

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
UNIVERSITY MOHAMMED SEDDIK BENYAHIA

JIJEL

Faculty of Exact Sciences and Computer Science

Department of Computer Science



MASTER THESIS

Submitted to the Department of Computer Science in Partial
Fulfillment of the Requirements for the Degree of Master

Option: Computer System and Decision Support

Topic

**Optimized modularity for class detection in a graph:
Application Compare various community detection based on modularity
optimization.**

Presented by:

Mr. *Faris BOUKARIOUA*

Supervised by:

Mr. *Ali LEMOUARI*

Empty

To my dear parents.

Acknowledgments

Above all, I thank my Almighty God, who gave me strength, faith, health, will, and guidance to accomplish this modest work.

*I wish particularly to acknowledge my Supervisor MR. **Ali LEMOUARI** for having supervised, helped, guided, trust, and encouraged me throughout my work.*

Thanks to his instructions, the work has been adequately done.

I would like to extend my sincere thanks to all those who have contributed in one way or another to the realization of this thesis.

My deepest gratitude goes to my family, who have been able to approach me without letting go of the support during all these long years of study.

Contents

INTRODUCTION.....	1
CHAPTER 1: BASIC CONCEPTS	2
1 INTRODUCTION.....	2
2 Complex network	2
2.1 Complex network definition	2
2.2 Complex network domains	2
2.3 Complex network types	4
2.3.1 Strongly evolving network	4
2.3.2 Small-world network	4
3 Graph theory	4
3.1 Concepts and notations.....	5
3.2 Temporal graphs	7
3.2.1 Contact sequences:.....	7
4 Conclusion:.....	7
CHAPTER 2:.....	8
1 INTRODUCTION.....	8
2 Definition of a community:	8
3 Detection of communities:.....	8
4 Detection of dynamic communities:.....	9
4.1 Operations on dynamic communities:.....	9
5 Community detection Methods:.....	10
5.1 Approaches based on static algorithms:	10
5.1.1 Hierarchical approaches.....	11
5.1.2 Approaches based on the optimization of an objective function	12
5.1.3 Approaches based on cliques	13
5.2 Dynamic approaches:	14
5.2.1 Approaches based on network snapshots.....	14
5.2.2 Approaches working on temporal networks	17
6 State of the art synthesis	19
7 Modularity.....	20
8 Conclusion:.....	21
CHAPTER 3:	22

1	INTRODUCTION	22
2	Clauset-Newman-Moore (CNM) algorithm	22
3	Louvain Algorithm	23
3.1	Community Aggregation	24
4	Leiden Algorithm	26
5	Paris algorithm	29
6	The eigenvectors of matrices method for community detection	29
7	Conclusion	30
CHAPTER 4:		31
1	INTRODUCTION	31
2	Development Tools	31
2.1	Parrot OS:	31
2.2	Pycharm IDE	32
2.3	Python3	32
2.4	Anaconda	33
2.5	Jupyter Notebook	34
2.6	Projet requirements	35
3	Execution	36
4	Results	38
4.1	Karate_club	38
4.1.1	Louvain algorithm	38
4.1.2	CNM algorithm	39
4.1.3	Leiden Algorithm	39
4.1.4	Paris Algorithm	40
4.1.5	Eigenvector Algorithm	40
4.2	LFR benchmark	41
4.2.1	CNM	41
4.2.2	Leiden	42
4.2.3	Louvain	42
4.2.4	Paris	43
4.3	generate LFR using uniform distribution in parameters	44
4.3.1	CNM	45
4.3.2	Leiden	46

4.3.3 Louvain 47
4.3.4 Paris..... 48
4.3.5 Eigenvector..... 49
4.4 Results of comparison..... 50
5 Conclusion 52
References 53

List of Tables

Table 1. Comparison of community detection approaches	19
Table 2. Mean running time & mean NMI	52

List of Figures

Figure 1. Graphical representation of a theoretical social network.....	3
Figure 2. The two best studied information networks.....	3
Figure 3. Operations in dynamic communities.....	10
Figure 4. Agglomerative and Divisive Hierarchical clustering algorithms	11
Figure 5. An example of the clink percolation algorithm with $k = 3$	13
Figure 6. Representation of a method by successive static informed detections	16
Figure 7. Sequence of steps followed by Louvain algorithm.	25
Figure 8. Illustration of the Leiden algorithm.....	26
Figure 9. Parrot version	31
Figure 10. Project execution	37
Figure 11. Jupyter code source Run	37
Figure 12. Louvain algorithm (karate club)	38
Figure 13. CNM algorithm (karate club)	39
Figure 14. Leiden Algorithm (karate club)	39
Figure 15. Paris Algorithm (karate club)	40
Figure 16. Eigenvector Algorithm (karate club)	40
Figure 17. LFR benchmark for CNM	41
Figure 18. LFR benchmark for Leiden	42
Figure 19. LFR benchmark for Louvain	42
Figure 20. LFR benchmark for Louvain	43
Figure 21. LFR benchmark for Eigenvector	44
Figure 22. LFR benchmark for CNM using uniform distribution in parameters	45
Figure 23. LFR benchmark for Leiden using uniform distribution in parameters	46
Figure 24. LFR benchmark for Louvain using uniform distribution in parameters.....	47
Figure 25. LFR benchmark for Paris using uniform distribution in parameters	48
Figure 26. LFR benchmark for Eigenvector using uniform distribution in parameters	49
Figure 27. Mean NMI information.....	50
Figure 28. Mean compute time.....	51

Abstract

Complex systems from many disciplines can be modeled by networks, specifically by nodes connected by edges. These networks exhibit a microscopic structure called "community structure". A community is seen as a subgraph composed of densely linked nodes together and weakly linked to other network nodes. The detection of this community structure is crucial to understanding the topology and operation of these networks. The majority of work in the literature concerning the community detection relate to static networks. However, many networks evolve over time. The traditional approach to community detection reuses static algorithms on different snapshots of the network and suffer of instability problems. In this work, we compare various community detection algorithms based on modularity optimization. The comparison is achieved by applying various algorithms on different datasets (increasing the number of nodes), while respecting two main aspects: time and information.

Keywords: algorithm, community detection, graph clustering, community, modularity, optimization.

Résumé

Les systèmes complexes de nombreuses disciplines peuvent être modélisés par des réseaux, plus précisément par des graphes de nœuds reliés par des arêtes. Ces réseaux présentent une structure microscopique appelée "structure de communauté". Une communauté est vue comme un sous-graphe composé de nœuds densément liés entre eux et faiblement liés aux autres nœuds du réseau. La détection de cette structure communautaire est cruciale pour comprendre la topologie et le fonctionnement de ces réseaux. La majorité des travaux dans la littérature concernant la détection des communautés portent sur des réseaux statiques. Cependant, de nombreux réseaux évoluent dans le temps. L'approche traditionnelle de la détection de communauté réutilise des algorithmes statiques sur différents instantanés du réseau et souffre de problèmes d'instabilité. Dans ce travail, nous comparons différents algorithmes de détection de communautés basés sur l'optimisation de la modularité. La comparaison est réalisée en appliquant divers algorithmes sur différents ensembles de données (en augmentant le nombre de nœuds), tout en respectant deux aspects principaux : le temps et l'information.

Mots clés : algorithme, détection de communauté, regroupement de graphes, communauté, modularité, optimisation

INTRODUCTION

Complex networks are used in all fields. In computer science, the Internet can be seen as a set of routers interacting via cables. In biology, the brain is a set of neurons interacting with each other. In the field of sociology, the study of social networks leads to the study of how various agents interact with each other. In general, these networks can be defined as a set of connected entities. And their modeling is done by graphs called complex graphs.

The presence of complex networks in all domains is the essential reason that has led researchers to want to understand and analyze these networks. The axis that has received a lot of interest in network analysis is the detection of communities. A community is seen as a part of a graph composed of nodes that are strongly connected to each other and weakly connected to the rest of the nodes in the network. Community detection allows us to understand how the network works.

In this thesis, our work aims to compare the results of five algorithms on synthetic generated network using Lancichinetti-Fortunato-Radicchi (LFR) benchmarks used for the detection of communities. The results are obtained by the optimization of the gain function (modularity).

This thesis is composed of four chapters to which are added this introduction and a general conclusion.

In the first chapter, we define what a complex network is, its types and its application domains, then we define some concepts considered necessary in graph theory.

The second chapter presents firstly the definitions related to the notion of community. A review of the research related to the detection of dynamic communities will be presented thereafter.

In the third chapter, we present the five algorithms we will be using in the next chapter to retrieve the results of the community detection. Each algorithm is presented with a brief description and a pseudo-code.

In the fourth chapter, we first present the necessary software that we used to obtain our results. Next, we provide from each Algorithm two application examples: the first one concerns a simple dataset of graphs composed of few nodes and the second one is based on a dataset of many nodes. At the end of the chapters, we present the results of our work in the form of curves, graphs and tables.

Finally, we will end this thesis by returning to our proposal and by proposing some perspectives and future wor

CHAPTER 1

BASIC CONCEPTS

CHAPTER 1: BASIC CONCEPTS

1 INTRODUCTION

The purpose of this chapter is to introduce the basic concepts that will be used throughout this work. It is divided into two parts. In the first part, we will introduce the notion of complex networks and their different types with their characteristics. The second part will present the notions related to graph theory, which is mainly used for the study of complex networks.

2 Complex network

2.1 Complex network definition

Complex network designates all graphs with a high number of *vertices* (or *nodes*) and *edges*. It differs from the graphs traditionally studied in social network analysis where the number of vertices rarely exceeds forty [1].

Complex networks are in fact a subclass of complex systems. A complex system is generally considered as a set of mutually interacting elements, where the global behavior of the system cannot be deduced from the sum of its parts and their properties.

2.2 Complex network domains

Complex networks are present in many different fields: biology, sociology, psychology, computer science, They cover networks as diverse as the Internet, human networks, or even protein networks. Thus, many studies have been done on them [2]. These networks can be grouped into four categories[1]:

- social networks
- information networks
- technological networks
- biological networks

A *Social Network* (SN) is a kind of network that reflects the social structure of its nodes and their interdependency, such as friendship of people, co-authorship of researchers, and collaboration between different parties. A SN can be treated as a complex network, which is made up of individuals in the society and their relationships among the individuals. The scale of the network is usually very large. This kind of complex social structure plays an important role in dissemination and diffusion of information[3].

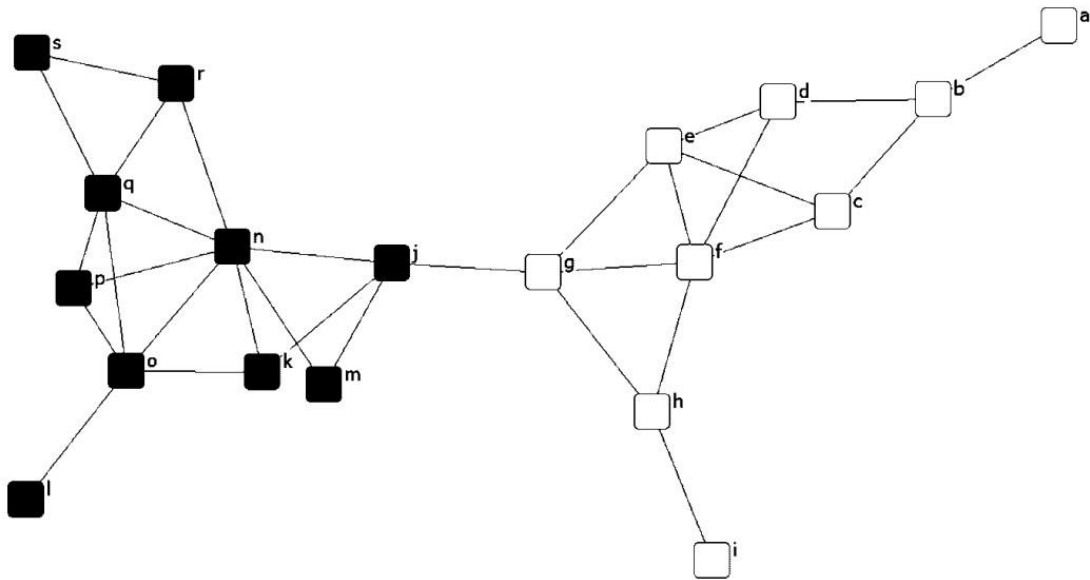


Figure 1. Graphical representation of a theoretical social network [4].

An *information network* (IN) can be related to the classical example of a network of citations between scientific papers. The structure of the information is stored in the nodes, which is why we use the term information network. The World Wide Web with its web pages (containing information) and its hyperlinks is also an information network (not to be confused with the Internet which is the physical network connecting computers all over the world) [1].

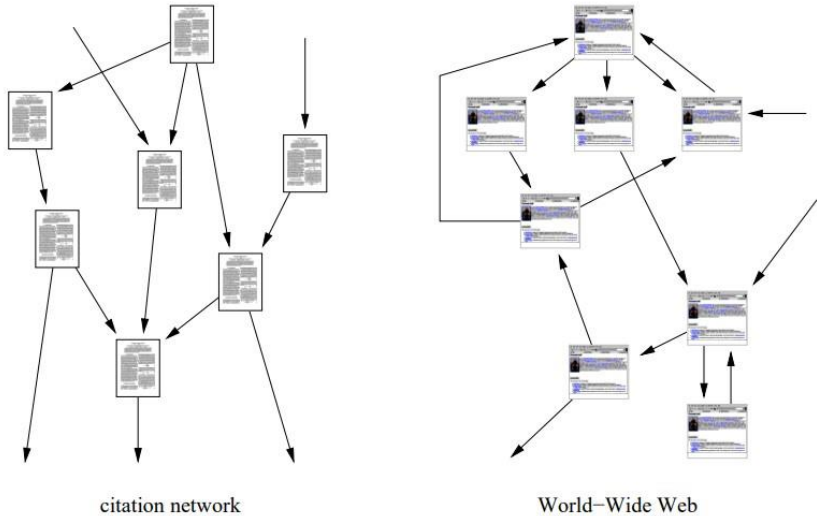


Figure 2. The two best studied information networks [1].

A *technology network* (TN) is a man-made network primarily for the distribution of a service or energy. Electrical, aerial and computer networks are part of this [1].

A *biological network* (BN) is a network of elements related to the living. An example of a biological network is a network of interactions between proteins [1].

2.3 Complex network types

There are two families of complex network: the strongly evolving networks and the small-world networks.

2.3.1 Strongly evolving network

We use this term to distinguish between, on the one hand, a network for which we have a small number of evolutionary steps, for example a citation network for which we have one snapshot per year, and, on the other hand, networks for which we have all the details of its evolution. If it is always arbitrary to give a precise definition of this notion, we can define a strongly evolving network as a network for which the number of evolution steps is higher than the total number of nodes of the network [5].

2.3.2 Small-world network

The notion of small world network can be slightly different. The general consensus is to characterize them by the following properties:

- **Short distance between nodes:** the average number to link two n nodes of the network chosen randomly by the shortest chain remains very small, whatever the size of the network. This average distance is of the order of $\ln(n)$. In practice, recent studies [6] on the Facebook network would tend to show that this number can still be small.
- **Strong clustering:** This property means that nodes tend to create dense local structures. It can come from transitivity: if there is a link (a, b) and a link (b, c), then the probability that a link (a, c) exists is reinforced [5].
- **Community structure:** This property is strongly related to the previous one. In a small world network, we observe microscopic structures, that is to say sets of nodes strongly linked to each other and more weakly linked to the rest of the network.

3 Graph theory

Graph theory was born in 1736 when Leonhard Euler demonstrated that it was impossible to cross each of the seven bridges in the Russian city of Königsberg (now Kaliningrad).

Graph theory then constitutes a field of mathematics which, historically, has also been used in various disciplines such as chemistry (structure modelling), biology (genome), social sciences (modelling of relationships) or for industrial applications (problem of the travelling salesman).

In general, a graph allows to represent simply the structure, the connections, the possible paths of a complex set including a great number of situations, by expressing the relations, the dependences between its elements.

3.1 Concepts and notations [7]

Graph theory is mainly the most used tool in the various studies of complex networks. In the following, we will give some definitions of the concepts of this theory and some notations concerning them. We will also present temporal graphs which represent a particular category for modeling dynamic networks.

Definition 1. A **graph** $G = (V, E)$ consists of a set V of **vertices** (also called **nodes**) and a set E of **edges**.

Definition 2. If an edge connects to a vertex we say the edge is **incident** to the vertex and say the vertex is an **endpoint** of the edge.

Definition 3. If an edge has only one endpoint then it is called a **loop edge**.

Definition 4. If two or more edges have the same endpoints then they are called **multiple** or **parallel** edges.

Definition 5. Two vertices that are joined by an edge are called **adjacent** vertices.

Definition 6. A **pendant** vertex is a vertex that is connected to exactly one other vertex by a single edge.

