with the Louvain algorithm, the TU variant finds a solution of modularity 0.93, i.e. 24% higher than Louvain, in only 19.5 seconds, i.e. more than 16 times faster than Louvain.

In conclusion, the results of this section seem to indicate that the discrete static algorithm used with the total update algorithm (TU) as static algorithm outperforms the Louvain algorithm in both modularity and running time if the graph is sufficiently sparse. In addition, this method is sufficiently robust to a variation of the parameter k , ensuring that the choice of a *good value* for k is not an issue. Recall that the algorithms presented in this thesis have been initially designed to be competitive only in the case of time-varying networks so this result is quite surprising.

Before closing this section, recall that the discrete static algorithm needs to know in advance the number of communities c , this is not the case of the Louvain algorithm. In addition, the discrete static algorithm is so efficient when the graph is sparse that it can be faster than Louvain to use it even if the number of clusters is unknown and try different values of c and to keep only the best one. Doing this can permit finding communities even if the number of communities is unknown, maybe faster than with the Louvain algorithm.

5.3 Discrete dynamic algorithm

The moment of the verdict draws nearer on this section. The different versions of the discrete static algorithm (one for each static algorithm) are compared to the Louvain algorithm in the case of time-varying networks.

Three different cases are considered.

The first one is when $t \bmod T = 0$, in this case the algorithm only calls the Louvain algorithm and then no comparison is required.

The second case is when $t \bmod T = 1$, in this case the algorithm calls the static algorithm and replicates r times the k -means algorithm. It is considered that this case is very similar to an application of the discrete static algorithm. According to this, the reader can refer to section 5.2. Note that in practice, one can hope slightly better results in the dynamic case because the initial iterate $\mathbf{H}$ is not fully random but it is the solution obtained two time steps before.

Finally, when $t \bmod T > 1$, the algorithm call a static algorithm with the previous solution $\mathbf{H}$ as initial iterate and then call the k -means algorithm only one time, with the centroids previously found. This situation is the only one that has not yet been

studied and it is the case of interest of this section.

There is no standard Benchmark for time-varying networks so there is no obvious way to compare the algorithms. We follow the heuristic process presented in [Van Den Eeckhaut, 2016].

Let's compute an optimal solution $\mathbf{H}_{\text{exact}}$ using one of the static algorithms. Instead of modifying the graph and use $\mathbf{H}_{\text{exact}}$ as initial iterate, the graph is left unchanged but the initial iterate is a modified version of $\mathbf{H}_{\text{exact}}$. More precisely, 10% of the rows of $\mathbf{H}_{\text{exact}}$ are replaced by random rows with unit norm. Doing this, the obtained initial solution is feasible.

In [Van Den Eeckhaut, 2016], it is explained why this procedure is equivalent to all modifications that can occur during the evolution of a time-varying network.

“The fact of replacing a row by a random one can represent the appearance of a node into the network, by assuming that this node was not present at the previous execution of the algorithm, or the disappearance and appearance of some edges that made the node change of communities. As for the disappearance of a node, it translates into the assumption that some rows of the initial assignment matrix have been removed before using it as starting point. Note that this transformation is quite significant compared to what is generally observed in practice.”

The different versions of the discrete dynamic algorithm presented in section 4.2.2 (one for each static algorithm: NBN, NBN-p, TU and RTR) are compared to the Louvain algorithm. Both the modularity and the running time are presented, they are obtained as the mean on 5 executions of the algorithms. They are run for different values of the embedding dimension k .

5.3.1 Power Europe's graph ($n = 2783$, $m = 3937$)

The results obtained on the Power Europe's graph are presented in table 5.4.⁸

When the embedding dimension k is such that $k \geq 4$, all the variants of the discrete dynamic algorithm outperform the Louvain algorithm in both modularity and running time. They give all approximately the same value of modularity. One can observe that as soon as $k \geq 4$, the TU variant outperforms all the others. In addition, as in the static case, the TU variant is robust to a variation of k .

When $k = 14$, the TU variant of the discrete dynamic algorithm finds a modularity about 0.9 in 1.57 seconds, it is a quality 21.6% higher found 43 times faster.

⁸One can observe that the modularity and the running time of Louvain are not the same than on table 5.1. It is due to the fact that the running time on a computer depends on external factors and that the random shuffling of the nodes is different.

k	NBN		NBN-p		TU		RTR		Louvain	
	mod	time	mod	time	mod	time	mod	time	mod	time
2	0.61	208.76	0.60	24.50	0.61	5.08	0.61	4.74	0.74	67.73
4	0.89	30.79	0.88	24.67	0.89	2.17	0.89	13.18		
6	0.89	29.51	0.89	27.71	0.89	1.85	0.89	17.71		
8	0.89	27.93	0.89	28.66	0.89	1.83	0.89	17.71		
10	0.89	27.50	0.89	31.59	0.89	1.64	0.89	19.71		
12	0.89	27.35	0.89	32.67	0.89	1.60	0.89	25.14		
14	0.89	28.19	0.89	35.33	0.90	1.57	0.89	30.65		
16	0.89	26.53	0.89	36.74	0.89	1.51	0.89	32.80		
18	0.89	28.65	0.89	39.38	0.89	1.42	0.89	31.09		
20	0.88	27.50	0.89	40.55	0.89	1.38	0.88	33.89		
22	0.89	29.29	0.89	43.24	0.89	1.40	0.89	37.66		
24	0.89	28.43	0.89	44.22	0.89	1.43	0.89	35.94		
26	0.89	28.88	0.89	47.45	0.89	1.29	0.89	37.34		
28	0.89	30.74	0.89	48.14	0.89	1.38	0.89	45.20		
30	0.89	30.32	0.89	50.75	0.89	1.34	0.89	43.70		

Table 5.4: Comparison on the Power Europe's graph of the different versions of the discrete dynamic algorithm and the Louvain algorithm. Both modularity and running time (in seconds) are presented for different values of the embedding dimension k . Note that the Louvain algorithm is independent of k . The shaded cells indicate the best results of each algorithm.

5.3.2 Polblog's graph ($n = 1224$, $m = 9545$)

The results on the Polblog's graph are in table 5.5.

k	NBN		NBN-p		TU		RTR		Louvain	
	mod	time	mod	time	mod	time	mod	time	mod	time
2	0.41	5.80	0.29	14.06	0.37	2.55	0.41	1.74	0.43	4.98
4	0.39	10.81	0.37	14.67	0.40	1.97	0.41	38.26		
6	0.39	8.28	0.37	15.66	0.40	1.69	0.39	57.22		
8	0.39	7.94	0.37	15.84	0.42	1.56	0.42	66.47		
10	0.42	9.34	0.39	16.41	0.41	1.61	0.41	64.80		
12	0.41	5.98	0.39	16.78	0.41	1.21	0.41	69.47		
14	0.37	10.18	0.39	17.33	0.40	1.51	0.40	62.97		
16	0.39	6.34	0.38	17.68	0.40	1.23	0.40	89.51		
18	0.36	5.50	0.39	18.31	0.41	1.32	0.40	83.98		
20	0.39	5.95	0.40	18.58	0.40	1.15	0.40	87.92		
22	0.36	5.73	0.39	18.75	0.41	1.31	0.39	81.41		
24	0.40	7.24	0.40	19.20	0.41	1.12	0.42	79.90		
26	0.39	7.42	0.40	19.71	0.40	1.19	0.39	96.34		
28	0.39	5.40	0.38	19.64	0.41	1.17	0.41	102.13		
30	0.39	6.76	0.39	20.74	0.41	1.22	0.40	106.28		

Table 5.5: Comparison on the Polblog's graph of the different versions of the discrete dynamic algorithm and the Louvain algorithm. Both modularity and running time (in seconds) are presented for different values of the embedding dimension k . Note that the Louvain algorithm is independent of k . The shaded cells indicate the best results of each algorithm.

As in the static case, the results on the Polblog's graph are not as good as on the Power Europe's graph. The NBN and the NBN-p variants are both outperformed in modularity and in running time by the Louvain algorithm. When $k = 2$, the RTR variant gives a good result because it finds a modularity 0.41 in 1.74 seconds (against a modularity of 0.43 in 4.98 second for Louvain). When k increases, its running time grows fast and as soon as $k \geq 4$, it is outperformed by Louvain in both modularity and running time.

The TU variant gives modularity around 0.40-0.41 when $k \geq 4$ in a running time generally smaller than 1.5 seconds. The modularity is smaller than with Louvain but it is reached faster. As before, the exact value of k is not determining as long as it is not too small.