Definition 7. A **walk** in a graph is a sequence of alternating vertices and edges $v_1e_1v_2e_2 \dots v_n e_n v_{n+1}$ with $n \geq 0$. If $v_1 = v_{n+1}$ then the walk is **closed**. The **length** of the walk is the number of edges in the walk. A walk of length zero is a **trivial walk**.

Definition 8. A **trail** is a walk with no repeated edges. A **path** is a walk with no repeated vertices. A **circuit** is a closed trail and a **trivial circuit** has a single vertex and no edges. A trail or circuit is **Eulerian** if it uses every edge in the graph.

Definition 9. A **cycle** is a nontrivial circuit in which the only repeated vertex is the first/last one.

Definition 10. A **simple graph** is a graph with no loop edges or multiple edges. Edges in a simple graph may be specified by a set $\{v_i, v_j\}$ of the two vertices that the edge makes adjacent. A graph with more than one edge between a pair of vertices is called a **multigraph** while a graph with loop edges is called a **pseudograph**.

Definition 11. A **directed graph** is a graph in which the edges may only be traversed in one direction. Edges in a simple directed graph may be specified by an ordered pair (v_i, v_j) of the two vertices that the edge connects. We say that v_i is **adjacent to** v_j and v_j is **adjacent from** v_i .

Definition 12. The **degree** of a vertex is the number of edges incident to the vertex and is denoted $\deg(v)$.

Definition 13. In a directed graph, the **in-degree** of a vertex is the number of edges **incident to** the vertex and the **out-degree** of a vertex is the number of edges **incident from** the vertex.

Definition 14. A graph is **connected** if there is a walk between every pair of distinct vertices in the graph.

Definition 15. A graph H is a **subgraph** of a graph G if all vertices and edges in H are also in G .

Definition 16. A **connected component** of G is a connected subgraph H of G such that no other connected subgraph of G contains H .

Definition 17. A graph is called **Eulerian** if it contains an Eulerian circuit.

Definition 18. A **tree** is a connected, simple graph that has no cycles. Vertices of degree 1 in a tree are called the **leaves** of the tree.

Definition 19. Let G be a simple, connected graph. The subgraph T is a **spanning tree** of G if T is a tree and every node in G is a node in T .

Definition 20. A **weighted graph** is a graph $G = (V, E)$ along with a function $w: E \rightarrow \mathbb{R}$ that associates a numerical weight to each edge. If G is a weighted graph, then T is a **minimal spanning tree** of G if it is a spanning tree and no other spanning tree of G has smaller total weight.

Definition 21. The complete graph on n nodes, denoted K_n , is the simple graph with nodes $\{1, \dots, n\}$ and an edge between every pair of distinct nodes.

Definition 22. A graph is called **bipartite** if its set of nodes can be partitioned into two disjoint sets S_1 and S_2 so that every edge in the graph has one endpoint in S_1 and one endpoint in S_2 .

Definition 23. The **complete bipartite graph** on n, m nodes, denoted $K_{n,m}$, is the simple bipartite graph with nodes $S_1 = \{ a_1, \dots, a_n \}$ and $S_2 = \{ b_1, \dots, b_m \}$ and with edges connecting each node in S_1 to every node in S_2 .

Definition 24. Simple graphs G and H are called **isomorphic** if there is a bijection f from the nodes of G to the nodes of H such that $\{v, w\}$ is an edge in G if and only if $\{f(v), f(w)\}$ is an edge of H . The function f is called an isomorphism.

Definition 25. A simple, connected graph is called **planar** if there is a way to draw it on a plane so that no edges cross. Such a drawing is called an **embedding** of the graph in the plane.

Definition 26. For a planar graph G embedded in the plane, a **face** of the graph is a region of the plane created by the drawing. The area of the plane outside the graph is also a face, called the unbounded face.

3.2 Temporal graphs

Temporal graphs are oriented acyclic graphs. The presence of an arc between two nodes n_1 and n_2 translates the fact that n_2 is temporally located after n_1 . We distinguish two types of temporal graphs. We will address the type called contract sequences[8].

3.2.1 Contact sequences:

In this type of graph, a link is represented by a triplet (i, j, t) , such that i and j are nodes of the graph and t is the instant when the link was activated. This triplet thus represents the interaction between individuals i and j at time t . Contact sequences are particularly used to represent relationships between individuals for which the duration of the interaction is not known, such as asynchronous communication networks (emails, mails), or telephone communication networks, etc. [9].

4 Conclusion:

In this chapter we have presented the basic notions concerning complex networks, their types and properties. We have also described the concepts and notation related to graph theory. The latter is considered as the most adequate tool for the representation of complex networks. In particular, we have introduced the concept of temporal graph which is dedicated to the study of complex networks evolving in time. The following chapter will present one of the essential problems of these networks which is the detection of communities.

CHAPTER 2

COMMUNITY DETECTION

CHAPTER 2:

1 INTRODUCTION

In this chapter, we introduce the notion of community as well as the problem of community detection. On the other hand, we will present the different approaches to detection. Thus, we will focus our attention on dynamic detection approaches.

2 Definition of a community:

A characteristic of complex graphs is the possibility of dividing them into communities. According to Newman and Girvan in 2004 [10] a community is defined as a subgraph composed of nodes densely linked to each other and weakly linked to other nodes of the graph.

The communities may have different interpretations depending on the type of network considered. For information networks, they correspond to web pages dealing with the same subject. For metabolic networks, communities correspond for example to biological functions of the cell, etc. Community detection is therefore an important tool for understanding the structure and functioning of complex networks.

3 Detection of communities:

The detection of communities is a crucial problem when analyzing complex systems, for example, one may wish to study the interactions between individuals, between proteins or the links between different websites. These data can be represented in the form of a graph where each node represents an individual and a line an interaction between two individuals. These networks have the property of dividing into communities. This problem was first posed in the article by Girvan and Newman in 2002 [11].

The detection of communities approaches the two classical themes of Graph Partitioning and Data Clustering. The first one, initially introduced for the parallelization of processes, seeks to distribute the tasks represented by the vertices of a graph while minimizing the exchanges, represented by the edges.

The second theme of data clustering is a more general theme in which we try to group data with common characteristics.

4 Detection of dynamic communities:

Many works on community definition and detection have been proposed in the last years. However, in complex networks, interactions between communities evolve dynamically over time. For example, in the social network Facebook, users add or remove friends. The data is very often dynamic and a large amount of information is therefore completely ignored. This dynamism is a new challenge that has recently appeared in the detection of communities.

4.1 Operations on dynamic communities:

When communities are required to evolve, they can do so in several ways [12].

- **Growth and Contraction:** means the addition and removal of a node in an existing community.
- **Merging and splitting:** are two slightly more complex operations, corresponding to the regrouping of two communities into a single community, or the decomposition of a community into two new communities.
- **Birth and Death:** means the appearance of new communities and the disappearance of old communities.
- **Resurgence:** a community can only disappear for a period and come back after this period as if it has never stopped. For example, in the case of the soccer team, let's imagine that they stop to play together during the summer vacations, while a snapshot of the network is taken every month. The community will be present in the snapshots of June and September, and all other months to the exception of the snapshots corresponding to July and August. This can be modeled as a resurgence of the community after two months.

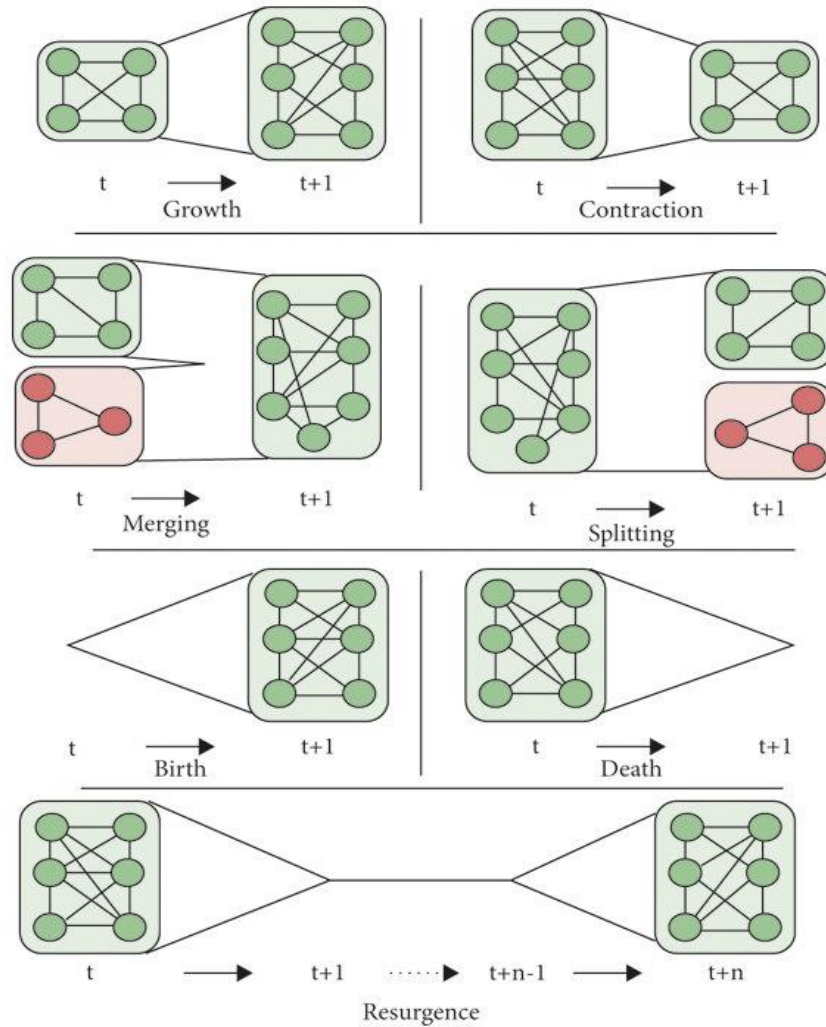


Figure 3. Operations in dynamic communities[13].

5 Community detection Methods:

5.1 Approaches based on static algorithms:

Static community detection methods are classified into three categories. The first one contains the hierarchical classification methods which allow to choose a community structure by several hierarchical levels representing different possible structures. The second identifies communities by maximizing a quality function. Finally, the last category concerns heuristic-based methods whose formalisms are applied to the nodes of the network in an iterative way until a stable community structure is obtained [14].

5.1.1 Hierarchical approaches

Hierarchical methods work on several levels. They are divided into two types, the first is the bottom-up (agglomerative) hierarchical algorithms and the second is the top-down (divisive) hierarchical algorithms.

The idea of the first type is that the vertices of a graph are iteratively grouped into communities starting from a partition of ' n ' communities composed of a single vertex. The groupings of communities are continued until a single community is obtained which groups all the vertices and a hierarchical structure of communities which is called a *dendrogram*.

The second type of hierarchical methods consists in dividing the network into several communities by iteratively eliminating the links between the nodes. Starting with a single community (the whole network), at the top of the *dendrogram*, until we have ' n ' communities at a single node representing the leaves of the *dendrogram*. In each iteration, any related network is considered as a community.



Figure 4. Agglomerative and Divisive Hierarchical clustering algorithms

Hierarchical methods have the advantage of finding small communities even in a very large graph. However, the disadvantage with these approaches is the determination of the threshold of the dendrogram cut, as well as the lack of overlapping of the classes.

5.1.2 Approaches based on the optimization of an objective function

The first modern method for community detection, still used in several domains, is the one proposed in 2002 by Girvan and Newman [11].

This method is based on the principle of divisive algorithms. Initially, all nodes in the network are considered to belong to a single community. We then successively remove the links, one by one, always removing the one with the maximum intermediary centrality. Gradually, the graph becomes non-connected, and each of the connected components thus formed is a community. The result is a *dendrogram*, containing at its root one community, two in the next step, then three, and so on until each node forms its own community (tree roots). For the splitting of the *dendrogram* in an optimal way, there was penetration of modularity, which is a metric used in the field of community detection.

This metric could be considered as a definition of what a "good community" is. From then on, they had the idea to directly search for the community partitioning corresponding to the maximum value of the modularity for a given graph. The problem of community detection thus became a mathematical problem of optimization, consisting in exploring a space of solutions to find the one corresponding to the maximum of modularity, this one was defined by Girvan and Newman of 2002 [11]. It is calculated as the difference between the proportion of internal links in the community and the proportion of links that random communities of the same size would have, this quality function is the following:

$$M = \sum C_i e(C_i) - a(C_i)^2 \quad (1)$$

Where: $e(C_i)$ represents the proportion of edges with both ends in the community with node i . and $a(C_i)$: is the probability that an edge has an end in the community C_i .

This function has been used in several works like [10], [15], [16],[17]

The problem highlighted is that modularity presupposes certain properties of the communities from the properties of the network to be studied. For a given size of network with a given density, the modularity will not be able to find communities smaller than $\sqrt{\frac{M}{2}}$.

Any smaller community, even one that is clearly separated from the network, will be merged with other communities to obtain communities of the size expected by the modularity.

Moreover, in large graphs there are a large number of partitions that have modularity values that are very close to the maximum modularity and yet correspond to very different partitions.

5.1.3 Approaches based on cliques

In this category of approaches a community is defined as a chain of adjacent k -cliques. A k -clique is a subset of k nodes all adjacent to each other, and two k -cliques are adjacent if they share $k-1$ nodes. The idea of these algorithms is that, starting from k -cliques, to build little by little the communities. The immediate advantage of such an approach is the detection of overlapping communities, where a node can belong to several k -cliques that are not necessarily adjacent. CPM (clique percolation method) is the first proposed method. It is the basis of many works like [12]. Later on, like the EGALE algorithm [18].

CPM method [12]:

Palla et al propose the **CFinder** algorithm which is structured in three main steps:

- Compute the set of cliques of size k (parameter of the algorithm) in the target graph G .
- Construct a graph of cliques where each clique is represented by a node. Two nodes are connected by a link if the two associated cliques share $k-1$ nodes in the graph G .

The communities in the graph G are then the related components identified in the clique graph constructed in step 2.

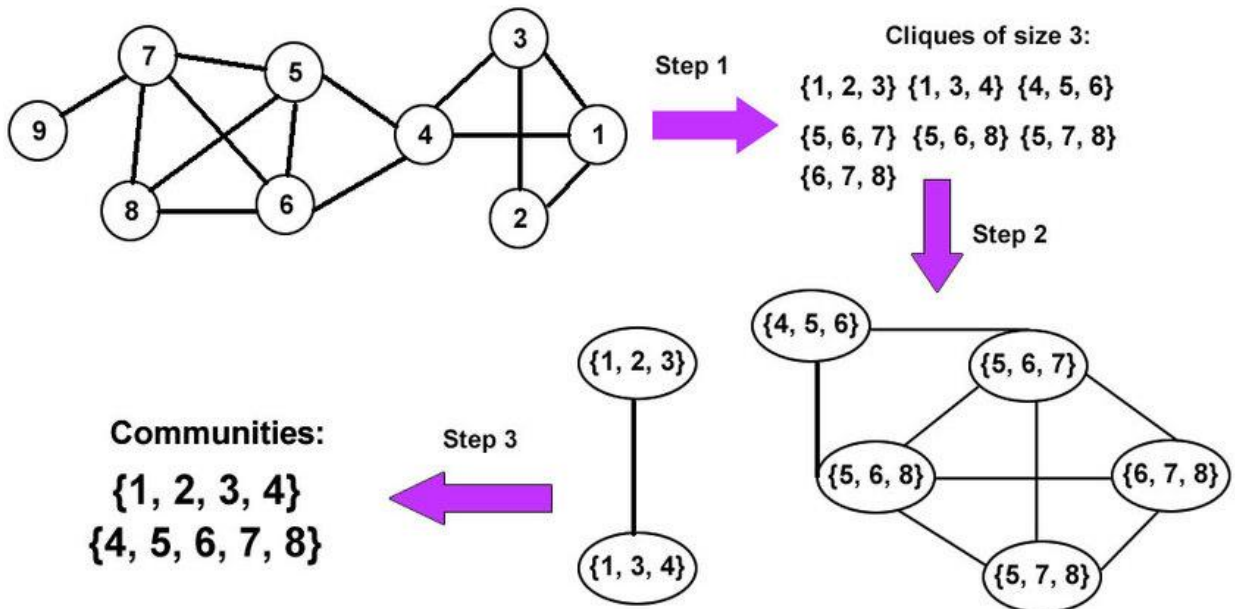


Figure 5. An example of the clique percolation algorithm with $k = 3$

This approach is conceptually simple. Also, its advantage is that it works very well in field networks. Another limitation is related to the parameterization: the value of k (the size of the communities to consider). Moreover, this method is sensitive to certain configurations. For example, if we imagine a sequence of cliques of size k , having $(k-1)$ nodes in common, and forming a chain, this method will detect them as a community, whereas this is generally not relevant.

5.2 Dynamic approaches:

The study of dynamic communities is carried out in two broad directions:

To study communities among different captures using algorithms adapted for static graphs or to use directly the temporal information during the detection.