5.3.3 US Power Grid's graph ($n = 4941$, $m = 6594$)

The tests on the US Power Grid's graph are presented in table 5.6. One can see that the four variants are faster than Louvain for any value of k . In addition, when $k \geq 4$,

k	NBN		NBN-p		TU		RTR		Louvain	
	mod	time	mod	time	mod	time	mod	time	mod	time
2	0.71	202.75	0.68	71.70	0.70	9.67	0.71	5.12	0.75	321.04
4	0.91	94.17	0.90	77.17	0.91	4.49	0.91	25.16		
6	0.91	97.17	0.90	92.07	0.91	4.11	0.91	31.05		
8	0.91	104.14	0.91	97.97	0.91	3.88	0.91	47.47		
10	0.91	95.53	0.91	103.41	0.91	3.52	0.91	43.13		
12	0.91	91.79	0.91	106.19	0.91	3.21	0.91	48.44		
14	0.91	98.01	0.91	115.52	0.91	3.18	0.91	62.13		
16	0.91	88.74	0.91	112.05	0.91	2.82	0.91	61.40		
18	0.91	94.21	0.90	120.46	0.91	2.69	0.91	59.09		
20	0.91	93.45	0.91	121.61	0.91	2.71	0.91	62.78		
22	0.91	96.79	0.91	130.22	0.91	2.70	0.91	66.52		
24	0.91	93.90	0.91	133.41	0.91	2.79	0.91	83.16		
26	0.91	96.83	0.91	143.60	0.91	2.67	0.91	90.95		
28	0.91	93.76	0.91	145.26	0.91	2.56	0.91	86.37		
30	0.91	97.16	0.91	154.28	0.91	2.67	0.91	84.60		

Table 5.6: Comparison on the US Power Grid’s graph of the different versions of the discrete dynamic algorithm and the Louvain algorithm. Both modularity and running time (in seconds) are presented for different values of the embedding dimension k . Note that the Louvain algorithm is independent of k . The shaded cells indicate the best results of each algorithm.

they give higher modularity. When $k = 2$, the modularity is pretty small and when $k \geq 4$, the TU variant outperforms all the others in modularity and in time. The same robustness with respect to k is observed for the TU algorithm. When $k = 28$, it finds a modularity of 0.91 instead of 0.75 with Louvain (it is 21.3% better) and it runs during 2.56 seconds instead of 321 seconds with Louvain (it is 125 times faster).

This battery of tests indicates that the total update (TU) variant of the discrete dynamic algorithm is the best of the four and is very efficient in the case of sparse graphs. In addition, the choice of a reasonable value of k is not an issue.

5.4 Summary

This chapter was devoted to the comparison of the algorithms presented in this thesis. They are compared between each other but are also compared to the Louvain algorithm.

In the case of a static network that is sufficiently sparse, the total update variant of the discrete static algorithm outperforms the other variants and the Louvain algorithm in both modularity and running time. Note that it would not be surprising that the Riemannian trust-region variant would be competitive with a better implementation,

as suggested by the modularity versus time graphs depicted on figures 5.4 and 5.5.

For a time-varying network, the discrete dynamic algorithm that uses the total update algorithm as static algorithm is better than the other variants. It is systematically faster than the Louvain algorithm but the quality of the solution, i.e. the modularity, can be poor if the graph is not sufficiently sparse. The size of the modifications of the graph between two successive time steps is another important factor. The smaller the modifications are, the more the discrete static algorithm is advantaged.

Finally, in both static and time-varying cases, the total update variant has the advantage that it is not difficult to find a good value of the embedding dimension k .

To summarize, the tests performed in this chapter seem to indicate that if the graph is sufficiently sparse or if it evolves with small modifications or both, the discrete static algorithm that uses the total update algorithm is a good alternative to Louvain and can give significantly better results.

When the graph is not sparse, the algorithms find pretty poor discrete solutions, and this independently of the static algorithm that is used and of tolerance on the stopping criterion. This seems to indicate that it is not a property of the static algorithm or its parameters but that the relaxation (P) is not well suited for such a graph.

Before closing this section, some comments are made on the validity of the conclusions presented above. Firstly, the MATLAB implementation of the Louvain algorithm that has been used is suboptimal. In addition, the Louvain algorithm has been tested on graphs much larger than the ones met in this thesis: the greater has $n = 118$ millions nodes and $m = 1$ billion edges [Blondel et al., 2008]. To definitely decide the matter, some tests would have to be done on greater graphs using a C++ implementation, for example.

Chapter 6

Further work

This chapter firstly presents some possible improvements of the work made in this thesis are proposed throughout a discussion. They are organized chapter by chapter.

Finally, we take a step back from this thesis to open new horizons. The notion of community structure is questioned and some arguments are presented to argue that solving the relaxed problem (P) does not reveal the community structure in the most intuitive sense. Then, this observation is considered in the light of the performances of the algorithms observed in the previous chapter.

6.1 Discussion

We briefly recall some results of this thesis and propose some ideas for further work.

In chapter 2, the minimal embedding dimension is introduced, along with an upper bound on it. An interesting theoretical question is to know if this bound is tight or if it can be improved. In addition, it was shown that the numerical solutions are generally degenerated. It would be interesting to have a similar result for the exact solutions.

In chapter 3, the node by node and the total update algorithm are introduced. They require the assumption that $\mathbf{c}^N \neq \mathbf{0}$ not to divide by 0 in the algorithm. It was shown that a necessary condition is to remove the isolated nodes. Even if this condition is not sufficient, no problems occur in practice. An explanation of this phenomenon would consolidate the theoretical foundations of these algorithms. As for the node by node algorithm, one can search for a good preconditioning of the problem. It could also be interesting to construct a discrete static algorithm that does not require knowing the number of communities in advance. This idea is detailed at the end of section 5.2.

In chapter 4, a description of the discrete dynamic algorithm is presented. It uses the k -means algorithm that is sometimes called r times to keep only the best solution. Finding a good value of this parameter r could have a significantly positive impact on both the running time of the algorithm and the quality of the found solution. Another parameter of the discrete dynamic algorithm is the period T between two updates

of the number of clusters. Instead of doing this update periodically, one can imagine designing a heuristic process to evaluate if an update is soon needed.

Chapter 5 compares the algorithms. Deeper tests would be useful to confirm the conclusions proposed in this chapter. One can propose to implement the algorithms in another language like C++ or PYTHON for example. Note that a C++ implementation of the Louvain algorithm exists and outperforms the MATLAB implementation. In addition, some tests on greater graphs could be lighting. Note that the MATLAB implementations of the static algorithms can probably be improved, particularly for the Riemannian trust-region algorithm. It could be interesting to test the discrete static algorithm on a real-world time-varying network.

6.2 An extended notion of communities

This section shows that the embedding on the sphere formalized by problem (P) does not necessarily reveal communities in the common sense. This is shown by an example on the bowtie graph depicted in figure 6.1. This graph has all edges of unit weight except the central one (between nodes 2 and 4) that has a weight of ϵ .

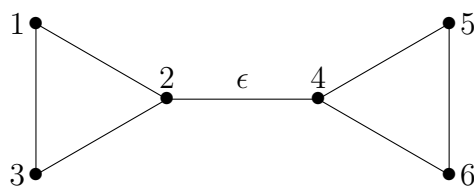


Figure 6.1: Representation of the bowtie graph. All edges have unit weight except the central one that has a weight ϵ .

The embedding of this graph is studied on the circle $\mathbb{S}^1$, i.e. the embedding dimension k is fixed to 2. Tests with greater values of k give the same results. The problem (P) is solved for this graph for different values of ϵ .⁹

When $\epsilon = 1$, we find the solution $\mathbf{H}$ depicted on figure 6.2. Only two dots appear because some nodes coincide. The coinciding nodes can be considered as being in the same community. For the value $\epsilon = 1$, two communities $\{1, 2, 3\}$, $\{4, 5, 6\}$ are found, as one would intuitively expect for the bowtie.

⁹The results that are obtained are independent of the static algorithm that is used to solve the problem.