5.2.1 Approaches based on network snapshots

There are two categories of approaches working on network snapshots. The first category is the successive static detection approaches. The idea is to consider the dynamic graph as a succession of independent snapshots. A first step is to apply a static algorithm on each of these snapshots, which allows to obtain a series of partitions, one for each snapshot. Then to find a correspondence (association) between the existing communities in consecutive time instants. Many works fall into this category [12], [19], [20]

Chen and al. algorithm [20]

Chen et al. use pure nodes defined as the nodes existing at time $t-1$, t and $t+1$ to reduce the number of communities to consider.

They define the communities initially as the maximal cliques and can thus have overlapping communities. Nevertheless, the number of communities can be high and they use the notion of core nodes to consider only communities containing core nodes and thus reduce their number.

Obviously, the main weakness of this method is that it defines the communities as being maximal cliques, which, on the one hand, often gives communities that are not very relevant and, on the other hand, leads to a far too large number of communities in large graphs with a high density of large graphs with a high density.

The second category is that of the successive static informed detection approaches. They propose to consider the results obtained at time t during the detection of communities at time $t+1$. This allows to reduce the instability of the algorithms, i.e. the algorithm gives quite distant results for very close graphs. By imposing the choice between two different decoupling, and could take the most similar to the previous decoupling, this approach is applied in the works [21], [22], [23] In the following we have detailed an algorithm that uses the second category.

Lin and al. algorithm [21]

This algorithm proposes a solution based on a probabilistic generative model, which consists in formulating a quality function as a non-negative matrix factorization problem that jointly optimizes the quality and stability of communities.

Although this method has the advantage of allowing the detection of overlapping communities, it imposes strong constraints: the number of communities must be known in advance, and it is a priori not possible to add or remove nodes over time. It also does not allow operations such as merging or splitting communities.

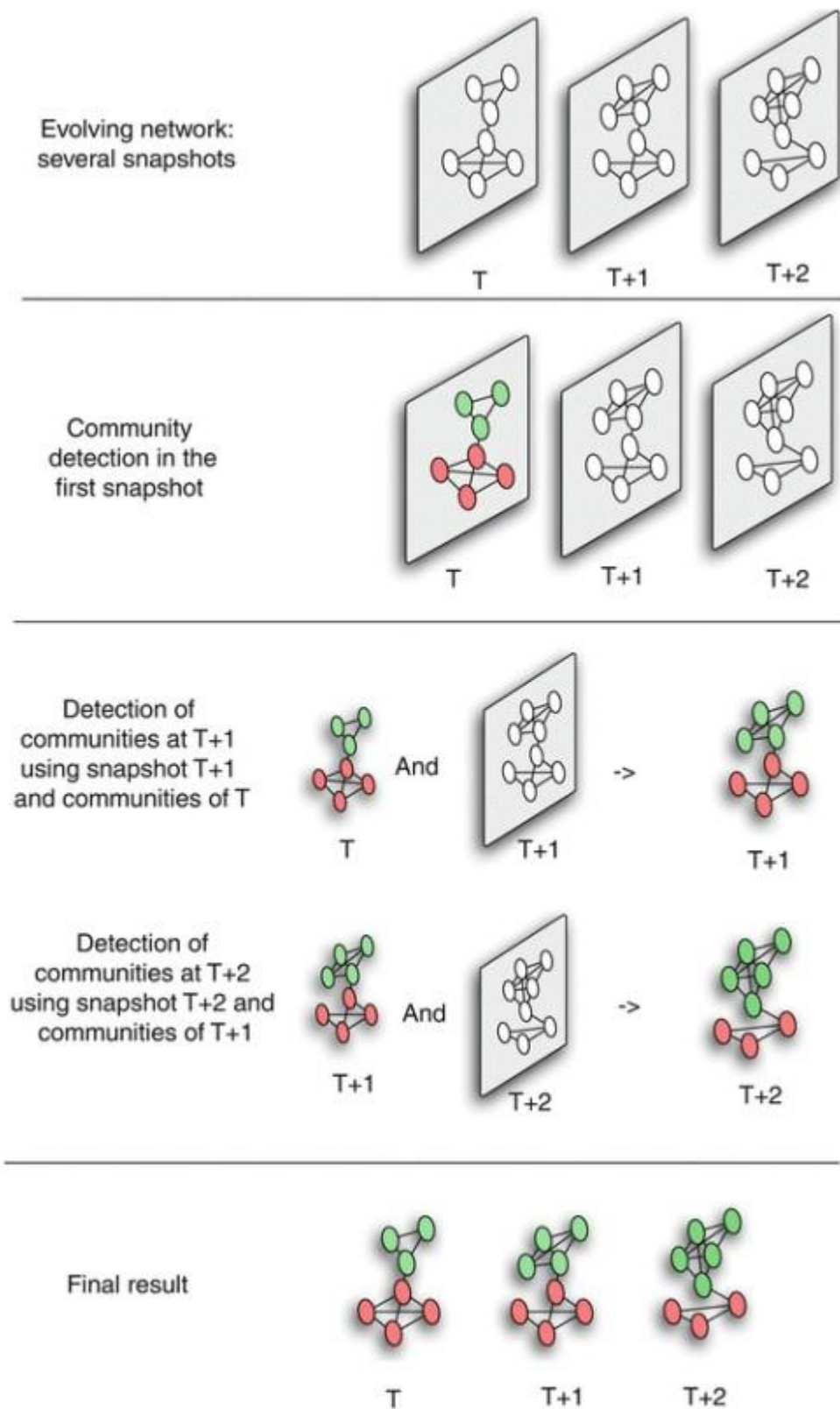


Figure 6. Representation of a method by successive static informed detections

5.2.2 Approaches working on temporal networks

In the algorithms working directly on the temporal network, the evolution of the network is considered as a succession of modifications on the network. The idea is to consider the last modifications made on the network, and to modify the existing communities accordingly. Its advantage is that there is no more problem of instability. There are two algorithms that work based on this method and that treat the case of overlapping. We will describe them in the next section.

Falkowski and al. algorithm [24]:

Falkowski et al first define a distance between the nodes of a network and then define the neighborhood of a node as the topological ball of radius R (varying between 0 and 1). They consider only the nodes whose neighborhood is larger than a given boundary S and consider them as core nodes. The nodes present in the neighborhood of a core node are border nodes. They then define communities as the unions of neighborhoods sharing nodes.

They then propose techniques to update neighborhoods and communities: at each new step, the distance values between nodes are updated. If one of these modifications makes a new core node appear, it is integrated into a new community, or a new one is created if it has no core node in its neighborhood. In the same way, if the nodes that were in the radius r of a core node go beyond this radius, they leave the community, and so on.

However, the problem with this solution is that the definition of community used is very particular, and is quite far from what is generally considered as a good community. Moreover, it still depends on the chosen values of parameters R and S .

Cazabet Algorithm [5]

The **iLCD** (intrinsic Longitudinal Community Detection) algorithm proposed by Cazabet[5], is innovative because it was one of the first algorithms to be able to detect dynamic communities in a temporal network, i.e., communities that realistically change when the network evolves.

iLCD describes the way community detection should be performed and represents the set of actions performed (T_e) on temporal networks. The sequence of actions T_e is composed of quadruplets (i, j, a, t) with i and j the affected nodes, a , the action that takes place, which can be an addition or a

deletion of a link, and t the instant at which this modification takes place. These actions are ordered by increasing t . The principle of **iLCD** is the following:

- For each addition of a link (i, j) in the network, if i is in a community and j does not belong to C , we look if j must be integrated in C .
- Then we find all the new communities formed by the appearance of this new link. New communities that are not included in an existing community are kept.
- For each deletion of a link (i, j) , if this link is inside a community ($i \in C \wedge j \in C$), we calculate if C loses one or more nodes, which can cause it to split.
- After each change in the network, if one or more communities have been changed. We look if they should be merged with other communities. The candidate communities are those that share nodes with the modified community.

The **iLCD** algorithm uses two metrics:

Representativeness: is a metric that indicates how representative a node is of a community. It is defined by the following function:

$$RP(i, c) = \frac{d^{int_c(i)}}{d(i)} \quad (2)$$

Where: $d^{int_c(i)}$: is the internal degree of node i in the community c .

$d(i)$: is the total degree of node i .

Membership strength: it is quantified by a node i and a community C . Its value is bounded between 0 and 1. It is defined by the following function:

$$FA(i, c) = \sum_{j \in N_c(i)} RP(j, c) \quad (3)$$

Where: $N_c(i)$: are the nodes of the community where node i is located.

iLCD is not based on modularity, but rather on the idea that communities are defined locally. This algorithm allows to follow the operations of communities (birth, death, division, death). The advantage of this algorithm is that there is no more instability problem. However, this algorithm has some limitations.

First of all, it does not consider the hierarchical aspect and it is only applicable on non-oriented graphs. Moreover, **iLCD** does not perform a division operation in the real sense of the word but it treats it in a partial way.

6 State of the art synthesis

A comparison between the different approaches is provided in the table below. We have considered the following criteria:

- 1) **Instability**: the algorithm gives quite distant results for very close graphs.
- 2) **Hierarchical**: the algorithm allows a community detection at several levels.
- 3) **Large graph**: the algorithm is applicable on complex graphs.
- 4) **Iterative**: the final result provided by the algorithm appears only after several executions.
- 5) **Parameterization**: the algorithm requires one or more input parameters.
- 6) **Community overlapping**: the algorithm allows to detect the multiple membership of a node to several communities.

The following symbols are also used:

K: size of the initial communities

R: radius of the node

S: neighborhood boundary

Algorithms criteria	Static approaches				Dynamic approaches			
	Girvan and Newman 2002	CPM 2007	Lin and All 2008	Eagle 2009	Palla 2007	Falkowski 2008	Lin and All 2012	ILCD 2013
overlapping	+	+	+	+	+	+	-	+
Instability	-	-	-	-	+	+	+	-
Hierarchical	-	-	-	-	-	-	-	-
Large graph	-	+	-	+	+	+	+	+
Iterative	+	+	-	+	-	+	+	-
Parameterization	-	K	-	-	-	S, R	+	-

Table 1. Comparison of community detection approaches

From the state of the art, we found that there are a large number of iterative algorithms for community detection. These algorithms are often costly in execution.

Moreover, in methods requiring parameterization such as CPM [12], the results depend on the input parameters. However, this method (CPM) is among the first to be able to detect overlapping communities and to be applied in large graphs in contrast to traditional approaches like Girvan and Newman [11].

The state-of-the-art shows that there are two approaches for dynamic community detection. The first approach considers the graph as a succession of snapshots and has the clear advantage of reusing static algorithms. However, this approach has a very clear limitation which is instability.

In the second approach, working on temporal graphs, there is no more instability problem. Nevertheless, the evolution of communities is not consideration. For example, in the work of Cazabet [5] the division operation is partially implemented.

7 Modularity

Comparing results of different network partitioning algorithms can be challenging, especially when network structure is not known beforehand. A concept of modularity defined in [15] provides a measure of the quality of a particular partitioning of a network. Modularity (Q) quantifies the community strength by comparing the fraction of edges within the community with such fraction when random connections between the nodes are made. The justification is that a community should have more links between themselves than a random gathering of people. Thus, the Q value close to 0 means that the fraction of edges inside communities is no better than the random case, and the value of 1 means that a network community structure has the highest possible strength.

Formally, modularity (Q) can be defined as:

$$Q = \frac{1}{2W} \sum_{i,j} B_{ij} \delta(C_i, C_j), B_{ij} = A_{ij} - P_{ij} \quad (4)$$

$$Q = \sum_{c_i \in C} \left[\frac{|E_{c_i}^{in}|}{|E|} - \left(\frac{|E_{c_i}^{in}| + |E_{c_i}^{out}|}{2|E|} \right)^2 \right] \quad (5)$$

where C is the set of all the communities, c_i is a specific community in C , $|E_{c_i}^{in}|$ is the number of edges between nodes within community C , $|E_{c_i}^{out}|$ is the number of edges from the nodes in community c_i to the nodes outside c_i , and $|E|$ is the total number of edges in the network.

Modularity can also be expressed in the following form:

$$Q = \frac{1}{2|E|} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2|E|} \right] \delta_{c_i, c_j} \quad (6)$$

where k_i is the degree of node i , A_{ij} is an element of the adjacency matrix, δ_{c_i, c_j} is the Kronecker delta symbol, and c_i is the label of the community to which node i is assigned.

8 Conclusion:

In this chapter we have examined the different static and dynamic approaches to community detection. Three static approaches have been presented: hierarchical approaches, approaches based on modularity optimization and clique-based approaches. The latter is very simple and is the first to be able to detect overlapping communities.

For dynamic approaches, we have distinguished two categories. The first one considers the network as instantaneous and uses static algorithms. It suffers from the problem of instability. The second approach using temporal networks eliminates the instability problem. However, it does not allow to follow all the evolutionary stages of the communities. Our proposal, which will be described in the next chapter, is interested in this last category and tries to follow the whole evolution of communities.

CHAPTER 3:

MODULARITY MAXIMIZATION ALGORITHMS

CHAPTER 3:

1 INTRODUCTION

In this chapter we will talk about different algorithms we used in our project which is based on comparing various Community Detection using modularity maximization algorithms on synthetic generated network using Lancichinetti-Fortunato-Radicchi (LFR) benchmarks.

2 Clauset-Newman-Moore (CNM) algorithm

The method of Clauset, Newman and More (CNM)[15] is a heuristic method designed to identify communities of large-scale networks in a fast and efficient manner. Due to the greedy nature of CNM, its application may lead to partitions which differ from the optimal solution and, in many cases, the modularity obtained is lesser than what could be obtained using other approaches.

The CNM method uses an algorithm proposed by Newman (later improved by Clauset et al.) as follows:

1. Associate each node of the network with a community;
2. Repeatedly combine the communities to produce the highest increase in modularity;
3. After $n-1$ combination the result is only one community containing all the nodes, and the algorithm stops (considering a network of n nodes).

Newman [2] proposes a strategy to evaluate the gain obtained by the union of two generic communities C_a and C_b , taking an $n \times n$ matrix as basis. The union of two communities C_a and C_b corresponds to the substitution of the a^{th} and b^{th} lines (and columns) of the matrix by their sums.

However, as shown by Clauset et al, the sparsity of the matrix results in a memory waste and a high execution cost to apply the union of communities over the complete lines/columns.

Thus, Clauset et al. propose a matrix M in order to store the modularity gain caused by the union of two generic communities C_a and C_b , keeping just the elements M_{ab} linked by at least one edge [1]. The elements M_{ab} of M are initialized as:

$$M_{ab} = \begin{cases} \frac{1}{2m} - \frac{d_a}{(2m)^2}, & \text{if } C_a \text{ and } C_b \text{ are connected} \\ 0, & \text{otherwise} \end{cases} \quad [3] \quad (7)$$

where d is a vector which stores the sum of the degrees of the nodes belonging to a community C_a and the elements d_a are defined as $d_a = \sum_i k_i, v_i \in C_a$ [4].

After calculating the initial value of M , the method performs successive unions of communities until no more gain in the modularity can be obtained. For the union of a particular pair of communities C_a and C_b (where the resulting community is stored as C_a), only the line and the column indexed by a must be updated.

```

Input: A network  $G = (V, E)$ 
Output: A community structure  $C = \{C_a, a = 1, \dots, nc\}$ 
Calculate the initial values for  $M$  (3);
Calculate the initial modularity value  $Q$ ;
 $nc \leftarrow n$ ;
repeat
  Join the pair of communities  $C_a$  and  $C_b$  corresponding to the highest value of  $M$  ( $\max(M)$ ):  $Mab$ ;
  Update matrix  $M$  (2);
   $nc \leftarrow nc - 1$ ;
   $Q = Q + Mab$ ;
until  $\max(\Delta Q) < 0$ ;

```

Algorithm of Clauset, Newman, and Moore

3 Louvain Algorithm

Louvain algorithm is an efficient hierarchical clustering algorithm based on graph theory created by Blondel et al.[25] from the University of Louvain . The idea behind it is to continuously iterate using mobile nodes to increase the modularity of the community partition outcome and ultimately provide the best possible community partition.

The main steps of the Louvain algorithm are as follows:

- Step 1: Initialize the community and set each node as a separate community, namely, community 1: (node1), community 2: (node2), and so on.
- Step 2: Find out all the communities connected to node 1, and calculate the change of modularity after moving node 2 to each neighbor community. Move node 1 to the community, which can increase the modularity to the maximum.
- Step 3: Iterate over all the nodes and execute step 2 until there are no nodes to move and get a layer of community partition.
- Step 4: Merge each community in step 3 into a new node. The relationship between new nodes is the relationship between the original communities. Return to step 1 until all nodes are finally

merged into one community. The multilevel community partition is obtained, and the partition with the highest modularity is selected as the final partition result.

This algorithm has the advantage of requiring less calculation time each iteration than the Fast Newman algorithm. Meanwhile, when calculating the movement of a single node, only the modularity gain of the node in the two communities where the movement occurs needs to be calculated. So, the computational efficiency is obviously better than that of the Fast Newman algorithm and the time complexity of this algorithm is approximately to $\mathbf{o(m + n)}$.

The value to be optimized is modularity is defined within the range -1 and 1 that measures the density of links inside communities compared to links between communities. Modularity for a weighted graph is defined as [26]:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(C_i, C_j), \quad (8)$$

where:

A_{ij} is the adjacency matrix.

k_i and k_j are the degrees of nodes i and j respectively.

m is the number of edges.

the δ -function is the Kronecker function: 1 if both nodes i and j belong on the same community (C_i, C_j) , 0 otherwise [27] [28].

3.1 Community Aggregation

After finishing the first step, all nodes belonging to the same community are merged into a single giant node. Links connecting giant nodes are the sum of the ones previously connecting nodes from the same different communities. This step also generates self-loops which are the sum of all links inside a given community, before being collapsed into one node.

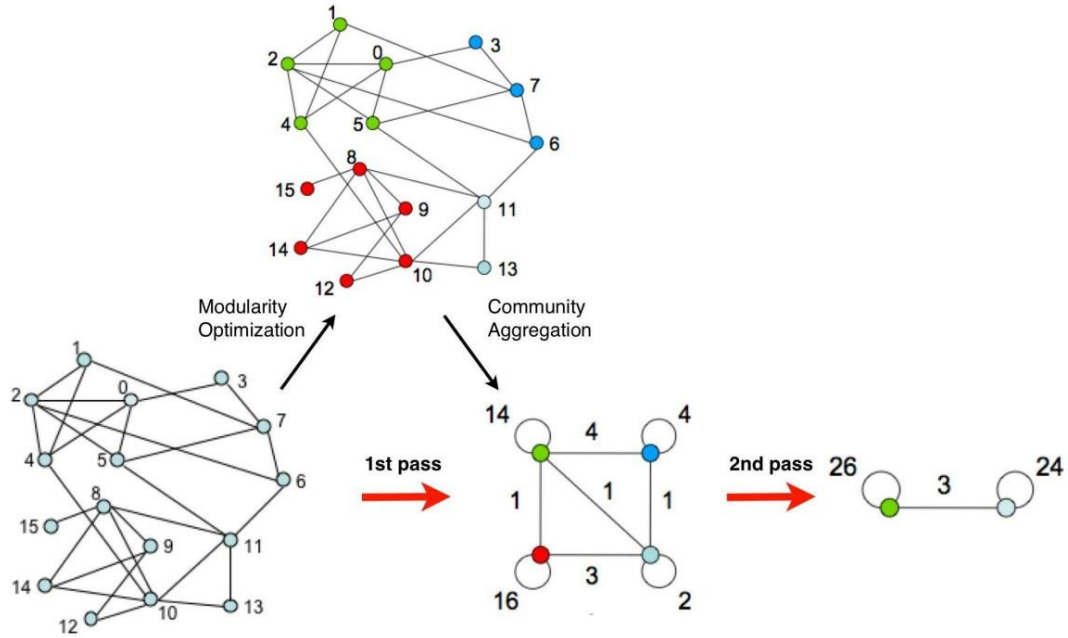


Figure 7. Sequence of steps followed by Louvain algorithm [25].

```

G the initial network
repeat
  put each node of G in its own community
  while
    for all node n of G do
      place n in its neighboring community including
      its own which maximizes the modularity gain
    end for
  end while
  if the new modularity is higher than the initial then
    G = the network between communities of G
  else
    Terminate
  end if
until
    
```

Algorithm: Pseudo-code of Louvain Method [29]

4 Leiden Algorithm

Traag et al [30] introduced the Leiden algorithm based on smart local move algorithm [31], which is also an improvement of the Louvain algorithm. The Leiden algorithm also takes advantage of the idea of speeding up the local moving of nodes[32], [33] and the idea of moving nodes to random neighbors[34]. The Leiden algorithm consists of three phases:

- local moving of nodes
- refinement of the partition and
- aggregation of the network based on the refined partition, using the non-refined partition to create an initial partition for the aggregate network.

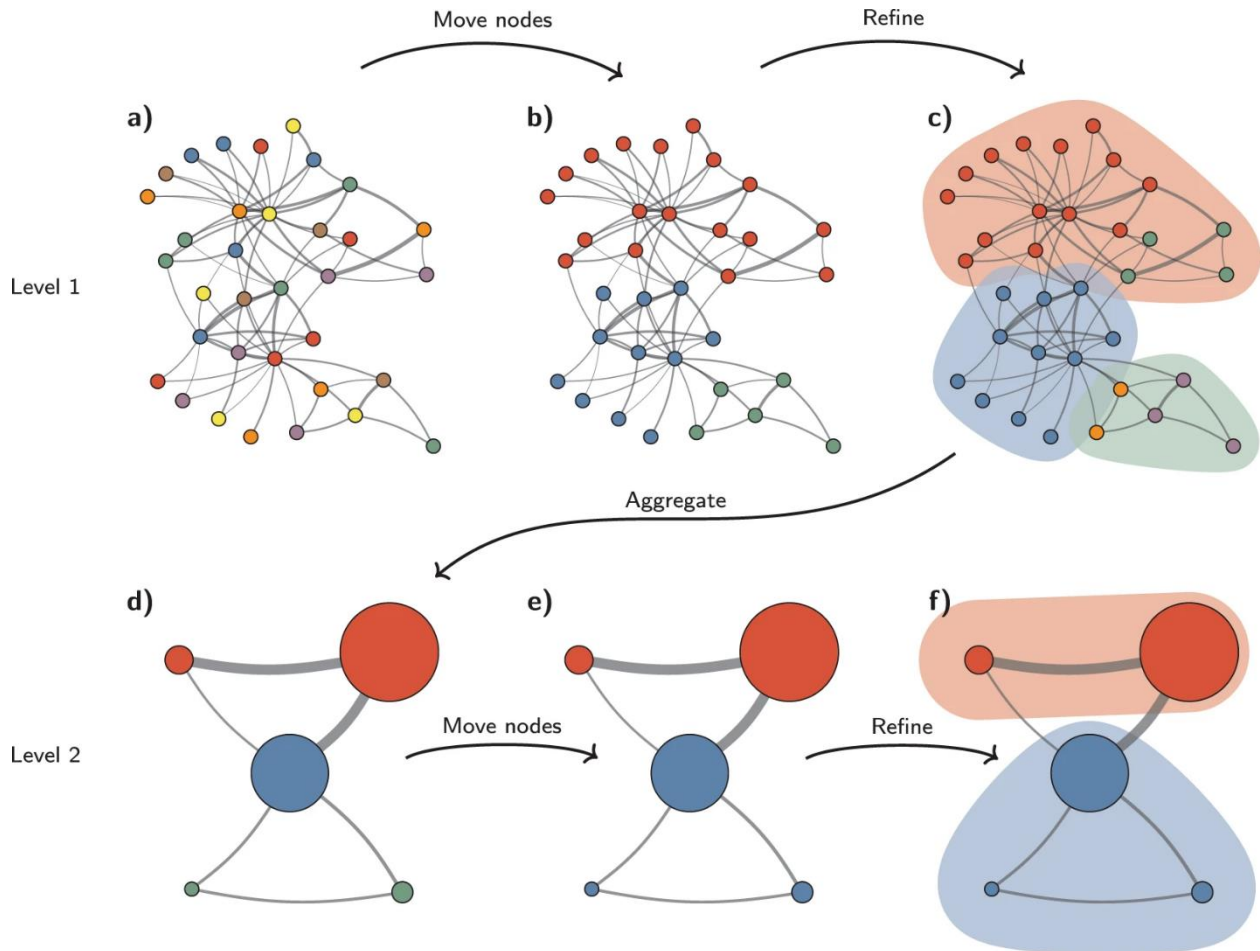


Figure 8. Illustration of the Leiden algorithm.

The Leiden algorithm starts from a singleton partition (**a**). The algorithm moves individual nodes from one community to another to find a partition (**b**), which is then refined (**c**). An aggregate network (**d**) is created based on the refined partition, using the non-refined partition to create an initial partition for the aggregate network. For example, the red community in (**b**) is refined into two subcommunities in (**c**), which after aggregation become two separate nodes in (**d**), both belonging to the same community. The algorithm then moves individual nodes in the aggregate network (**e**). In this case, refinement does not change the partition (**f**). These steps are repeated until no further improvements can be made.

After each iteration of the Leiden algorithm, it is guaranteed that

- 1) All communities are γ -separated.
- 2) All communities are γ -connected.
- 3) All nodes are locally optimally assigned.
- 4) All communities are sub-partition γ -dense.
- 5) All communities are uniformly γ -dense.
- 6) All communities are subset optimal.

```

function Leiden (Graph G, partition P)
  do
    p  $\leftarrow$  MoveNodesFast (G, P)
    done  $\leftarrow$  |P| = |V(G)|
    if not done then
      Prefined  $\leftarrow$  RefinePartition (G,P)
      G  $\leftarrow$  AggregateGraph(G, Prefined)
      P  $\leftarrow$  { {v / v  $\subseteq$  C, v  $\in$  V(G)} | C  $\in$  P }
    end if
  while not done
  return flat*(P)
end function

function MoveNodesFast(Graph G, Partition P)
  Q  $\leftarrow$  Queue(V(G))
  do
    V  $\leftarrow$  Q.remove()
    C'  $\leftarrow$  argmaxC $\in$ P $\cup$  $\emptyset$   $\Delta H_p$ (v  $\rightarrow$  C)
    if  $\mathcal{H}_p > 0$  then
      v  $\rightarrow$  C'(v  $\rightarrow$  C)
      N  $\leftarrow$  {u | (u,v)  $\in$  E(G), u  $\notin$  C'}
      Q.add(N-Q)
    end if

```

```

while  $Q \neq \theta$ 
  return P
end function

function REFINEPARTITION (Graph G. Partition P)
   $P_{refined} \leftarrow \text{SingletonPartition}(G)$ 
  For  $C \in P$  do
     $P_{refined} \leftarrow \text{MergoNodesSubset}(G, P_{refined}, G)$ 
  end for
  return  $P_{refined}$ 
end function

function MergoNodesSubset(Graph G. Partition P. Subset S)
   $R = \{v \mid v \in S, E(v, S - v) \geq \gamma \|v\| \cdot (\|S\| - \|v\|)\}$ 
  for  $v \in R$  do
    if  $v$  in singleton community then
       $T \leftarrow \{C \mid C \in P, C \subseteq S, E(S - C) \geq \gamma \|C\| \cdot (\|S\| - \|C\|)\}$ 
       $Pr(C' = C) \sim \begin{cases} \exp\left(\frac{1}{\theta} \Delta \mathcal{H}p(v \mapsto C)\right) & \text{if } \Delta \mathcal{H}p(v \mapsto C) \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{for } C \in T$ 
       $v \mapsto C'$ 
    end if
  end for
  return  $\mathcal{P}$ 
end function

function AggregateGraph(Graph G, Partition  $\mathcal{P}$ )
   $V \leftarrow \mathcal{P}$ 
   $E \leftarrow \{(C, D) \mid (u, v) \in E(G), u \in C \in \mathcal{P}, v \in D \in \mathcal{P}\}$ 
  return Graph( $V, E$ )
end function

function SINGLETONPARTITION(Graph G)
  return  $\{\{v\} \mid v \in V(G)\}$ 
end function

```

Pseudocode Algorithm Leiden (Traag et al., 2019)

5 Paris algorithm

Many datasets can be represented as graphs, being the graph explicitly embedded in data (e.g., the friendship relation of a social network) or built through some suitable similarity measure between data items (e.g., the number of papers co-authored by two researchers). Most graph clustering algorithms are not hierarchical and rely on some resolution parameter that allows one to adapt the clustering to the dataset and to the intended purpose [35], [36], [37], [38]. This parameter is hard to adjust in practice.

A novel algorithm for hierarchical clustering is called Hierarchical Graph Clustering by Node Pair Sampling (which is based on Louvain Algorithm) that captures the multi-scale nature of real graphs. The algorithm was referred as *Paris*¹ (developed by Thomas Bonaldis, Bertrand Charpentier, Alexis Galland and Alexandre Hollocou) is fast, memory-efficient and parameter-free. It relies on a novel notion of distance between clusters induced by the probability of sampling node pairs. We prove that this distance is reducible, which guarantees that the resulting hierarchical clustering can be represented by regular dendrograms and enables a fast implementation of our algorithm through the nearest-neighbor chain scheme, a classical technique for agglomerative algorithms [39].

The hierarchy is induced by the successive aggregation steps of the algorithm using Louvain Algorithm [25]. This is not a full hierarchy, however, as there are typically a few aggregation steps. Moreover, the same resolution is used in the optimization of modularity across all levels of the hierarchy, while the numbers of clusters decrease rapidly after a few aggregation steps.

6 The eigenvectors of matrices method for community detection

The “eigenvectors of matrices” by M. E. J. Newman is a method aiming at detecting communities or modules in networks, groups of vertices with a higher-than-average density of edges connecting them.

Unlike other methods which use the maximization of the modularity (the benefit function), this approach is different in which it rewrites the modularity function in matrix terms, which allows to express the optimization task as a spectral problem in linear algebra.

This approach leads to a family of fast new computer algorithms (such as leading eigenvector method, vector partitioning algorithm...) for community detection that produce results competitive with the best previous methods. This work is by no means the first to find connections between divisions of networks and matrix spectra [40], [41], [42]. Once an explicit expression for the modularity is built,

¹ Pairwise AgglomeRation Induced by Sampling

the community structure is determined by maximizing the modularity over possible divisions of the network.

7 Conclusion

In this chapter we have presented some community detection algorithms. These algorithms are based on modularity maximization on synthetic generated network using Lancichinetti-Fortunato-Radicchi (LFR) benchmarks. In the next chapter, we will focus on the implementation and validation of these algorithms.

CHAPTER 4:
REALIZATION &
IMPLEMENTATION

CHAPTER 4:

1 INTRODUCTION

In order to highlight the problem of community detection, we have given in the previous chapter a brief overview of the methods applied in the community detection.

In order to support these theoretical concepts, we propose in this chapter some applications on different set of nodes (250 to 10000). Our goal is to compare several methods in order to verify:

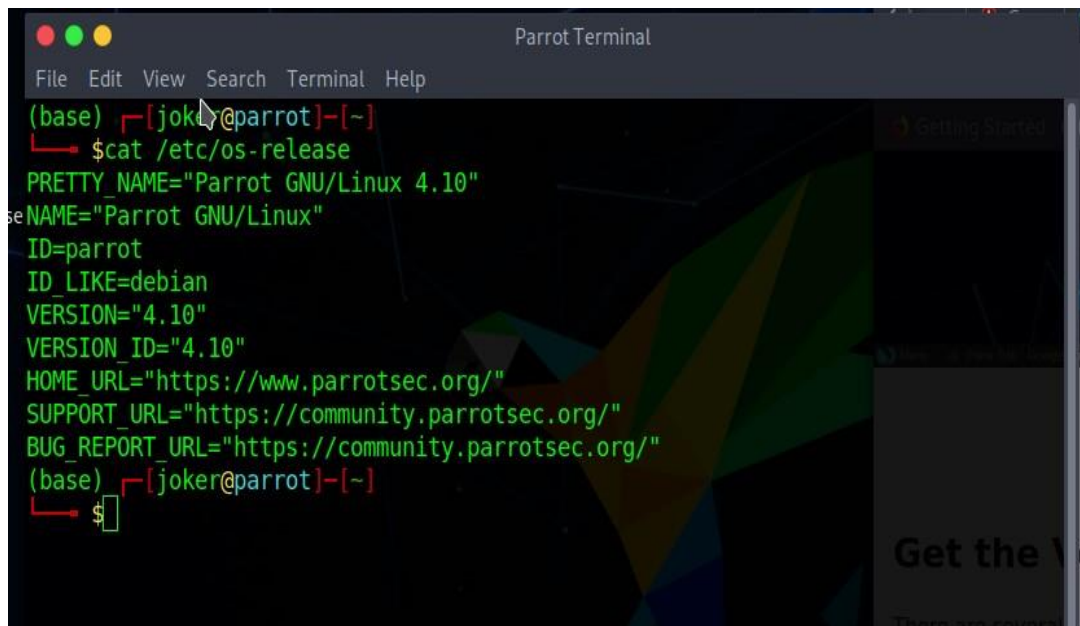
- The robustness of the solution (convergence to the global maximum).
- The performance of the numerical method (simulation time, number of iterations).
- The quality of the solution provided by each method.

All our applications are realized on the Jupyter Notebook where we computed our results using different methods. Our results from each algorithm have been compared.