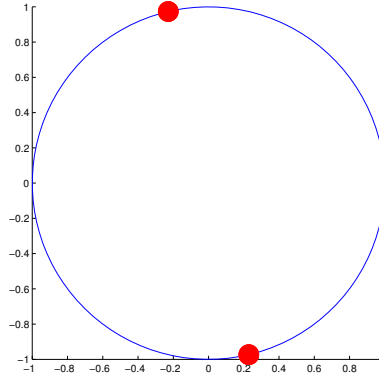


Figure 6.2: Representation of an optimal solution of problem (P) for the bowtie graph depicted on figure 6.1 when $\epsilon = 1$ and when the embedding dimension k is 2.

Something interesting appears when ϵ increases. For $\epsilon = 15$, the representation of the solution $\mathbf{H}$ is the same, i.e. only two dots appear, but the nodes are distributed differently, meaning that the communities are different. In that case, the communities C_1 and C_2 are

$$C_1 = \{2, 4\} \text{ and } C_2 = \{1, 3, 5, 6\}.$$

The community C_1 can be explained by the classic interpretation: the value of ϵ is so high that nodes 2 and 4 are very connected in comparison to the other nodes. To make this more intuitive, one can consider that these two nodes are connected with $\epsilon = 15$ edges.

Understanding community C_2 is trickier. First of all, recall that in this thesis, the notion of community has been introduced as a set of nodes densely connected and weakly connected with the rest of the graph. It is formalized by maximum modularity partitions [Newman and Girvan, 2004]. This definition implies in particular connected communities, as otherwise splitting a community into its connected components would improve the modularity, as one can easily check. In the case of C_2 , one can observe that this set of nodes is disconnected and then, it is not a community.

An explanation of such observation is that this method finds groups of nodes such that the nodes that are in a same group have similar *roles* in the graph. It is known as equitable partitions [Godsil, 1997]. In the case of our example, any nodes in C_1 has exactly one neighbor of weight ϵ in C_1 and exactly two neighbors in C_2 . Conversely, any node in C_2 has exactly one neighbor in C_2 and one neighbor in C_1 .

Deeper experiments would be necessary to confirm this explanation. In all cases, this particular behavior of the relaxation (P) can explain why, sometimes, some poor results are obtained when the algorithms presented in this thesis are used for community detection. If it can be seen as an issue, one can argue that the embedding on the sphere reveals something different from the community structure, maybe more meaningful. Some deeper study of the properties of this embedding could highlight for which task it is the most adapted.

This numerical curiosity seems to indicate that the approach presented on this thesis can be useful to understand the different roles of the nodes in a graph and not only the community structure.

Conclusion

The community detection question has been studied a lot in the last few decades because of its wide range of applications. The goal of this thesis was to design efficient algorithms to track communities in time-varying networks. Previously, there was no such algorithm.

Our approach is based on a particular relaxation of the problem of modularity maximization. The main idea is to solve the relaxed problem with an iterative algorithm called *static algorithm* and then, recover a solution of the unrelaxed problem using a clustering algorithm.

Before the algorithmic part, some theoretical and numerical results are presented in chapter 2 to understand the behavior of the solutions of the relaxed problem. The conclusions obtained in this chapter are useful to set a reasonable range of values of the embedding dimension k in the static algorithms.

In chapter 3, four static algorithms are presented. The first one is the node by node algorithm, which is a random block-coordinate ascent method for which each block corresponds to a node of the graph. The second algorithm is called the preconditioned node by node algorithm and corresponds to preconditioning the problem before calling the node by node algorithm. The third one, the total update algorithm, is inspired on the node by node algorithm, the difference is that all blocks, i.e. all nodes, are updated simultaneously. The last static algorithm presented in this thesis is the Riemannian trust-region algorithm from [Absil et al., 2008] that adapts the idea of trust-region methods to the case of Riemannian manifolds. All the static algorithms except the last one are personal contributions.

Chapter 4 presents the discrete dynamic algorithm. It details how to efficiently implement a dynamic algorithm to track communities in time-varying networks. This heuristic process that uses a relaxation of the problem is a contribution of this work.

Finally, in chapter 5, the algorithms are compared in both static and time-varying cases. It was shown that most of the time, the total update algorithm gives the best results in comparison to the other algorithms developed in this thesis. In addition, this algorithm is robust with respect to the embedding dimension k , meaning that a large range of values of this parameter gives good results. This property makes its use easier.

Some numerical experiments indicate that this algorithm can be significantly better

than the Louvain algorithm, i.e. find better solutions in a shorter time. The cases where our algorithm is faster than Louvain are (i) when the graph is sufficiently sparse or (ii) when the modifications between two successive time steps are not too big. A very sparse example in the time-varying case is exhibited for which, when 10% of the data is modified, our algorithm gives a solution of quality 21% better than Louvain, 125 times faster.

It was shown that even in the static case, our algorithm can outperform Louvain in both modularity and running time on sparse graphs. Quantitatively, some examples are exhibited where our algorithm finds a solution of quality 24% higher than Louvain 16 times faster. Such results were unexpected, indeed, as mentioned in the title of this thesis, the algorithms developed here were designed to be competitive in the case of time-varying networks only.

These results indicate that the main objective of this thesis is achieved, at least partially.

The algorithms presented here are well suited for tasks like tracking groups of friends in a large, ever-changing social network, studying how the preferences of buyers evolve on an e-commerce website in real time, following objects on a high-definition video, etc.

As the amount of data available and the speed at which it evolves will dramatically increase in the coming years, we can only imagine that a fast algorithm for community detection in large-scale time-varying networks has a bright future ahead of it.

Bibliography

- Absil, P.-A., Baker, C. G., and Gallivan, K. A. (2007). Trust-region methods on Riemannian manifolds. *Found. Comput. Math.*, 7(3):303–330.
- Absil, P.-A., Mahony, R., and Sepulchre, R. (2008). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ.
- Adamic, L. A. and Glance, N. (2005). The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM.
- Aktunc, R., Toroslu, I. H., Ozer, M., and Davulcu, H. (2015). A dynamic modularity based community detection algorithm for large-scale networks: Dslm. In *Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on*, pages 1177–1183. IEEE.
- Alexanderson, G. (2006). About the cover: Euler and königsberg’s bridges: A historical view. *Bulletin of the american mathematical society*, 43(4):567–573.
- Aloise, D., Deshpande, A., Hansen, P., and Popat, P. (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248.
- Bandeira, A. S., Boumal, N., and Voroninski, V. (2008). On the low-rank approach for semidefinite programs arising in synchronization and community detection. *JMLR: Workshop and Conference Proceedings*, 49:1–22.
- Barvinok, A. (2001). A remark on the rank of positive semidefinite matrices subject to affine constraints. *Discrete & Computational Geometry*, 25(1):23–31.
- Barvinok, A. I. (1995). Problems of distance geometry and convex properties of quadratic maps. *Discrete & Computational Geometry*, 13(2):189–202.
- Bassett, D. S., Porter, M. A., Wymbs, N. F., Grafton, S. T., Carlson, J. M., and Mucha, P. J. (2012). Robust detection of dynamic community structure in networks. *CoRR*, abs/1206.4358.
- Bassett, D. S., Wymbs, N. F., Porter, M. A., Mucha, P. J., Carlson, J. M., and Grafton, S. T. (2011). Dynamic reconfiguration of human brain networks during learning. *Proceedings of the National Academy of Sciences*, 108(18):7641–7646.

- Blondel, V., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, (10) P10008.
- Boumal, N., Mishra, B., Absil, P.-A., and Sepulchre, R. (2014). Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459.
- Boumal, N., Voroninski, V., and Bandeira, A. (2016). The non-convex burer-monteiro approach works on smooth semidefinite programs. In *Advances in Neural Information Processing Systems*, pages 2757–2765.
- Brandes, U., Delling, D., Gaertler, M., Goerke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2006). Maximizing modularity is hard. cite arxiv:physics/0608255.
- Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2007). On finding graph clusterings with maximum modularity. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 121–132. Springer.
- Burer, S. and Monteiro, R. D. (2003). A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357.
- Burer, S. and Monteiro, R. D. (2005). Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103(3):427–444.
- Burer, S., Monteiro, R. D., and Zhang, Y. (2002). Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization*, 12(2):503–521.
- Caimo, A. and Mira, A. (2015). Efficient computational strategies for doubly intractable problems with applications to bayesian social networks. *Statistics and Computing*, 25(1):113–125.
- Clauset, A., Newman, M. E., and Moore, C. (2004). Finding community structure in very large networks. *Physical review E*, 70(6):066111.
- Dreyfus, S. E. (1969). An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412.
- Fanuel, M. (2017). Graph embedding, relaxed modularity optimization and density matrices. *In preparation*.
- Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486 (3-5):75–174.
- Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *PNAS*, 99(12):7821–7826.