2 Development Tools

2.1 Parrot OS:

is a Linux distribution based on Debian.²



```

Parrot Terminal
File Edit View Search Terminal Help
(base) [joker@parrot]~$ cat /etc/os-release
PRETTY_NAME="Parrot GNU/Linux 4.10"
NAME="Parrot GNU/Linux"
ID=parrot
ID_LIKE=debian
VERSION="4.10"
VERSION_ID="4.10"
HOME_URL="https://www.parrotsec.org/"
SUPPORT_URL="https://community.parrotsec.org/"
BUG_REPORT_URL="https://community.parrotsec.org/"
(base) [joker@parrot]~$
  
```

Figure 9. Parrot version

² <https://www.parrotsec.org/>

2.2 Pycharm IDE

Is an open source, single-language integrated developer environment (IDE) for Python projects created by JetBrains. ³

PyCharm features:

- Coding Assistance and Analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes
- Project and Code Navigation: specialized project views, file structure views and quick jumping between files, classes, methods and usages
- Python Refactoring: including rename, extract method, introduce variable, introduce constant, pull up, push down and others
- Support for web frameworks: Django, web2py and Flask
- Integrated Python Debugger
- Integrated Unit Testing, with line-by-line coverage
- Google App Engine Python Development
- Version Control Integration: unified user interface for Mercurial, Git, Subversion, Perforce and CVS with changelists and merge.

In this project we used **pycharm-community-2022.2**

2.3 Python3

Is a great object-oriented, interpreted, and interactive programming language. First appeared 20 February 1991⁴.

Installation:

1. Install the dependencies necessary to build Python

```
sudo apt update
sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev
libssl-dev libsqlite3-dev libreadline-dev libffi-dev curl libbz2-dev
```

³ <https://www.jetbrains.com/pycharm/>

⁴ <https://wiki.python.org/moin/FrontPage>

2. Download the latest release's source code from the Python download page with wget:

```
wget https://www.python.org/ftp/python/3.9.12/Python-3.9.12.tgz
```

3. Once the download is complete, extract the gzipped archive:

```
tar -xf Python-3.9.12.tgz
```

4. Navigate to the Python source directory and execute the configure script:

```
cd Python-3.9.12  
./configure --enable-optimizations
```

5. Start the Python 3.9.12 build process:

```
make -j 2
```

6. When the build process is complete, install the Python binaries by typing:

```
sudo make altinstall
```

```
input  
python3 -version  
output  
Python 3.9.12
```

2.4 Anaconda

Anaconda is a software development and consulting company of passionate open source advocates based in Austin, Texas, USA. founded by Peter Wang and Travis Oliphant in 2012⁵. Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.)

⁵ <https://docs.anaconda.com/anacondaorg/faq/#what-is-anaconda-inc>

Installation:

1.

```
shasum -a 256 /Downloads/Anaconda-2022.05-Linux-x86_64.sh
```

2.

```
bash ~/Downloads/Anaconda-2022.05-Linux-x86_64.sh
```

3. Press Enter to review the license agreement. Then press and hold Enter to scroll
4. Enter “yes” to agree to the license agreement.
5. Use Enter to accept the default install location
6. The installer prompts you to choose whether to initialize Anaconda Distribution by running *conda init*. Anaconda recommends entering “yes”.
7. The installer finishes and displays, “Thank you for installing Anaconda”
8. To open *anaconda navigator* just open terminal and type:

```
anaconda-navigator
```

more details about [anaconda](#)⁶.

2.5 Jupyter Notebook

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.⁷

Jupyter Notebook open in browser, you may have notice that the URL for the dashboard is something like *https://localhost:8888/tree*. Localhost is not a website, but indicates that the content is being served from your local machine: your own computer.

Installation :

```
conda install -c anaconda jupyter
```

⁶ [Installing on Linux — Anaconda documentation](#)

⁷ [Project Jupyter | Home](#)

2.6 Project requirements

To run our project, you need all mentioned software above, as well as for some required libraries:

1. Create new conda environment:

```
conda create --name FsProect (name of the environment) python==3.8
conda activate FsProect
```

2. Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.⁸

Installation :

```
conda install -c anaconda jupyter
```

3. Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.⁹

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

Installation:

```
conda jupyter conda install -c conda-forge matplotlib
```

⁸ [NumPy documentation — NumPy v1.23 Manual](#)

⁹ [Matplotlib — Visualization with Python](#)

```
conda install -c conda-forge/label/testing/gcc7 matplotlib
conda install -c conda-forge/label/testing matplotlib
conda install -c conda-forge/label/cf202003 matplotlib
conda install -c conda-forge/label/matplotlib_rc matplotlib
conda install -c conda-forge/label/gcc7 matplotlib
conda install -c conda-forge/label/broken matplotlib
conda install -c conda-forge/label/broken-test matplotlib
conda install -c conda-forge/label/rc matplotlib
conda install -c conda-forge/label/cf201901 matplotlib
```

4. Cdlib

is a Python software package that allows to extract, compare and evaluate communities from complex networks.

The library provides a standardized input/output for several existing Community Discovery algorithms. The implementations of all CD algorithms are inherited from existing projects.¹⁰

Installation:

```
conda config --add channels giuliorossetti
conda config --add channels conda-forge
conda install cdlib
```

Note: the reason why we choose Linux is this specific library. It's easier to install on Linux and MacOS. And all steps mentioned above are required to be able to execute our project.

3 Execution

In our project we made sure the execution is as simple as possible. After all required libraries are installed all we need to do is to run the python main page.

¹⁰ [CDlib - Community Discovery Library — CDlib - Community Discovery library](#)

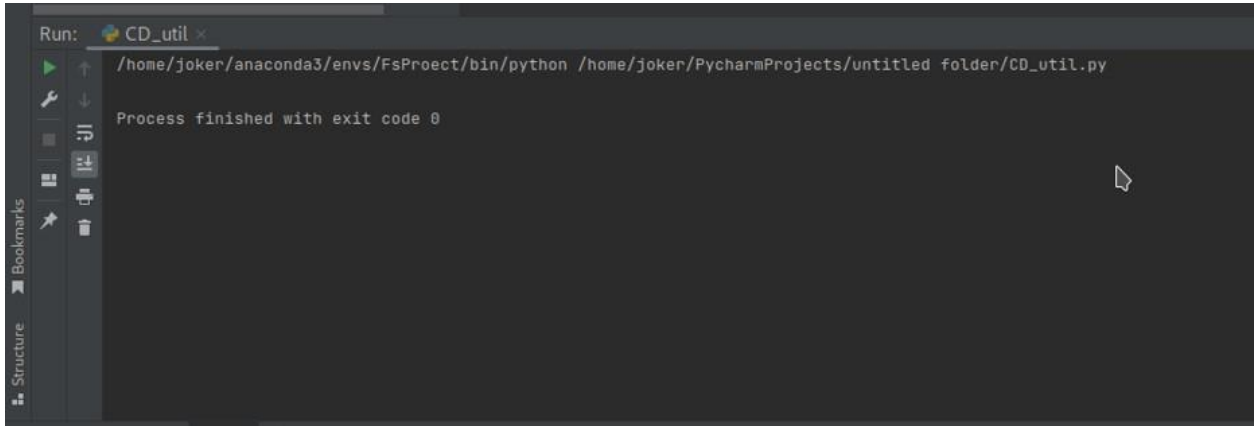


Figure 10. Project execution

To run the Jupyter code source you need to lunch Jupyter throw Anaconda navigator. Then you can simply run it (see fig 11).

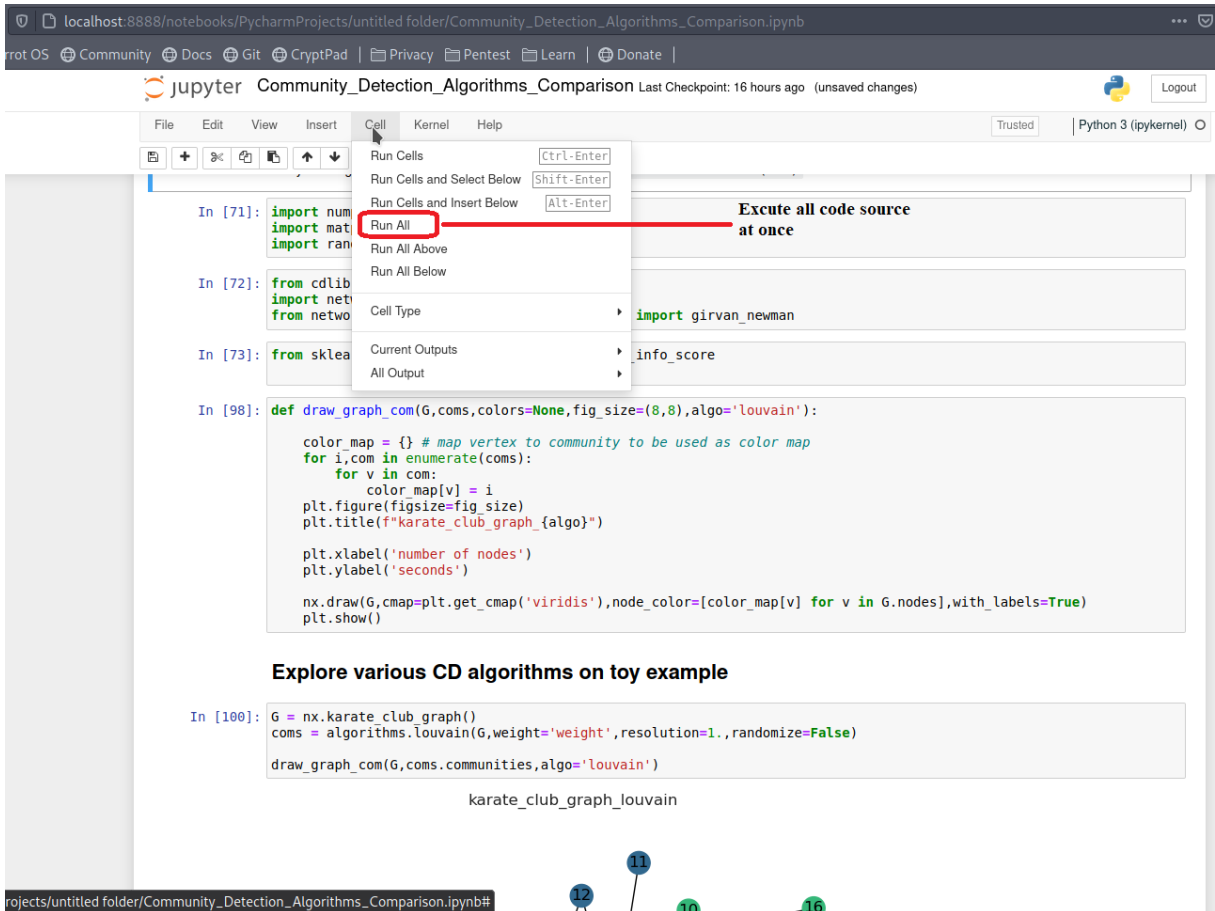


Figure 11. Jupyter code source Run

4 Results

4.1 Karate_club

The dataset contains social ties among the members of a university karate club collected by Wayne Zachary in 1977.

Network Data Statistics:

Nodes 34

Edges 78

Density 0.139037

4.1.1 Louvain algorithm

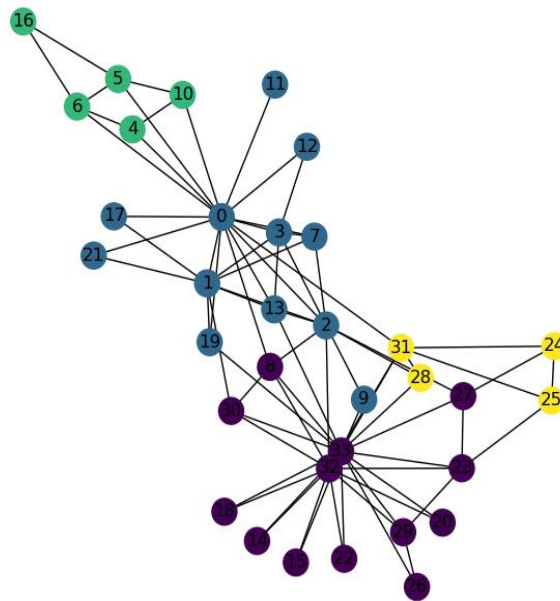


Figure 12. Louvain algorithm (karate club)

4.1.2 CNM algorithm

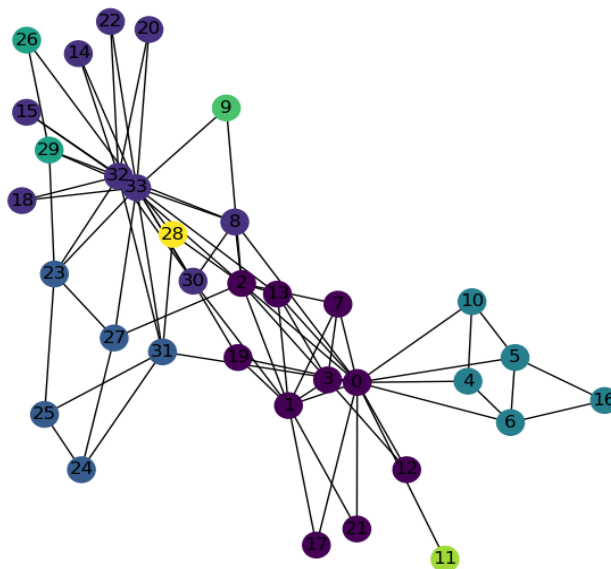


Figure 13. CNM algorithm (karate club)

4.1.3 Leiden Algorithm

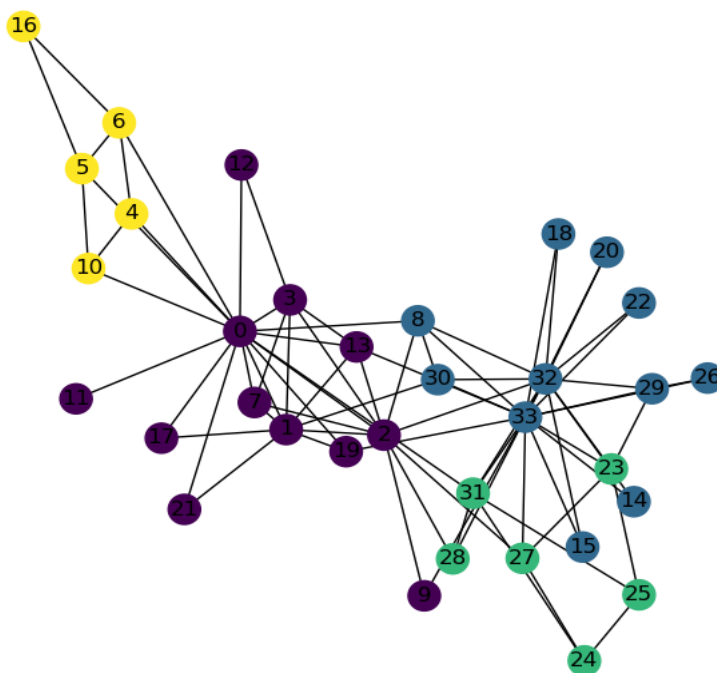


Figure 14. Leiden Algorithm (karate club)

4.1.4 Paris Algorithm

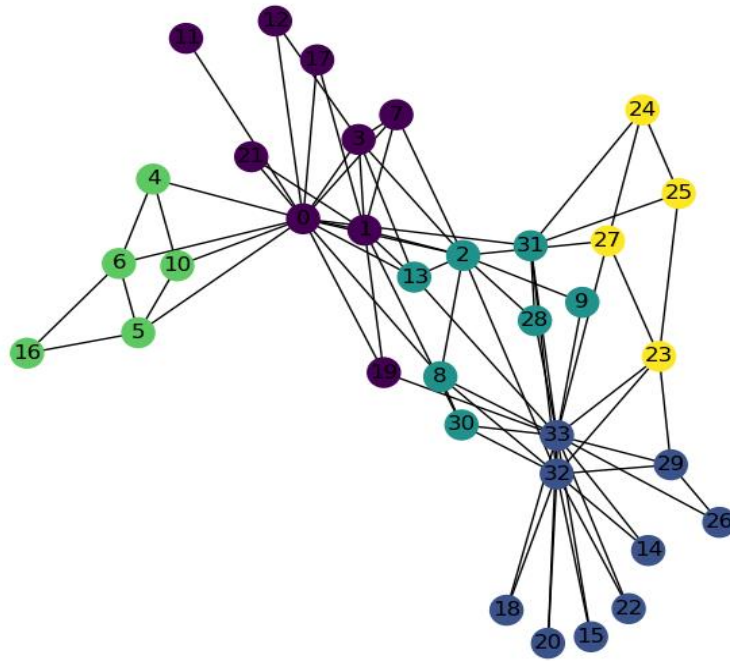


Figure 15. Paris Algorithm (karate club)

4.1.5 Eigenvector Algorithm

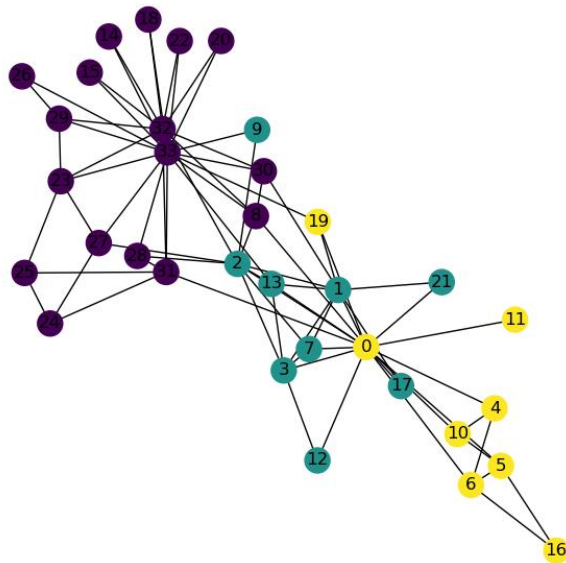


Figure 16. Eigenvector Algorithm (karate club)

4.2 LFR benchmark

LFR IS an algorithm that generates benchmark networks (artificial networks that resemble real-world networks). They have a priori known communities and are used to compare different community detection methods.

This dataset is a collection of undirected and unweighted LFR benchmark graphs as proposed by Lancichinetti et al.

4.2.1 CNM

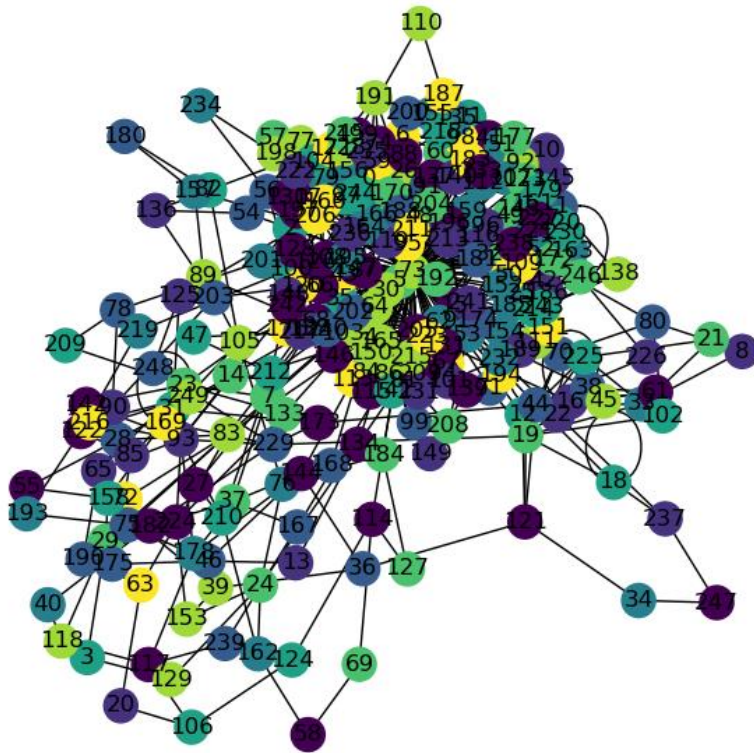


Figure 17. LFR benchmark for CNM

4.2.2 Leiden

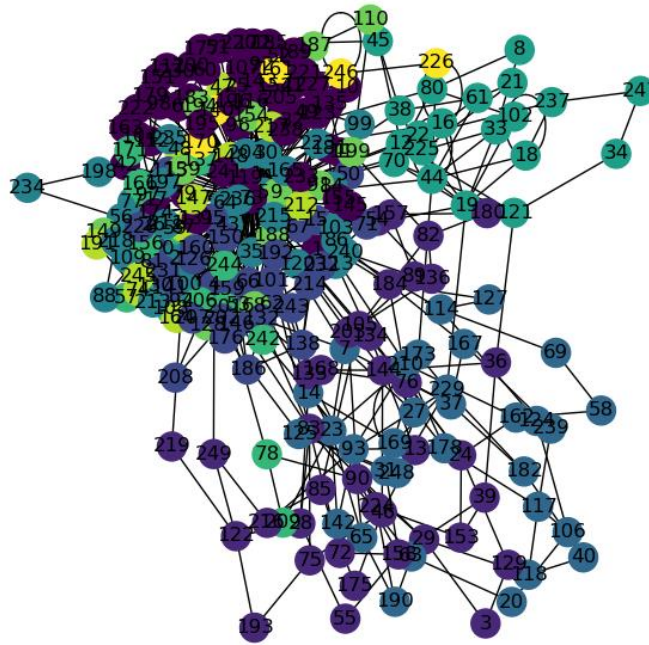


Figure 18. LFR benchmark for Leiden

4.2.3 Louvain

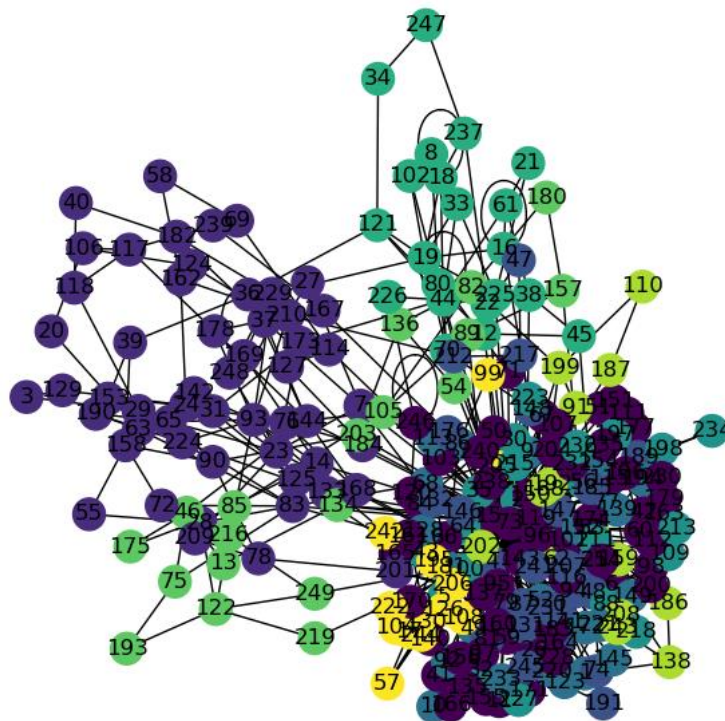


Figure 19. LFR benchmark for Louvain

4.2.4 Paris

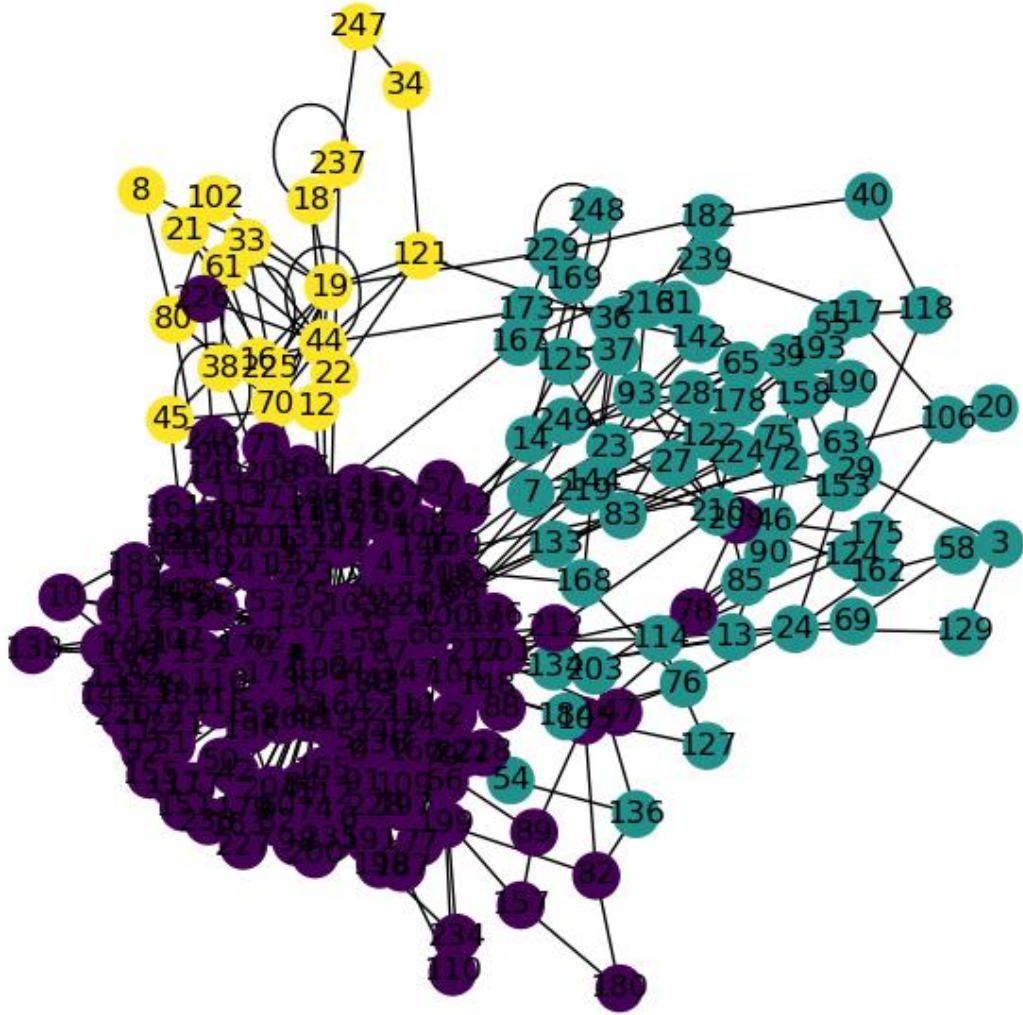


Figure 20. LFR benchmark for Louvain

4.2.5 Eigenvector

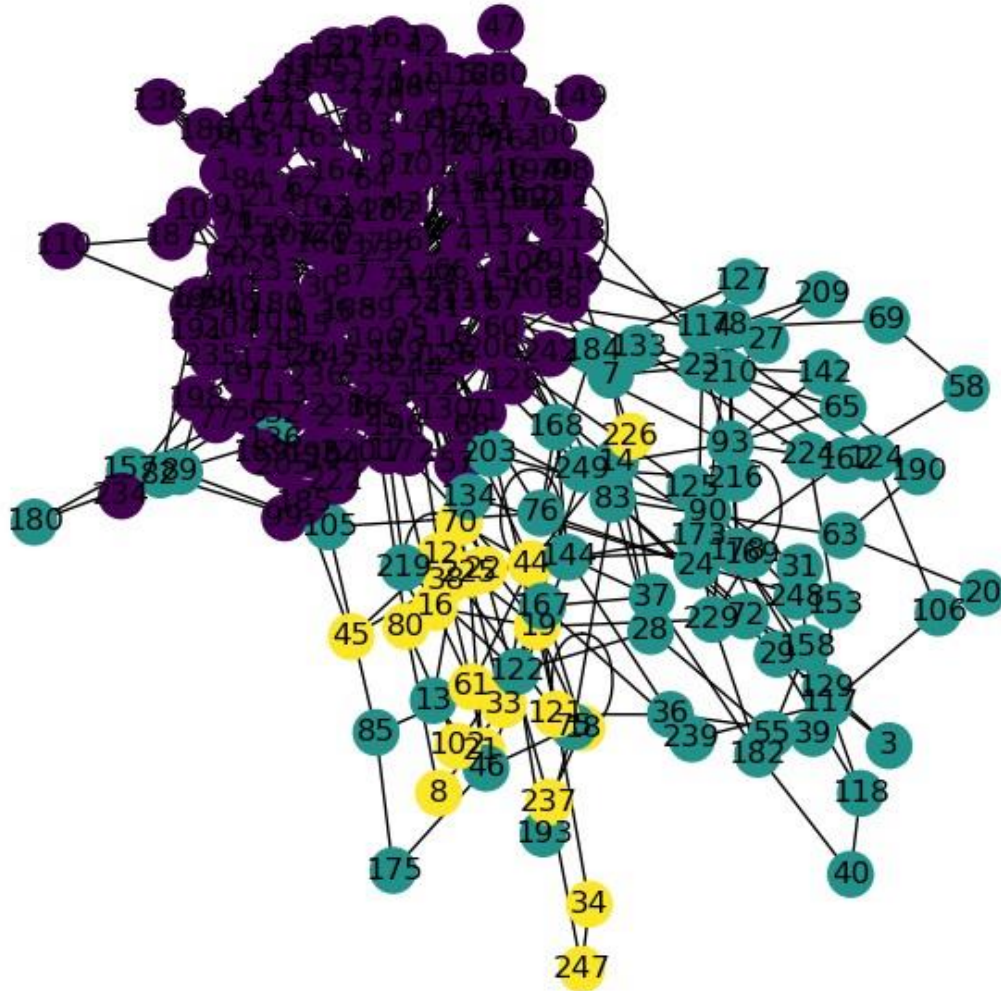


Figure 21. LFR benchmark for Eigenvector

4.3 generate LFR using uniform distribution in parameters

The results of the methods generating LFR using uniform distribution in parameters applied on a set of networks composed of 250 to 10000 nodes. Where we compare using two factors time and information.

Execution parameters

Generated graphs: low = 250, high = 10000.

Mean time limit: 250

Std (standard time) limit: 250

Mean NMI limit: 0.9

Std NMI limit: 0.9

4.3.1 CNM

The results of the CNM method generating LFR using uniform distribution in parameters applied on a set of networks composed of 250 to 10000 nodes are given in the figure 21.

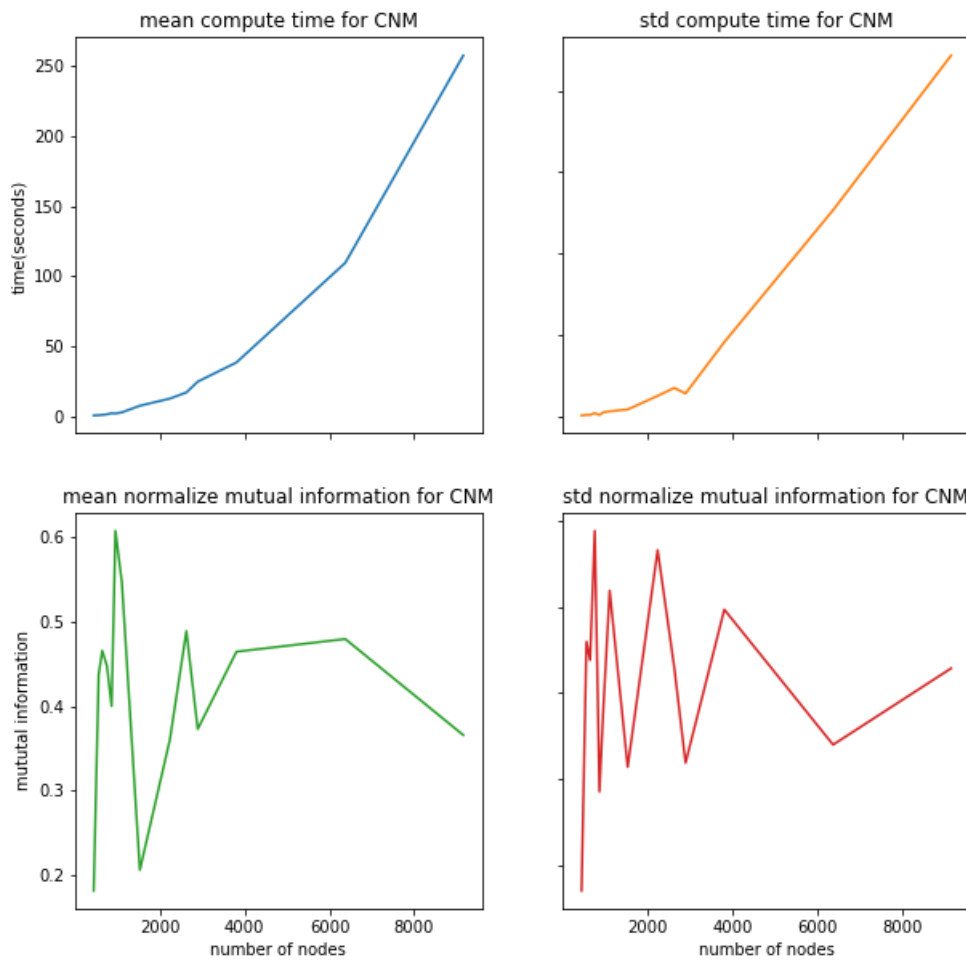


Figure 22. LFR benchmark for CNM using uniform distribution in parameters

The increase of nodes for CNM method results in an exponential increase of the time value for both mean time and standard time. on the other hand, the mean NMI presents two fast peaks (<2000 nodes) max and min. after that the method converges to a maximum. As far for std NMI presents instability.

4.3.2 Leiden

The results of the Leiden method generating LFR using uniform distribution in parameters applied on a set of networks are given in the figure 22.