- Godsil, C. D. (1997). Compact graphs and equitable partitions. *Linear Algebra and its Applications*, 255(1-3):259–266.
- Good, B., Montjoye, Y. D., and Clauset, A. (2010). Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4):046106.
- Hager, W. W. (2001). Minimizing a quadratic over a sphere. *SIAM Journal on Optimization (SIOPT)*, (12) 1:188–208.
- Izunaga, Y., Matsui, T., and Yamamoto, Y. (2016). A doubly nonnegative relaxation for modularity density maximization.
- Kocvara, M., Nesterov, Y., and Xia, Y. (2016). A subgradient method for free material design. *SIAM Journal on Optimization (SIOPT)*, (26) 4:2314–2354.
- Kyng, R. and Sachdeva, S. (2016). Approximate gaussian elimination for laplacians: Fast, sparse, and simple. *CoRR*, abs/1605.02353.
- Lambiotte, R., Delvenne, J.-C., and Barahona, M. (2008). Laplacian dynamics and multiscale modular structure in networks. *arXiv preprint arXiv:0812.1770*.
- Lancichinetti, A., Fortunato, S., and Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78(4):046110.
- Leskovec, J., Lang, K. J., and Mahoney, M. (2010). Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, pages 631–640. ACM.
- Lu, Z. and Xiao, L. (2015). On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, (152) 1:615–642.
- MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.
- Meunier, D., Lambiotte, R., Fornito, A., Ersche, K., and Bullmore, E. (2016). Hierarchical modularity in human brain functional networks. *Frontiers in Neuroinformatics*, (37) 3.
- Montanari, A. and Sen, S. (2015). Semidefinite programs on sparse random graphs and their application to community detection. *arXiv preprint arXiv:1504.05910*.
- Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38.
- Nesterov, Y. (2010). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization (SIOPT)*, (22) 2:341–362.
- Nesterov, Y. (2014). Subgradient methods for huge-scale optimization problems. *Mathematical Programming*, (146) 1-2:275–297.

- Nesterov, Y. and Shpirko, S. (2014). Primal-dual subgradient method for huge-scale linear conic problems. *SIAM Journal on Optimization (SIOPT)*, (24) 3:1444–1457.
- Nesterov, Y. and Stich, S. (2017a). Efficiency of accelerated coordinate descent method on structured optimization problems. *SIAM Journal on Optimization (SIOPT)*, (27) 1:110–123.
- Nesterov, Y. and Stich, S. U. (2017b). Efficiency of the accelerated coordinate descent method on structured optimization problems. *SIAM Journal on Optimization (SIOPT)*, (27) 1:110–123.
- Newman, M. E. (2006). Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582.
- Newman, M. E. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113.
- Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.
- Pataki, G. (1998). On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of operations research*, 23(2):339–358.
- Qin, T. and Rohe, K. (2013). Regularized spectral clustering under the degree-corrected stochastic blockmodel. *CoRR*, abs/1309.4111.
- Reddy, P. K., Kitsuregawa, M., Sreekanth, P., and Rao, S. S. (2002). A graph based approach to extract a neighborhood customer community for collaborative filtering. In *International Workshop on Databases in Networked Information Systems*, pages 188–200. Springer.
- Richtárik, P. and Takáč, M. (2013). On Optimal Probabilities in Stochastic Coordinate Descent Methods. *ArXiv e-prints*.
- Rosas-Casals, M., Valverde, S., and Solé, R. V. (2007). Topological vulnerability of the european power grid under errors and attacks. *International Journal of Bifurcation and Chaos*, 17(07):2465–2475.
- Schaub, M. T., Delvenne, J.-C., Yaliraki, S. N., and Barahona, M. (2012). Markov dynamics as a zooming lens for multiscale community detection: non clique-like communities and the field-of-view limit. *PloS one*, 7(2):e32210.
- Schobbens, E. (2014). Continuous relaxations for community detection in networks. Master’s thesis, Université Catholique de Louvain (École Polytechnique de Louvain).
- Sharma, A. and Pandey, R. (2011). Graph theory applied in school schedule network. *Int. J. Contemp. Math. Sciences*, 6(46):2247–2260.

- Steinbach, M., Karypis, G., Kumar, V., et al. (2000). A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston.
- Tappenden, R., Takáč, M., and Richtárik, P. (2015). On the Complexity of Parallel Coordinate Descent. *ArXiv e-prints*.
- Van Den Eeckhaut, K. (2016). Dynamic community detection in networks. Master’s thesis, Université Catholique de Louvain (École Polytechnique de Louvain).
- Van Dooren, P. and Absil, P.-A. (2014-2015). Linma1170 - analyse numérique.
- Vanderlinden, Q. (2015). Community detection in complex networks. Master’s thesis, Université Catholique de Louvain (École Polytechnique de Louvain).
- von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, (17) 4:395–416.
- Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of anthropological research*, (33) 4:452–473.

Appendix A

Theoretical results and proofs

This appendix presents theoretical results and proofs that were moved from the main part of the thesis.

A.1 Proofs

A.1.1 Proof of result 2

Proof. Rotations and reflections are exactly the linear operations obtained by product with an orthogonal matrix, i.e. a square matrix $\mathbf{R}$ such that $\mathbf{R}^T \mathbf{R} = \mathbf{I}$.

Let $\mathbf{H} \in \mathbb{R}^{n \times k}$ be feasible for (P) and let $\mathbf{R} \in \mathbb{R}^{k \times k}$ be an orthogonal matrix.

The transformed matrix will be written $\tilde{\mathbf{H}}$. The transformation of each row gives $(\tilde{\mathbf{H}}_{i,:})^T := \mathbf{R}(\mathbf{H}_{i,:})^T$ or equivalently, $\tilde{\mathbf{H}}_{i,:} := \mathbf{H}_{i,:} \mathbf{R}^T$. In matrix notation, we write $\tilde{\mathbf{H}} = \mathbf{H} \mathbf{R}^T$.

The following relations will be used: for any $i, j \in V$,

$$\sum_{l=1}^k \tilde{\mathbf{H}}_{il} \tilde{\mathbf{H}}_{jl} = \tilde{\mathbf{H}}_{i,:} (\tilde{\mathbf{H}}_{j,:})^T = \mathbf{H}_{i,:} \underbrace{\mathbf{R}^T \mathbf{R}}_{\mathbf{I}} (\mathbf{H}_{j,:})^T = \mathbf{H}_{i,:} (\mathbf{H}_{j,:})^T = \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl}. \quad (\text{A.1})$$

- Firstly, we show that $\tilde{\mathbf{H}}$ is feasible for (P): for any $i \in V$,

$$\sum_{l=1}^k \tilde{\mathbf{H}}_{il}^2 = \sum_{l=1}^k \mathbf{H}_{il}^2 = 1$$

where formula (A.1) and the fact that $\mathbf{H}$ is feasible have been used.

- Secondly, we show that the objective value does not change:

$$Q_k(\tilde{\mathbf{H}}) = \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^k \tilde{\mathbf{H}}_{il} \tilde{\mathbf{H}}_{jl} = \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} = Q_k(\mathbf{H})$$

where formula (A.1) has been used.

We have proved that $\tilde{\mathbf{H}}$ is still feasible and that the modularity does not change and then, we are done. $\square$

A.1.2 Proof of lemma 2

Proof. Let $\mathbf{H}^* \in \mathbb{R}^{n \times k}$ be an optimal solution of problem (P) with respect to k , i.e. $Q_k(\mathbf{H}^*) = Q_k^*$. We define $\tilde{\mathbf{H}} \in \mathbb{R}^{n \times (k+1)}$ by $\tilde{\mathbf{H}} = \begin{pmatrix} \mathbf{H}^* & \mathbf{0}_{n \times 1} \end{pmatrix}$. By the definition of $\tilde{\mathbf{H}}$, one can write

$$\begin{aligned} Q_{k+1}(\tilde{\mathbf{H}}) &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^{k+1} \tilde{\mathbf{H}}_{il} \tilde{\mathbf{H}}_{jl} \\ &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{H}_{il}^* \mathbf{H}_{jl}^* \\ &= Q_k(\mathbf{H}^*). \end{aligned}$$

Note that $\tilde{\mathbf{H}}$ is a feasible solution of the problem (P) with respect to $k+1$ and then $Q_{k+1}^* \geq Q_{k+1}(\tilde{\mathbf{H}}) = Q_k(\mathbf{H}^*) = Q_k^*$. $\square$

A.1.3 Proof of result 4

Proof. This result is proved in two parts, firstly, it is shown that the bound holds and secondly, that it can be reached.

- Let $\mathbf{H}^* \in \mathbb{R}^{n \times k}$ be any optimal solution, i.e. $Q_k(\mathbf{H}^*) = Q_k^*$. To show $\text{rank}(\mathbf{H}^*) \geq \delta(\mathcal{G})$, define $r := \text{rank}(\mathbf{H}^*)$ and proceed by contradiction. Assume $r < \delta(\mathcal{G})$.

The proof works fully similarly as the proof of lemma 3. Define $\mathcal{L}$, $\mathcal{R}(\cdot)$ and $\mathbf{H}_{small} \in \mathbb{R}^{n \times r}$ as in the proof of lemma 3. It holds that

$$Q_r(\mathbf{H}_{small}) = Q_k^*.$$

In addition, it follows from the assumption $k \geq \delta(\mathcal{G})$ and lemma 2 that $Q_k^* \geq Q_{\delta(\mathcal{G})}^*$. Using the definition of $\delta(\mathcal{G})$ yields to

$$Q_k^* = Q_{\delta(\mathcal{G})}^*.$$

Putting those equations together, one has $Q_r^* \geq Q_r(\mathbf{H}_{small}) = Q_k^* = Q_{\delta(\mathcal{G})}^*$, it is a contradiction with the assumption $r < \delta(\mathcal{G})$.

- It remains to show that this bound is reached for some feasible $\mathbf{H}^* \in \mathbb{R}^{n \times k}$.

By definition of $\delta(\mathcal{G})$ and thanks to the assumption $k \geq \delta(\mathcal{G})$, it exists an optimal solution $\mathbf{H}_{small}^* \in \mathbb{R}^{n \times \delta(\mathcal{G})}$ such that $Q_k^* = Q_{\delta(\mathcal{G})}(\mathbf{H}_{small}^*) = Q_{\delta(\mathcal{G})}^*$. Construct $\mathbf{H}^* := \begin{pmatrix} \mathbf{H}_{small}^* & \mathbf{0}_{n, (k-\delta(\mathcal{G}))} \end{pmatrix} \in \mathbb{R}^{n \times k}$. It holds that

$$\text{rank}(\mathbf{H}^*) = \text{rank}(\mathbf{H}_{small}^*).$$

Observe that $\mathbf{H}_{small}^* \in \mathbb{R}^{n \times \delta(\mathcal{G})}$ is feasible for (P) and then, by the first part of this result (already proven), it holds that $\text{rank}(\mathbf{H}_{small}^*) \geq \delta(\mathcal{G})$.

Otherwise, the matrix $\mathbf{H}_{small}^*$ has $\delta(\mathcal{G})$ columns and then $\text{rank}(\mathbf{H}_{small}^*) \leq \delta(\mathcal{G})$. One can conclude that

$$\text{rank}(\mathbf{H}_{small}^*) = \delta(\mathcal{G}).$$

It follows that $\text{rank}(\mathbf{H}^*) = \text{rank}(\mathbf{H}_{small}^*) = \delta(\mathcal{G})$ and the proof is complete. $\square$

A.1.4 Proof of lemma 4

Proof. The adjacency matrix of the complete graph K_n is $\mathbf{A}_{ij} = 1 - \delta_{ij}$, the number of edges is $m = \frac{1}{2}n(n-1)$ and the degrees are all $\mathbf{d}_i = n-1$. This allows writing

$$\begin{aligned} \mathbf{B}_{ij} &= \frac{1}{2m} \left(\mathbf{A}_{ij} - \frac{\mathbf{d}_i \mathbf{d}_j}{2m} \right) \\ &= \frac{1}{n(n-1)} \left(1 - \delta_{ij} - \frac{(n-1)^2}{n(n-1)} \right) \\ &= \frac{1}{n(n-1)} \left(\frac{1}{n} - \delta_{ij} \right). \end{aligned}$$

Otherwise, for any feasible solution $\mathbf{H} \in \mathbb{R}^{n \times k}$ of (P) and for any $1 \leq i, j \leq n$, it holds that $\sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} = \langle \mathbf{H}_{i,:}, \mathbf{H}_{j,:} \rangle \leq \|\mathbf{H}_{i,:}\| \cdot \|\mathbf{H}_{j,:}\| = 1$ where the Cauchy-Schwarz inequality and the constraint of (P) have been used.

Let $\mathbf{H} \in \mathbb{R}^{n \times k}$ be a feasible solution of (P). One can compute

$$\begin{aligned} Q_k(\mathbf{H}) &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^n \mathbf{H}_{il} \mathbf{H}_{jl} \\ &= \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=1}^n \left(\frac{1}{n} - \delta_{ij} \right) \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} \\ &= \frac{1}{n(n-1)} \left(\sum_{i=1}^n \sum_{j=1, j \neq i}^n \underbrace{\frac{1}{n} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl}}_{\leq 1} + \left(\frac{1}{n} - 1 \right) \sum_{i=1}^n \underbrace{\sum_{l=1}^k \mathbf{H}_{il}^2}_{=1} \right) \\ &\leq \frac{1}{n(n-1)} \left(\frac{1}{n} n(n-1) + \frac{1-n}{n} n \right) \\ &= 0. \end{aligned}$$

This is true for any feasible solution thus it is true for an optimal solution and then $Q_k^* \leq 0$.

Thanks to lemma 5 (see page 84), one can conclude that $Q_k^* = 0$.

As seen on the proof of lemma 5, this optimum is reached by $\mathbf{H} = \begin{pmatrix} \mathbf{1}_{n \times 1} & \mathbf{0}_{n \times (k-1)} \end{pmatrix}$ for any $k \in \mathbb{N}_0$. Taking the minimal value of k , i.e. $k = 1$, yields $\delta(K_n) = 1$ in the light of the definition of δ , i.e. (2.3). $\square$

A.1.5 Proof of result 6

Proof. This result is a particular setting of the optimization programs studied in [Pataki, 1998] and [Barvinok, 1995]. The only assumption to show is that the set of matrices defined by $\mathbf{X} := \mathbf{H}\mathbf{H}^T \in \mathbb{R}^{n \times n}$ is bounded. One can use the fact that all norms are equivalent in finite dimensional vector spaces and prove it in the Frobenius norm. It is sufficient to show that each entry is bounded

$$|\mathbf{X}_{ij}| = |(\mathbf{H}\mathbf{H}^T)_{ij}| = | \langle \mathbf{H}_{i,:}, \mathbf{H}_{j,:} \rangle | \leq \|\mathbf{H}_{i,:}\| \|\mathbf{H}_{j,:}\| = 1$$

where the inequality comes from the Cauchy-Schwarz inequality and the final equality comes from the constraints of (P).

It shows that the set of such matrices $\mathbf{X}$ is bounded and the proof is complete. $\square$

A.1.6 Proof of result 10

Proof. Let $\mathcal{G}$ be a graph with an isolated node with number n , i.e. the isolated node is the last one in the adjacency matrix. Denote by $\tilde{\mathcal{G}}$ the graph $\mathcal{G}$ with this isolated node removed. The notation $\tilde{\square}$ is used to refer to characteristic $\square$ of $\tilde{\mathcal{G}}$.

One can express the quantities of $\mathcal{G}$ in function of the quantities of $\tilde{\mathcal{G}}$:

$$\mathbf{A}_{ij} = \begin{cases} \tilde{\mathbf{A}}_{ij} & \text{if } i < n \text{ and } j < n \\ 0 & \text{if } i = n \text{ or } j = n \end{cases}$$

and

$$\mathbf{d}_i = \begin{cases} \tilde{\mathbf{d}}_i & \text{if } i < n \\ 0 & \text{if } i = n \end{cases}$$

and finally, $m = \tilde{m}$.

By the definition of the modularity matrix and the previous considerations, we have

$$\mathbf{B}_{ij} = \begin{cases} \tilde{\mathbf{B}}_{ij} & \text{if } i < n \text{ and } j < n \\ 0 & \text{if } i = n \text{ or } j = n. \end{cases}$$

Let $\mathbf{H} \in \mathbb{R}^{n \times k}$ be a feasible solution of the relaxed problem for the graph $\mathcal{G}$. We construct $\tilde{\mathbf{H}} \in \mathbb{R}^{(n-1) \times k}$ such that $\mathbf{H}$ restricted to its $n - 1$ first rows is equal to $\tilde{\mathbf{H}}$.