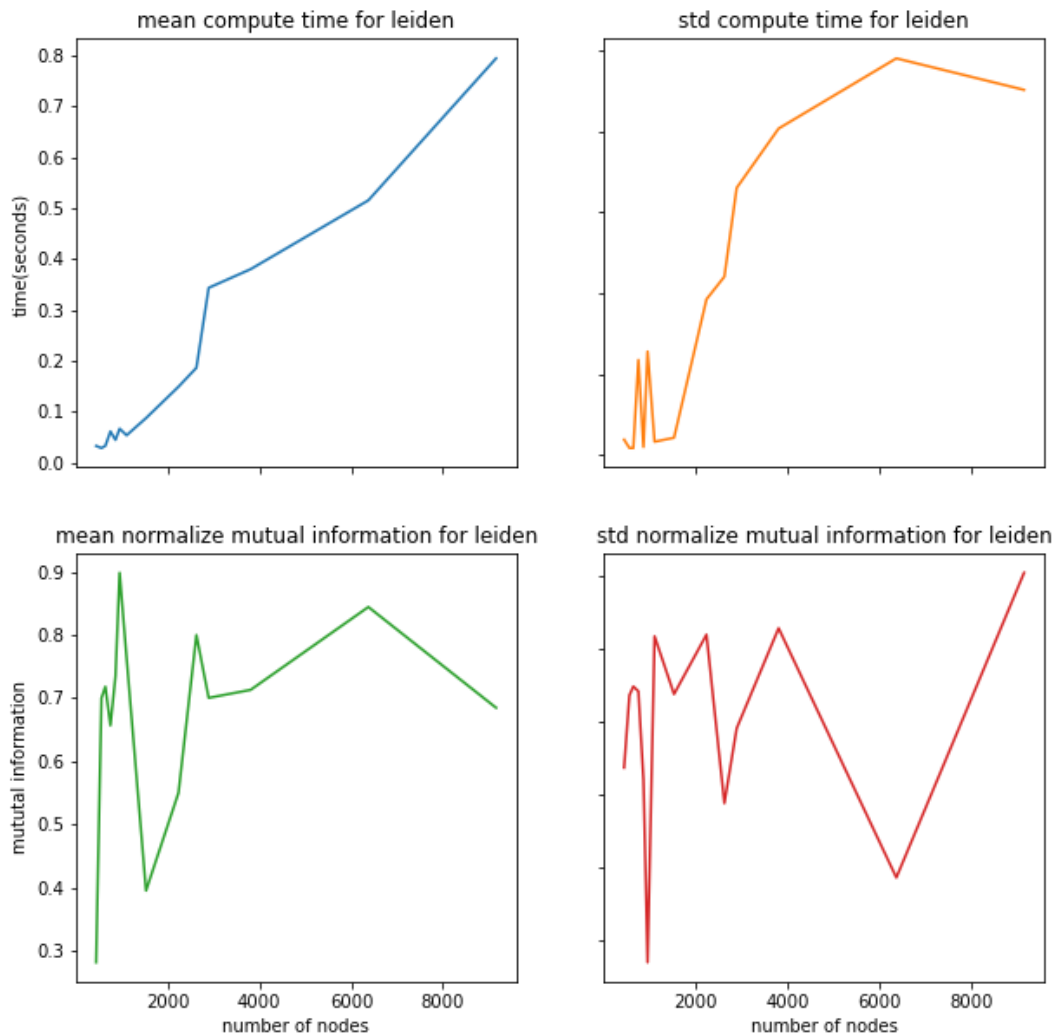


Figure 23. LFR benchmark for Leiden using uniform distribution in parameters

The increase of nodes in Leiden results in an exponential increase of the time value for mean time, but the Std compute time increases and converges to an acceptable maximum. on the other hand, the mean NMI converges to a maximum. Std time is more optimal than mean NMI. As far for std NMI presents instability.

4.3.3 Louvain

The results of the Louvain method generating LFR using uniform distribution in parameters applied on a set of networks composed of 250 to 10000 nodes are given in the figure 23.

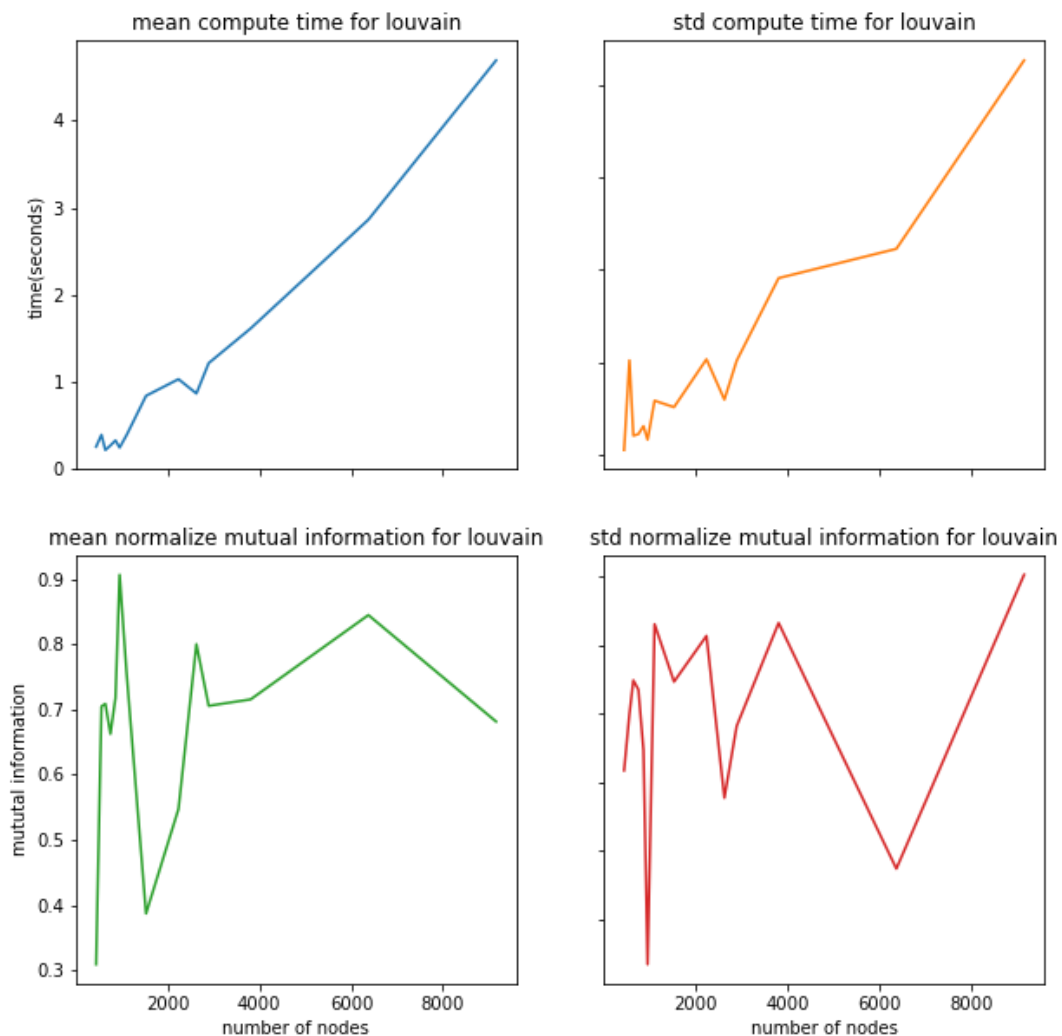


Figure 24. LFR benchmark for Louvain using uniform distribution in parameters

The increase of nodes results in an exponential increase of the time value for both mean time and standard time. on the other hand, the mean NMI presents two fast peaks (<2000 nodes) max and min. after that the method converges to an acceptable maximum. As far for std NMI presents instability.

4.3.4 Paris

The results of the Paris method generating LFR using uniform distribution in parameters applied on a set of networks are given in the figure 24.

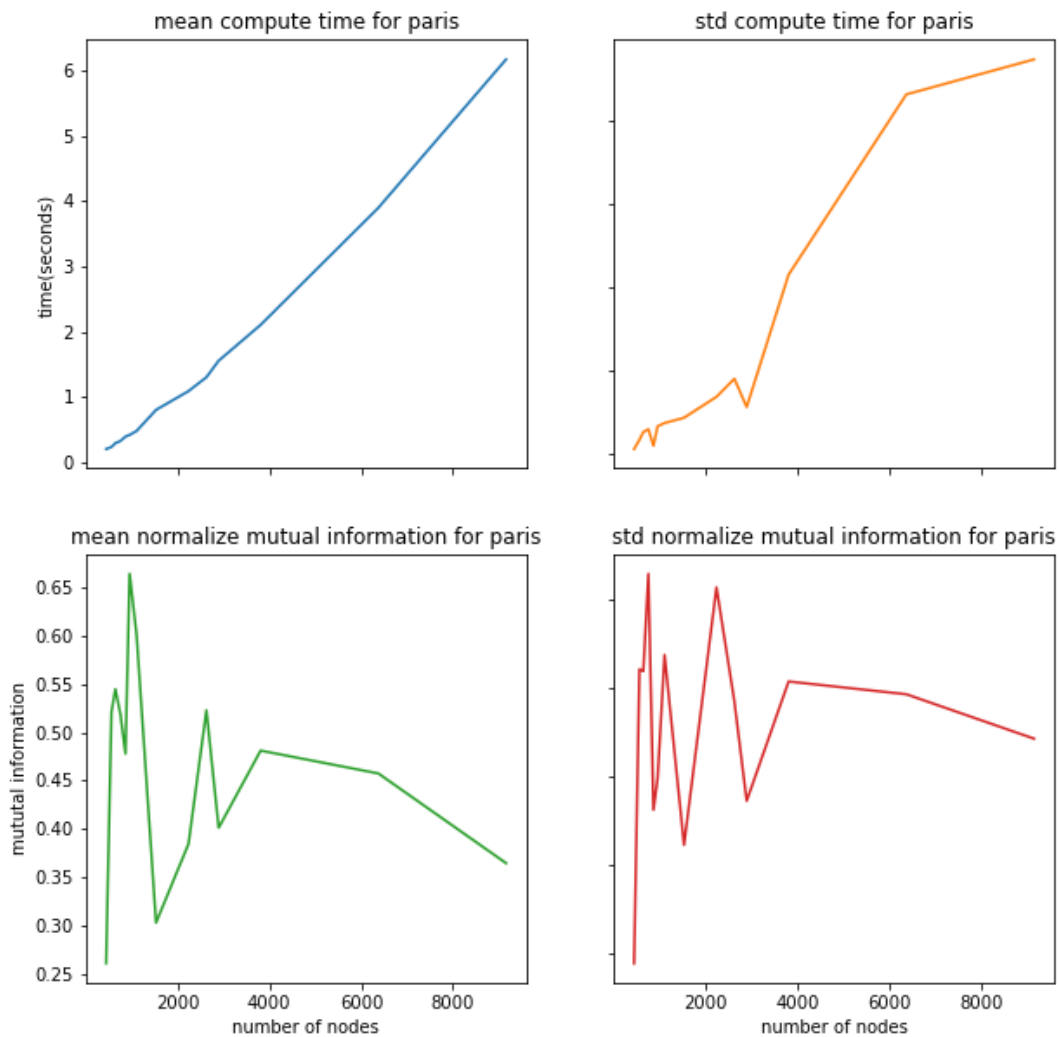


Figure 25. LFR benchmark for Paris using uniform distribution in parameters

The increase of nodes for Paris method results in an exponential increase of the time value for both mean time and standard time. on the other hand, both mean time and standard time in mutual information converges to an acceptable maximum

4.3.5 Eigenvector

The results of the Eigenvector method generating LFR using uniform distribution in parameters applied on a set of networks composed of 250 to 10000 nodes are given in the figure 25.

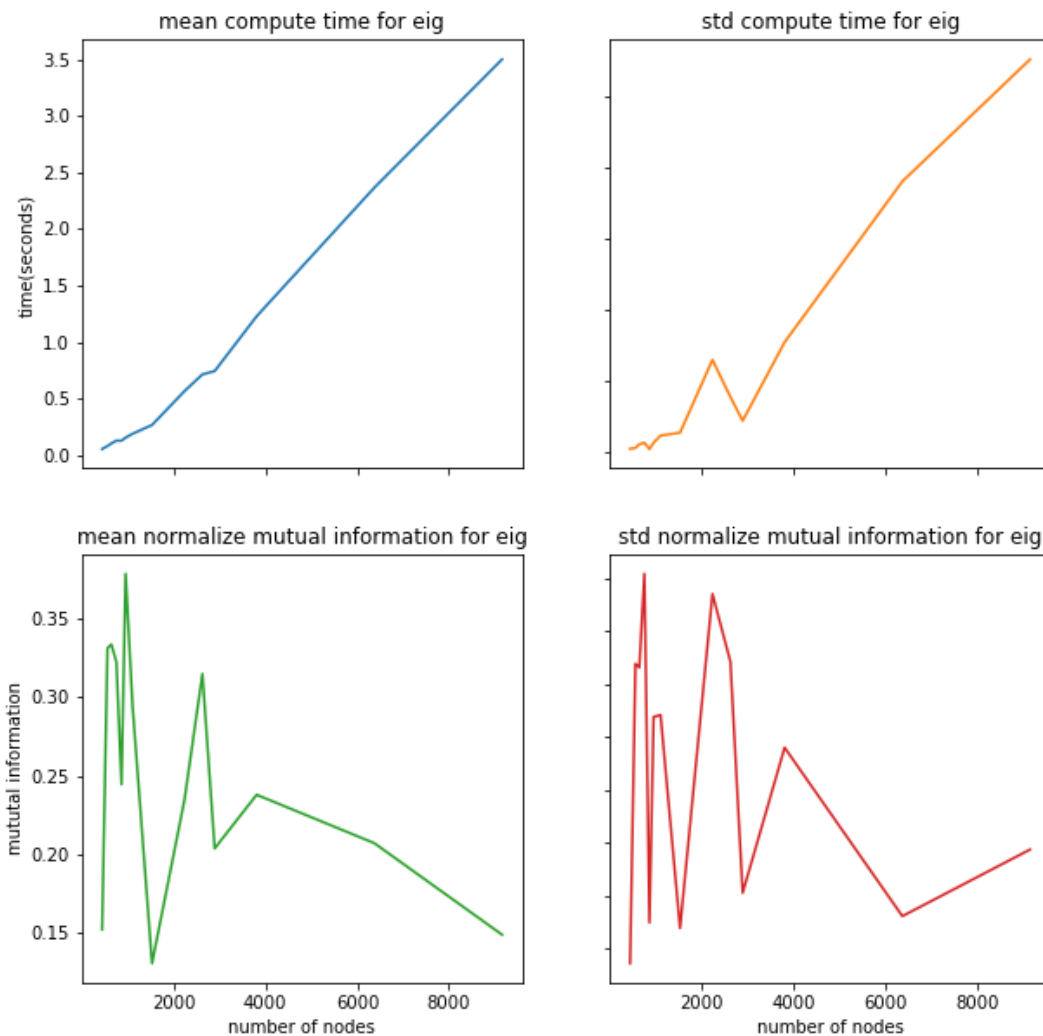


Figure 26. LFR benchmark for Eigenvector using uniform distribution in parameters

The increase of nodes for Eigenvector method results are not satisfied neither for time nor information.

4.4 Results of comparison

We distinguish two types of results obtained by the above-mentioned algorithms: results that converge (the existence of an acceptable maximum) and results that are unstable (the divergence from the desired solution).