Using the definition of Q_k gives

$$\begin{aligned}
Q_k(\mathbf{H}) &= \sum_{i=1}^{n-1} \sum_{j=1}^n \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} + \sum_{j=1}^n \underbrace{\mathbf{B}_{n,j}}_{=0} \sum_{l=1}^k \mathbf{H}_{n,l} \mathbf{H}_{jl} \\
&= \sum_{i=1}^{n-1} \left(\sum_{j=1}^{n-1} \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} + \underbrace{\mathbf{B}_{i,n}}_{=0} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{n,l} \right) \\
&= \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} \mathbf{B}_{ij} \sum_{l=1}^k \mathbf{H}_{il} \mathbf{H}_{jl} \\
&= \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} \tilde{\mathbf{B}}_{ij} \sum_{l=1}^k \tilde{\mathbf{H}}_{il} \tilde{\mathbf{H}}_{jl} \\
&= Q_k(\tilde{\mathbf{H}}).
\end{aligned}$$

It follows that $\tilde{Q}_k^* = Q_k^*$. □

A.2 Example of degenerated solution

This section focuses on the construction of an example of degenerated solution, i.e. an optimal solution $\mathbf{H}^*$ of problem (P) for a graph $\mathcal{G}$ such that $\delta(\mathcal{G}) < \text{rank}(\mathbf{H}^*)$.

A result that comes from the Vanderlinden's thesis [Vanderlinden, 2015, Proposition 2.1] is needed. It states that the objective function $Q_k(\mathbf{H})$ is a quadratic form. More precisely, if one defines

$$\mathbf{x} := \begin{pmatrix} \mathbf{H}_{:,1} \\ \vdots \\ \mathbf{H}_{:,k} \end{pmatrix}$$

and

$$\mathbf{M} := \begin{pmatrix} \mathbf{B} & & \\ & \ddots & \\ & & \mathbf{B} \end{pmatrix} \in \mathbb{R}^{n^2 \times n^2}, \quad (\text{A.2})$$

with n blocks, it is

$$Q_k(\mathbf{H}) = \mathbf{x}^T \mathbf{M} \mathbf{x}. \quad (\text{A.3})$$

The advantage of this formulation is that information can be extracted from the eigenvalues of $\mathbf{M}$.

This quadratic formulation reveals that

$$Q_k(\mathbf{H}) \leq 0 \text{ for any } \mathbf{H} \Leftrightarrow \mathbf{M} \preceq 0.$$

Otherwise, from the block-diagonal definition of $\mathbf{M}$, i.e. (A.2), it holds that

$$\mathbf{M} \preceq 0 \Leftrightarrow \mathbf{B} \preceq 0.$$

These relations give

$$Q_k(\mathbf{H}) \leq 0 \text{ for any } \mathbf{H} \Leftrightarrow \mathbf{B} \preceq 0. \quad (\text{A.4})$$

The last result that is needed to construct the example of degenerate solution is that the optimum value is always positive, it is lemma 5 that will be useful more than once.

Lemma 5. *For any graph and for any k , $Q_k^* \geq 0$.*

Proof. Define $\mathbf{H} \in \mathbb{R}^{n \times k}$ by $\mathbf{H}_{il} = \delta_{il}$. One can compute

$$\begin{aligned} Q_k(\mathbf{H}) &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \underbrace{\sum_{l=1}^k \delta_{il} \delta_{jl}}_{=1} \\ &= \sum_{i=1}^n \sum_{j=1}^n \mathbf{B}_{ij} \\ &= \frac{1}{2m} \left(\underbrace{\sum_{i=1}^n \sum_{j=1}^n \mathbf{A}_{ij}}_{=2m} - \frac{1}{2m} \underbrace{\sum_{i=1}^n \mathbf{d}_i}_{=2m} \underbrace{\sum_{j=1}^n \mathbf{d}_j}_{=2m} \right) \\ &= \frac{1}{2m} \left(2m - \frac{(2m)^2}{2m} \right) \\ &= 0. \end{aligned}$$

But $\mathbf{H}$ is a feasible solution of (P) and then $Q_k^* \geq Q_k(\mathbf{H}) = 0$. $\square$

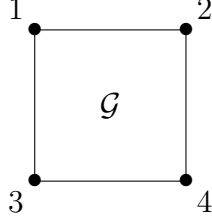
We have now all the tools to construct a graph $\mathcal{G}$ and an optimal solution $\mathbf{H}^*$ such that $\text{rank}(\mathbf{H}^*) > \delta(\mathcal{G})$.

Consider the square graph $\mathcal{G}$ depicted on figure A.1. Its adjacency matrix is

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad (\text{A.5})$$

and its modularity matrix is

$$\mathbf{B} = \frac{1}{16} \begin{pmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{pmatrix}.$$

Figure A.1: Representation of the graph $\mathcal{G}$.

One can see that the eigenvalues of $\mathbf{B}$ are 0, 0, 0 and $-\frac{1}{4}$ and then $\mathbf{B} \preceq 0$ from which relation (A.4) gives that for any value of $k \in \mathbb{N}_0$, it holds that $Q_k(\mathbf{H}) \leq 0$ for any feasible $\mathbf{H}$. Taking the maximum over all feasible $\mathbf{H}$ gives $Q_k^* \leq 0$ for any k . But lemma 5 indicates that $Q_k^* \geq 0$. One can conclude that for this graph $\mathcal{G}$ and for any $k \in \mathbb{N}_0$,

$$Q_k^*(\mathcal{G}) = 0.$$

In addition, the proof of lemma 5 indicates that this optimum is reached for any $k \in \mathbb{N}_0$ by

$$\mathbf{H} = \begin{pmatrix} \mathbf{1}_{n,1} & \mathbf{0}_{n,k-1} \end{pmatrix}.$$

Taking the minimum over $k \in \mathbb{N}_0$ gives $\delta(\mathcal{G}) = 1$.

It remains to show that there exists some optimal solutions $\mathbf{H}^* \in \mathbb{R}^{n \times k}$, i.e. such that $Q_k(\mathbf{H}^*) = 0$, for which $\text{rank}(\mathbf{H}^*) > 1$.

Set $k = 2 \geq \delta(\mathcal{G}) = 1$ and

$$\mathbf{H}^* = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \quad (\text{A.6})$$

Observe that $\mathbf{H}^*$ is feasible for (P). In addition, one can verify that $\mathbf{B}\mathbf{H}^* = \mathbf{0}_{n,k}$ and then $Q_k(\mathbf{H}^*) = \text{Tr}((\mathbf{H}^*)^T \mathbf{B} \mathbf{H}^*) = 0 = Q_\delta(\mathcal{G})^*$. That means that all assumptions of result 4 are fulfilled but $\text{rank}(\mathbf{H}^*) = 2 > 1 = \delta(\mathcal{G})$.

One can conclude that $\mathbf{H}^*$ defined in (A.6) is a degenerated solution of problem (P) for the graph with adjacency matrix given by (A.5).

A.3 Convergence results of the Riemannian trust-region algorithm

The results invoke the gradient operator and Hessian operator that are denoted grad and Hess respectively. They refer to the Riemannian gradient and the Riemannian Hessian respectively. We do not define those notions formally but they can be considered as an equivalent of the classical Euclidean ones but on a Riemannian manifold.

The first result of convergence is a global one, it states that the sequence generated by the RTR algorithm converges to a critical point of Q_k .

Result 11. *Let $\{\mathbf{H}^j\}$ be a sequence of iterates generated by the RTR algorithm. Then*

$$\lim_{j \rightarrow \infty} \text{grad } Q_k(\mathbf{H}^j) = \mathbf{0}.$$

It is shown in [Absil et al., 2008, Theorem 7.4.4, Proposition 7.4.5, Corollary 7.4.6] that if the objective function (in our case Q_k) is smooth and the riemannian manifold (in our case $(\mathbb{S}^{k-1})^n$) is compact then the result follows. The first is obvious when we consider the summation development of $Q_k(\mathbf{H})$. The compactness of the manifold follows from the fact that it is closed and bounded. Indeed, the Borel-Lebesgue theorem states that in $\mathbb{R}^{n \times k}$ (among others), the compact sets are exactly the closed bounded sets.

The next result of convergence is local and states that some local maxima of Q_k are attractive for the RTR algorithm.

Result 12. *Let $\mathbf{H}^*$ be a local maximum of Q_k such that the Riemannian gradient $\text{grad } Q_k(\mathbf{H}^*) = 0$ and the Riemannian Hessian $\text{Hess } f(\mathbf{H}^*)$ is negative-definite. Then there exists a neighborhood $\mathcal{V}$ of $\mathbf{H}^*$ such that, for all $\mathbf{H}^0 \in \mathcal{V}$, the sequence $\{\mathbf{H}^j\}$ generated by the RTR algorithm converges to $\mathbf{H}^*$.*

It comes from [Absil et al., 2008, Theorem 7.4.10]. The original theorem requires some additional assumptions. One is that $\mathbf{H} \mapsto \|\text{Hess}^{-1}Q_k(\mathbf{H})\|$ is bounded on a neighborhood of $\mathbf{H}^*$. It is fulfilled because the objective function is quadratic. All the other assumptions are weaker than the smoothness of Q_k and the compactness of the domain that are both satisfied (it comes from [Absil et al., 2008, Theorem 7.4.2, Theorem 7.4.4, Corollary 7.4.6, Equation (7.25)]).

The following last result states that locally, the RTR algorithm converges quadratically to a maximum if it has some properties.

Result 13. *Let $\mathbf{H}^*$ be a local maximum of Q_k such that $\text{grad } Q_k(\mathbf{H}^*) = 0$ and $\text{Hess } f(\mathbf{H}^*)$ is negative-definite. Then there exists $c > 0$ such that, for all sequences $\{\mathbf{H}^j\}$ generated by the RTR algorithm converging to $\mathbf{H}^*$, there exists $J > 0$ such that for all $j > J$,*

$$\text{dist}(\mathbf{H}^{k+1}, \mathbf{H}^*) \leq c(\text{dist}(\mathbf{H}^k, \mathbf{H}^*))^2.$$

This result follows from [Absil et al., 2008, Theorem 7.4.11] using the smoothness of Q_k .

Finally, in [Absil et al., 2008, Section 7.4.3], it is observed that the RTR algorithm is an ascent method and then, saddle points or local minima are unstable fixed points of the algorithm.

Appendix B

Pseudocode

This appendix contains pseudocode versions of the algorithms presented in this thesis. In addition, an analysis of both time and space complexities are presented.

B.1 Node by node algorithm

A pseudocode version of the node by node algorithm is presented in algorithm 1.

Lines 3 to 8 construct a set I such that for all $N \in I$, $\mathbf{B}_{N\bar{N}} \neq \mathbf{0}$ as in result 8. Note that in some programming languages, it can be faster to replace those 6 lines by a vectorial operation. For example, in MATLAB, it is preferred to write $I := \text{find}(\mathbf{d} \neq 0)$.

B.1.1 Time complexity

This section focuses on time complexity of the NBN algorithm. The number of iterations in the *while loop* is denoted by $\bar{K}$. As previously, $\bar{m}$ refers to the number of non-zero elements in $\mathbf{A}$.

Lines 1 and 2 are both in $O(n)$ operations. The operations between lines 3 and 8 are $O(n)$, the computations inside the *for loop* are $O(1)$.

The computation of $\mathbf{c}^N$ in line 14 is $O(k\bar{m})$ as explained after equation (3.3). Line 15 requires $O(k)$ operations, indeed, it computes the square of each element ($O(k)$), it sums the results ($O(k-1)$) and it computes the square root to have the norm ($O(1)$). After that, it divides $\mathbf{c}^N$ by the norm ($O(k)$). Summing these results yields that line 15 is $O(k + k + k) = O(k)$. Finally, those two lines 14 and 15 are in a *for loop* that goes through n elements itself in a *while loop* that do $\bar{K}$ iterations. Then the *while loop* is $O(k\bar{m} + k)n\bar{K}) = O(\bar{K}nk\bar{m})$. Note that compute $\|\mathbf{H} - \mathbf{H}_{prev}\|_F$ is $O(kn)$.

Finally, the whole NBN algorithm has a complexity as $O(n + n + \bar{K}nk\bar{m}) = O(\bar{K}nk\bar{m})$. Note that if we define $K = \bar{K}n$, the number of times that a node is updated, the complexity is $O(Kk\bar{m})$, which looks reasonable.

Algorithm 1 Node by node algorithm (NBN)

Input:

$\mathbf{A}$ an adjacency matrix
 $\mathbf{H}$ feasible
 M a maximal number of iterations
 $\epsilon > 0$ a tolerance

Output:

$\mathbf{H}$ a feasible better solution

```

1:  $\mathbf{d} := \sum_{j \in V} \mathbf{A}_{:,j}$ 
2:  $m := \frac{1}{2} \sum_{j \in V} \mathbf{d}_j$ 
3:  $I := \emptyset$ 
4: for  $i \in V$  do
5:   if  $\mathbf{d}_i \neq 0$  then
6:      $I := I \cup \{i\}$ 
7:   end if
8: end for
9:  $i := 1$ 
10:  $\mathbf{H}_{prev} := \mathbf{H} + 2\epsilon$ 
11: while  $i \leq M$  and  $\|\mathbf{H} - \mathbf{H}_{prev}\|_F > \epsilon$  do
12:    $\mathbf{H}_{prev} := \mathbf{H}$ 
13:   for  $N \in$  random permutation of  $I$  do
14:     compute  $\mathbf{c}^N$  from  $\mathbf{H}$  using formula (3.3)
15:      $\mathbf{H}_{N,:} := \mathbf{c}^N / \|\mathbf{c}^N\|$ 
16:   end for
17:    $i := i + 1$ 
18: end while

```

If one chooses the stopping criterion on the objective value, the complexity of the algorithm does not change because formula (3.4) is $O(k\bar{m})$.

B.1.2 Space complexity

To compute the space complexity, we look at the size of data that is stored during the execution of the algorithm. It is presented in table B.1.

Variable	Size
m, i, N	$\Theta(1)$
$\mathbf{d}, I$	$\Theta(n)$
$\mathbf{H}, \mathbf{H}_{prev}$	$\Theta(kn)$
$\mathbf{A}$	$\Theta(\bar{m})$
$\mathbf{c}$	$\Theta(k)$

Table B.1: Size of the data stored in the NBN Algorithm.

It is concluded that the NBN algorithm has a space complexity in $\Theta(\max\{kn, \bar{m}\})$.

B.2 Total update algorithm

A pseudocode version of the total update algorithm is presented in algorithm 2.

Lines 1 to 13 are the same than for the NBN Algorithm except for lines 4 and 8. The last one sets the diagonal elements of $\tilde{\mathbf{A}}_{ii}$. Observe that this operation is done only when $\mathbf{d}_i \neq 0$ because otherwise, $\tilde{\mathbf{A}}_{ii} = \mathbf{A}_{ii}$ is still 0.

As explained for the NBN algorithm, lines 3 to 10 can be replaced by a vectorial operation. In MATLAB we prefer to use $I := \text{find}(\mathbf{d} \neq 0)$ for 7 and $\tilde{\mathbf{A}}(1 : n + 1 : \text{end}) = \mathbf{d} . * \mathbf{d}$ for 8 where $1 : n + 1 : \text{end}$ refers to the diagonal elements and $\mathbf{d} . * \mathbf{d}$ is a component wise product.

Note that $\mathbf{A}$ can be removed from memory after line 4. It can be useful from a storage point of view. Alternately, one can replace all $\tilde{\mathbf{A}}$ in the code by $\mathbf{A}$ to overwrite memory.

B.2.1 Time complexity

We write K the number of iterations of the *while loop* in line 13. *A priori*, there is no link between this K and the K in the complexity analysis of the NBN algorithm. As previously, the number of non-zero elements in $\tilde{\mathbf{A}}$ is written $\bar{m}$.