NMI (normalized Mutual Information)

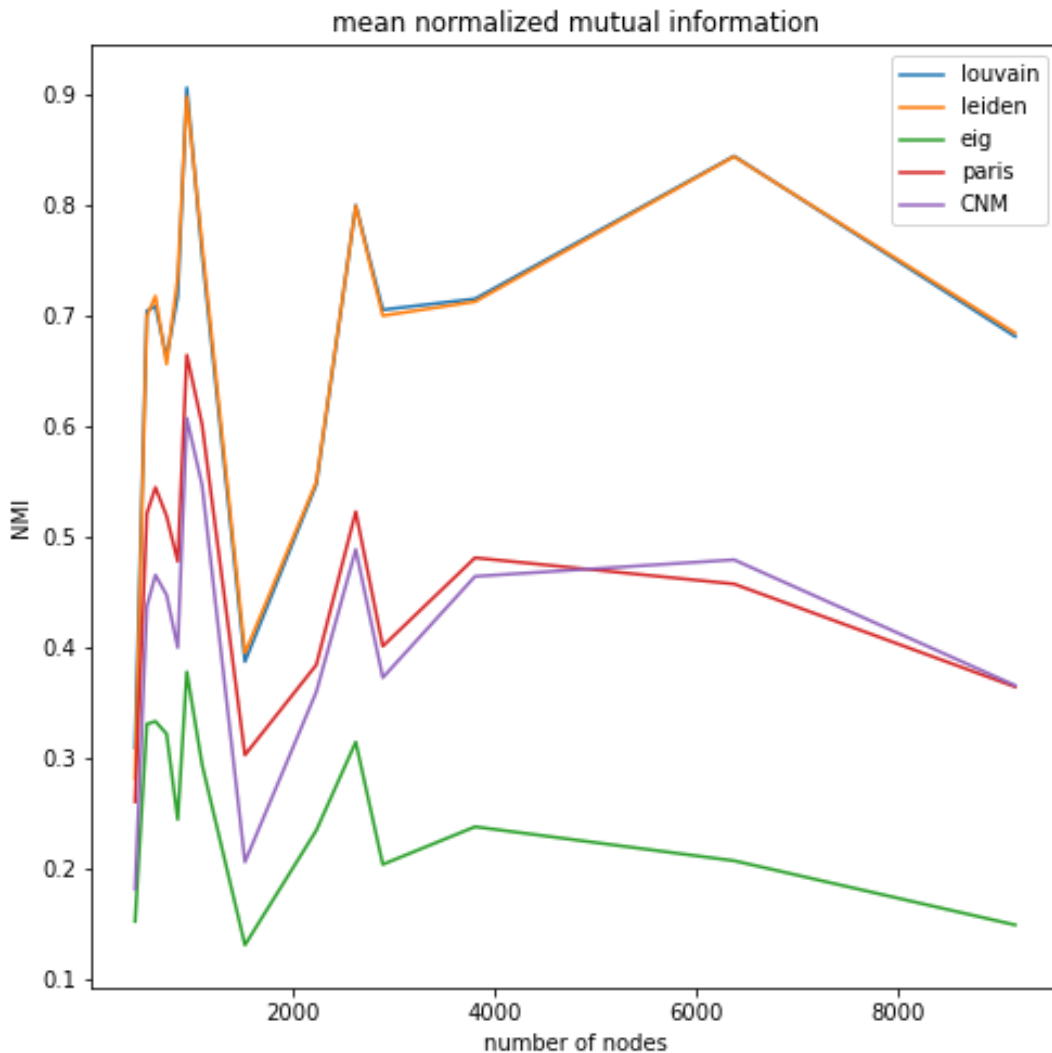


Figure 27. Mean NMI information

we notice that Louvain and Leiden converge to a local maximum (almost 7000 nodes). we note that after 7000 nodes these two methods diverge. however, CNM presents a stability of the solution (converges to a maximum at 4000 which remains stable until 7000) after the CNM diverges. Regarding the mean running time, all the methods are acceptable.

Compute time

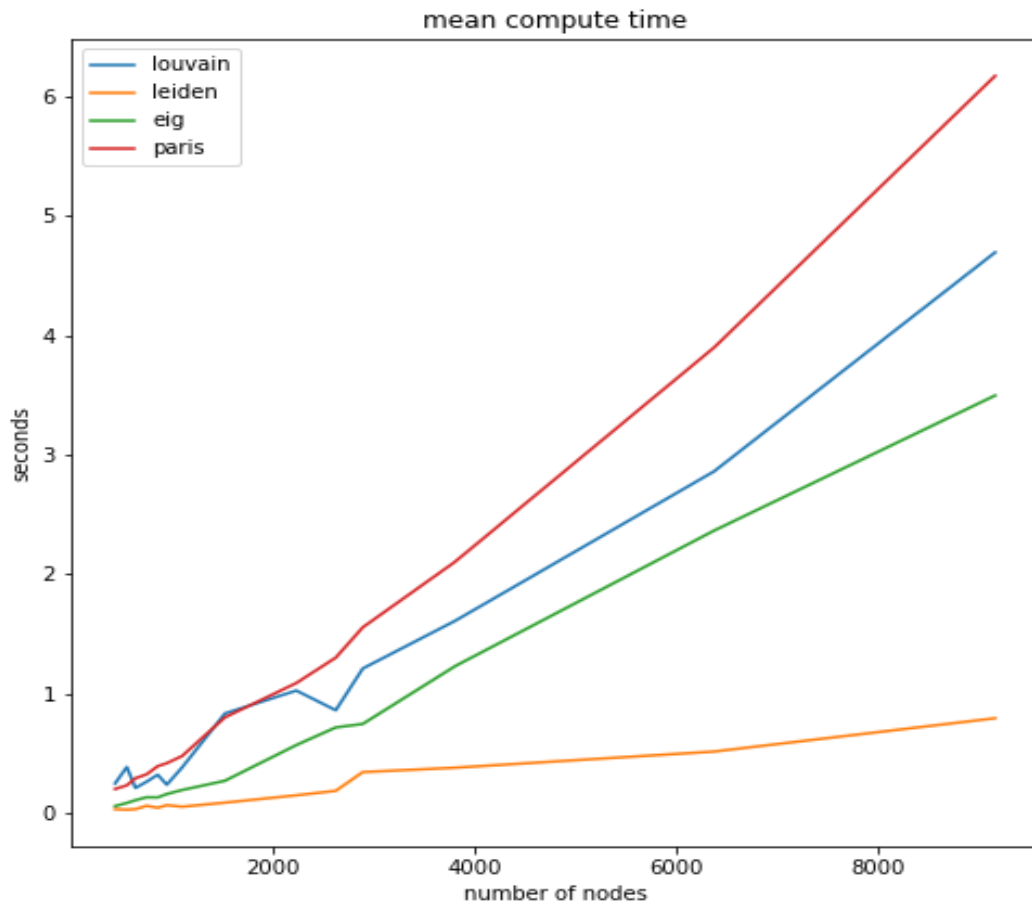


Figure 28. Mean compute time

We notice that Leiden method converges to a stable solution, in a short running time. All other methods didn't provide good solutions, in addition to that CNM took the longer running among all methods.

Mean running time & mean NMI (normalized Mutual Information)

	mean running time	mean NMI
louvain	1.0073463344573974 S	0.6948521613442102
leiden	0.19180045127868653 S	0.6952985766395726
CNM	47.10265590286255 S	0.4231966138797867
Eigenvector	0.856269947052002 S	0.2496902833722903
paris	1.0773232898712157 S	0.4477204113384476

Table 2. Mean running time & mean NMI

The final exclusion results for the mean running time for all five algorithms and the mean normalized mutual information for the same five algorithms.

5 Conclusion

The optimization strategy we have adopted in this work is not valid for networks consisting of a number of huge number of nodes. On the other hand, this strategy is better for small networks according to literature.

This does not preclude the existence of methods that give acceptable results for large networks. Such as CNM and Louvain for NMI, and Leiden for mean compute time.

The strategy we adopted confirmed that the complexity of some algorithms, which lead to considerable computation time, does not necessarily lead to a better solution. It is also important to remember that the computation time depends on the complexity (number of operations, search strategy) of the method.

CONCLUSION

When analyzing different networks, it may be important to discover communities inside them. Community detection techniques are useful for social media algorithms to discover people with common interests and keep them tightly connected. Community detection can be used in machine learning to detect groups with similar properties and extract groups for various reasons. For example, this technique can be used to discover manipulative groups inside a social network or a stock market.

To meet the requirements of community detection, different algorithms are applied in order to achieve the desired optimization results. The search for an optimal tell solution (the max in our case) has become then more than a necessity for this purpose, the detection of communities in a rather complex network is crucial, it requires the establishment of a more stable and efficient computational method to meet this need.

In this work, our first concern was to find a method to solve the problem of detection of communities, the solution is done by two methods (compute time, NMI). These methods are programmed on python (Andaconda) and are executed using Jupyter Notebook, and are tested on a variable network (the number of nodes goes from 250 to 10000).

In the second phase of our work, we presented the solution of the problem of community detection using five different algorithms that are based on modularity. These methods, despite providing some acceptable solutions, they don't always respond to the desired results.

References

- [1] M. E. J. Newman, “The structure and function of complex networks,” 2003, doi: 10.48550/ARXIV.COND-MAT/0303516.
- [2] R. Shang, J. Bai, L. Jiao, and C. Jin, “Community detection based on modularity and an improved genetic algorithm,” *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 5, pp. 1215–1231, Mar. 2013, doi: 10.1016/j.physa.2012.11.003.
- [3] F. Bonchi, C. Castillo, A. Gionis, and A. Jaimes, “Social Network Analysis and Mining for Business Applications,” *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–37, Apr. 2011, doi: 10.1145/1961189.1961194.
- [4] C. Sueur, A. Jacobs, F. Amblard, O. Petit, and A. J. King, “How can social network analysis improve the study of primate behavior?,” *Am. J. Primatol.*, vol. 73, no. 8, pp. 703–719, Aug. 2011, doi: 10.1002/ajp.20915.
- [5] Rémy Cazabet, “Détection de communautés dynamiques dans des réseaux temporels,” Université Paul Sabatier - Toulouse III, 2013.
- [6] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, “The Anatomy of the Facebook Social Graph,” 2011, doi: 10.48550/ARXIV.1111.4503.
- [7] M. S. Rahman, *Basic Graph Theory*, 1st ed. 2017. Cham: Springer International Publishing : Imprint: Springer, 2017. doi: 10.1007/978-3-319-49475-3.
- [8] V. Kostakos, “Temporal graphs,” *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, Mar. 2009, doi: 10.1016/j.physa.2008.11.021.
- [9] P. Hui and N. Sastry, “Real World Routing Using Virtual World Information,” in *2009 International Conference on Computational Science and Engineering*, Vancouver, BC, Canada, 2009, pp. 1103–1108. doi: 10.1109/CSE.2009.315.
- [10] M. E. J. Newman, “Fast algorithm for detecting community structure in networks,” *Phys. Rev. E*, vol. 69, no. 6, p. 066133, Jun. 2004, doi: 10.1103/PhysRevE.69.066133.
- [11] M. Girvan and M. E. J. Newman, “Community structure in social and biological networks,” *Proc. Natl. Acad. Sci. U.S.A.*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002, doi: 10.1073/pnas.122653799.
- [12] G. Palla, A.-L. Barabási, and T. Vicsek, “Quantifying social group evolution,” *Nature*, vol. 446, no. 7136, pp. 664–667, Apr. 2007, doi: 10.1038/nature05670.
- [13] J. Chen, D. Liu, F. Hao, and H. Wang, “Community detection in dynamic signed network: an intimacy evolutionary clustering algorithm,” *J Ambient Intell Human Comput*, vol. 11, no. 2, pp. 891–900, Feb. 2020, doi: 10.1007/s12652-019-01215-3.
- [14] K. Taha, “Static and Dynamic Community Detection Methods That Optimize a Specific Objective Function: A Survey and Experimental Evaluation,” *IEEE Access*, vol. 8, pp. 98330–98358, 2020, doi: 10.1109/ACCESS.2020.2996595.
- [15] A. Clauset, M. E. J. Newman, and C. Moore, “Finding community structure in very large networks,” *Phys. Rev. E*, vol. 70, no. 6, p. 066111, Dec. 2004, doi: 10.1103/PhysRevE.70.066111.
- [16] L. Danon, A. Díaz-Guilera, and A. Arenas, “The effect of size heterogeneity on community identification in complex networks,” *J. Stat. Mech.*, vol. 2006, no. 11, pp. P11010–P11010, Nov. 2006, doi: 10.1088/1742-5468/2006/11/P11010.
- [17] K. Wakita and T. Tsurumi, “Finding community structure in mega-scale social networks: [extended abstract],” in *Proceedings of the 16th international conference on World Wide Web - WWW '07*, Banff, Alberta, Canada, 2007, p. 1275. doi: 10.1145/1242572.1242805.

- [18] H. Shen, X. Cheng, K. Cai, and M.-B. Hu, “Detect overlapping and hierarchical community structure in networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 8, pp. 1706–1712, Apr. 2009, doi: 10.1016/j.physa.2008.12.021.
- [19] E. F. Qinna Wang, “Mining time-dependent communities,” 2010.
- [20] Z. Chen, K. A. Wilson, Y. Jin, W. Hendrix, and N. F. Samatova, “Detecting and Tracking Community Dynamics in Evolutionary Networks,” in *2010 IEEE International Conference on Data Mining Workshops*, Sydney, TBD, Australia, Dec. 2010, pp. 318–327. doi: 10.1109/ICDMW.2010.32.
- [21] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, “Analyzing communities and their evolutions in dynamic social networks,” *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 2, pp. 1–31, Apr. 2009, doi: 10.1145/1514888.1514891.
- [22] J. Xie, B. K. Szymanski, and X. Liu, “SLPA: Uncovering Overlapping Communities in Social Networks via A Speaker-listener Interaction Dynamic Process,” 2011, doi: 10.48550/ARXIV.1109.5720.
- [23] Qinna Wang, “Overlapping community detection in dynamic networks,” Ecole normale supérieure de lyon - ENS LYON, 2012.
- [24] Falkowski, Tanja & Barth, Anja & Spiliopoulou, Myra, “Studying Community Dynamics with an Incremental Graph Mining Algorithm,” vol. 5, Jan. 2008.
- [25] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *J. Stat. Mech.*, vol. 2008, no. 10, p. P10008, Oct. 2008, doi: 10.1088/1742-5468/2008/10/P10008.
- [26] M. E. J. Newman, “Analysis of weighted networks,” *Phys. Rev. E*, vol. 70, no. 5, p. 056131, Nov. 2004, doi: 10.1103/PhysRevE.70.056131.
- [27] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” 2003, doi: 10.48550/ARXIV.COND-MAT/0308217.
- [28] M. E. J. Newman, “Modularity and community structure in networks,” 2006, doi: 10.48550/ARXIV.PHYSICS/0602124.
- [29] C. S. Q. Siew, “Community structure in the phonological network,” *Front. Psychol.*, vol. 4, 2013, doi: 10.3389/fpsyg.2013.00553.
- [30] V. A. Traag, L. Waltman, and N. J. van Eck, “From Louvain to Leiden: guaranteeing well-connected communities,” *Sci Rep*, vol. 9, no. 1, p. 5233, Dec. 2019, doi: 10.1038/s41598-019-41695-z.
- [31] L. Waltman and N. J. van Eck, “A smart local moving algorithm for large-scale modularity-based community detection,” *Eur. Phys. J. B*, vol. 86, no. 11, p. 471, Nov. 2013, doi: 10.1140/epjb/e2013-40829-0.
- [32] the Graduate School of Information Science and Technology, University of Tokyo, Tokyo, Japan., N. Ozaki, H. Tezuka, and M. Inaba, “A Simple Acceleration Method for the Louvain Algorithm,” *IJCEE*, vol. 8, no. 3, pp. 207–218, 2016, doi: 10.17706/IJCEE.2016.8.3.207-218.
- [33] S.-H. Bae, D. Halperin, J. D. West, M. Rosvall, and B. Howe, “Scalable and Efficient Flow-Based Community Detection for Large-Scale Graph Analysis,” *ACM Trans. Knowl. Discov. Data*, vol. 11, no. 3, pp. 1–30, Apr. 2017, doi: 10.1145/2992785.
- [34] V. A. Traag, “Faster unfolding of communities: Speeding up the Louvain algorithm,” *Phys. Rev. E*, vol. 92, no. 3, p. 032801, Sep. 2015, doi: 10.1103/PhysRevE.92.032801.
- [35] A. Arenas, A. Fernandez, and S. Gomez, “Analysis of the structure of complex networks at different resolution levels,” 2007, doi: 10.48550/ARXIV.PHYSICS/0703218.

- [36] R. Lambiotte, J.-C. Delvenne, and M. Barahona, “Random Walks, Markov Processes and the Multiscale Modular Organization of Complex Networks,” *IEEE Trans. Netw. Sci. Eng.*, vol. 1, no. 2, pp. 76–90, Jul. 2014, doi: 10.1109/TNSE.2015.2391998.
- [37] M. E. J. Newman, “Community detection in networks: Modularity optimization and maximum likelihood are equivalent,” 2016, doi: 10.48550/ARXIV.1606.02319.
- [38] J. Reichardt and S. Bornholdt, “Statistical mechanics of community detection,” *Phys. Rev. E*, vol. 74, no. 1, p. 016110, Jul. 2006, doi: 10.1103/PhysRevE.74.016110.
- [39] F. Murtagh and P. Contreras, “Algorithms for hierarchical clustering: an overview,” *WIREs Data Mining Knowl Discov*, vol. 2, no. 1, pp. 86–97, Jan. 2012, doi: 10.1002/widm.53.
- [40] M. Fiedler, “Algebraic connectivity of graphs,” *Czech. Math. J.*, vol. 23, no. 2, pp. 298–305, 1973, doi: 10.21136/CMJ.1973.101168.
- [41] Per-Olof Fjällström, “Algorithms for Graph Partitioning: A Survey.,” no. Vol. 3(1998): nr 010, Aug. 1998, [Online]. Available: <http://www.ep.liu.se/ea/cis/1998/010/>
- [42] A. Pothen, H. D. Simon, and K.-P. Liou, “Partitioning Sparse Matrices with Eigenvectors of Graphs,” *SIAM J. Matrix Anal. & Appl.*, vol. 11, no. 3, pp. 430–452, Jul. 1990, doi: 10.1137/0611030.