Lines 1 to 10 are in $O(n)$, indeed lines 7 and 8 are both in $O(1)$. Line 15 is $O(k\bar{m})$ as explained after equation (3.8). The normalization in line 17 is in $O(k)$ as in the case

Algorithm 2 Total update algorithm (TU)

Input:

$\mathbf{A}$ an adjacency matrix
 $\mathbf{H}$ feasible
 M a maximal number of iterations
 $\epsilon > 0$ a tolerance

Output:

$\mathbf{H}$ a feasible better solution

```

1:  $\mathbf{d} := \sum_{j \in V} \mathbf{A}_{:,j}$ 
2:  $m := \frac{1}{2} \sum_{j \in V} \mathbf{d}_j$ 
3:  $I := \emptyset$ 
4:  $\tilde{\mathbf{A}} := \mathbf{A}$ 
5: for  $i \in V$  do
6:   if  $\mathbf{d}_i \neq 0$  then
7:      $I := I \cup \{i\}$ 
8:      $\tilde{\mathbf{A}}_{ii} := \frac{(\mathbf{d}_i)^2}{2m}$ 
9:   end if
10: end for
11:  $i := 1$ 
12:  $\mathbf{H}_{prev} := \mathbf{H} + 2\epsilon$ 
13: while  $i \leq M$  and  $\|\mathbf{H} - \mathbf{H}_{prev}\|_F > \epsilon$  do
14:    $\mathbf{H}_{prev} := \mathbf{H}$ 
15:   Compute  $\mathbf{C}$  from  $\mathbf{H}$  using formula (3.8)
16:   for  $N \in I$  do
17:      $\mathbf{H}_{N,:} := \mathbf{C}_{N,:} / \|\mathbf{C}_{N,:}\|$ 
18:   end for
19:    $i := i + 1$ 
20: end while

```

of the NBN algorithm. The *for loop* between lines 16 and 18 is then $O(kn)$. Finally, the *while loop* is in $O(K(k\bar{m} + kn)) = O(Kk\bar{m})$. Note that computing $\|\mathbf{H} - \mathbf{H}_{prev}\|_F$ is $O(kn)$.

Putting all this together, we see that the total update algorithm is $O(Kk\bar{m})$.

B.2.2 Space complexity

The sizes of the data used in the total update algorithm are presented in table B.2.

Variable	Size
m, i	$\Theta(1)$
$\mathbf{d}, I$	$\Theta(n)$
$\mathbf{H}, \mathbf{H}_{prev}, \mathbf{C}$	$\Theta(kn)$
$\mathbf{A}, \tilde{\mathbf{A}}$	$\Theta(\bar{m})$
$\mathbf{c}$	$\Theta(k)$

Table B.2: Size of the data stored in the total update algorithm.

One can conclude that the space complexity of the total update algorithm is in $\Theta(\max\{kn, \bar{m}\})$.

B.3 Fuzzy dynamic algorithm

At step $t \in \mathbb{N}$, the graph is denoted by $\mathcal{G}^t$ and all the quantities related to $\mathcal{G}^t$ are super-scripted by t to emphasis this time dependency.

A pseudocode version of the fuzzy dynamic algorithm is presented in algorithm 3.

The statement in line 3 is necessary in order to adapt the size of $\mathbf{H}$ in case some nodes are suppressed. The *for loop* in line 4 adjust the size of $\mathbf{H}$ in the case some nodes are added.

B.4 Discrete dynamic algorithm

We write $\mathbf{M} \in \mathbb{R}^{c \times k}$ a matrix of which each line is a mean: $\mathbf{M}_{i,:} := \mathbf{m}_i$.

To call the k -means algorithm, we write $[\mathbf{L}, \mathbf{M}^*] := kmeans(\mathbf{H}, c, -, r)$ where each line of $\mathbf{H}$ is a data point that will be associated to a cluster, i.e. $\mathbf{H}_{i,:} =: \mathbf{x}_i$. c is the number of clusters and r is the number of times that the algorithm is repeated to avoid sub-optimal local optimum. Concerning the output arguments, $\mathbf{L}$ is a matrix in $\{0, 1\}^{n \times c}$ such that $\mathbf{L}_{ij} = 1$ if data $\mathbf{x}_i$ is associated to cluster j and 0 otherwise.

Algorithm 3 Fuzzy dynamic algorithm (FDA)

Input:

$Adj(t) : \mathbb{N} \rightarrow \mathcal{M} : t \mapsto \mathbf{A}^t$: an oracle that returns the adjacency matrix of $\mathcal{G}^t$
 $added(t) : \mathbb{N} \rightarrow \mathbb{N} : t \mapsto added(t)$: an oracle that says how many nodes have been added from $\mathcal{G}^{t-1}$ to $\mathcal{G}^t$
 $deleted(t) : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N})$: an oracle that indicates which nodes have been deleted from $\mathcal{G}^{t-1}$ to $\mathcal{G}^t$
 $\mathbf{H}$ feasible for $\mathcal{G}^0$
 M a maximal number of iterations
 $\epsilon > 0$ a tolerance
 $StaticAlgorithm(\dots)$ an algorithm that solves (P) in the static case

Output:

No output but print the $\mathbf{H}$ computed at each time step

```

1: for  $t = 0, \dots$  do
2:    $\mathbf{A} := Adj(t)$ 
3:   Suppress the lines  $deleted(t)$  of  $\mathbf{H}$ 
4:   for  $i = 0, \dots, added(t) - 1$  do
5:     Let  $\mathbf{v} \in \mathbb{R}_0^{1 \times k}$  be a non zero random vector
6:      $\mathbf{H}_{n^t-i,:} := \mathbf{v} / \|\mathbf{v}\|$ 
7:   end for
8:    $\mathbf{H} := StaticAlgorithm(\mathbf{A}, \mathbf{H}, M, \epsilon)$ 
9:   print ' $\mathbf{H}$  is the solution at time  $t$ '
10: end for

```

Finally, $\mathbf{M}^* \in \mathbb{R}^{n \times c}$ is the matrix containing centroids of the outputted solution in its rows i.e. $\mathbf{M}_{i,:}^* =: \mathbf{m}_i$.

Similarly, we write $[\mathbf{L}, \mathbf{M}^*] := kmeans(\mathbf{H}, c, \mathbf{M})$ to call k -means but in this case the initial means $\mathbf{M}$ are given and then, the algorithm is ran only one time.

These notations allow writing the pseudocode of the discrete dynamic algorithm, it is algorithm 4.

First of all, the sparsity of the adjacency matrix $\mathbf{A}$ has to be used correctly. If a MATLAB implementation is used, the sparsity have to be indicated at the beginning and kept during the evolution of the graph. Indeed, convert in the memory a full matrix into a sparse matrix is in $O(n^2)$ and is not cheap and then the algorithm would be slowed down if at each step this conversion have to be done.

Algorithm 4 Discrete dynamic algorithm (DDA)

Input:

$Adj(t) : \mathbb{N} \rightarrow \mathcal{M} : t \mapsto \mathbf{A}^t$: an oracle that returns the adjacency matrix of $\mathcal{G}^t$ (without isolated nodes)

$added(t) : \mathbb{N} \rightarrow \mathbb{N} : t \mapsto added(t)$: an oracle that say how many nodes have been added from $\mathcal{G}^{t-1}$ to $\mathcal{G}^t$

$deleted(t) : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N})$: an oracle that indicates which nodes have been deleted from $\mathcal{G}^{t-1}$ to $\mathcal{G}^t$

$\mathbf{H}$ feasible for $\mathcal{G}^0$

M a maximal number of iterations

$\epsilon > 0$ a tolerance

$T \in \mathbb{N}_0$: the period between two updates of the number of clusters

$r \in \mathbb{N}_0$: the number of repetitions of k -means *StaticAlgorithm*(...): an algorithm that solves (P) in the static case

Output:

No output but print the $\mathbf{L}$ computed at each time step

```

1: for  $t = 0, \dots$  do
2:    $\mathbf{A} := Adj(t)$ 
3:   Suppress the lines  $deleted(t)$  of  $\mathbf{H}$ 
4:   for  $i = 0, \dots, added(t) - 1$  do
5:     Let  $\mathbf{v} \in \mathbb{R}_0^{1 \times k}$  be a non zero random vector
6:      $\mathbf{H}_{n^t-i,:} := \mathbf{v} / \|\mathbf{v}\|$ 
7:   end for
8:   if  $t \bmod T = 0$  then
9:      $[\mathbf{L}, c] := Louvain(\mathbf{A})$ 
10:  else
11:     $\mathbf{H} := StaticAlgorithm(\mathbf{A}, \mathbf{H}, M, \epsilon)$ 
12:    if  $t \bmod T = 1$  then
13:       $[\mathbf{L}, \mathbf{M}] := kmeans(\mathbf{H}, c, -, r)$ 
14:    else
15:       $[\mathbf{L}, \mathbf{M}] := kmeans(\mathbf{H}, c, \mathbf{M})$ 
16:    end if
17:  end if
18:  print 'L is the discrete solution at time  $t$ '
19: end for

```
